



# TBWinPE/RE Builder

## Plugins Manual



TeraByte Unlimited  
Las Vegas, Nevada, USA  
<https://www.terabyteunlimited.com>

Revision: 2024-05-16  
Copyright © 2024 TeraByte, Inc.

|                                   |    |
|-----------------------------------|----|
| Overview .....                    | 3  |
| Downloading Plugins .....         | 4  |
| Creating Plugins .....            | 6  |
| Plugin File Format .....          | 8  |
| Variables & Strings .....         | 12 |
| Commands .....                    | 16 |
| Functions .....                   | 27 |
| Options .....                     | 31 |
| Developer Mode .....              | 33 |
| Plugin Settings .....             | 38 |
| Obtaining Technical Support ..... | 43 |
| Trademarks .....                  | 44 |

## Overview

TBWinPE/RE Builder Plugins can be used to add functionality to builds as well as managing different custom build configurations. New plugins can be created and used to handle specific changes or additions to builds.

### Features

- Add / remove / change files & folders
- Change options and registry settings
- Add WinPE / .msu packages
- Add drivers
- Add compatible programs
- Handle build types and modes differently
- Run .cmd, .ps1, and .tbs scripts for advanced operations
- Download and update plugins from TeraByte's website
- and more...

### Plugin Folders & Data

Only plugin files in the build's **Plugins** folder will be found and be available for use when loading plugins. For plugins that utilize multiple files, placing them in a sub-folder is recommended so they don't conflict with any other plugins.

Plugin sub-folders that start with \_\_ (two underscore characters) are special and are managed differently than normal plugins:

- |                       |  |
|-----------------------|--|
| • Plugins\__Download  | Downloaded plugins                               |
| • Plugins\__TeraByte  | Installed downloaded plugins                     |
| • Plugins\__TestBuild | TestBuild plugin                                 |
| • Plugins\__TestReq   | TestReq plugin, test folders, test registry hive |

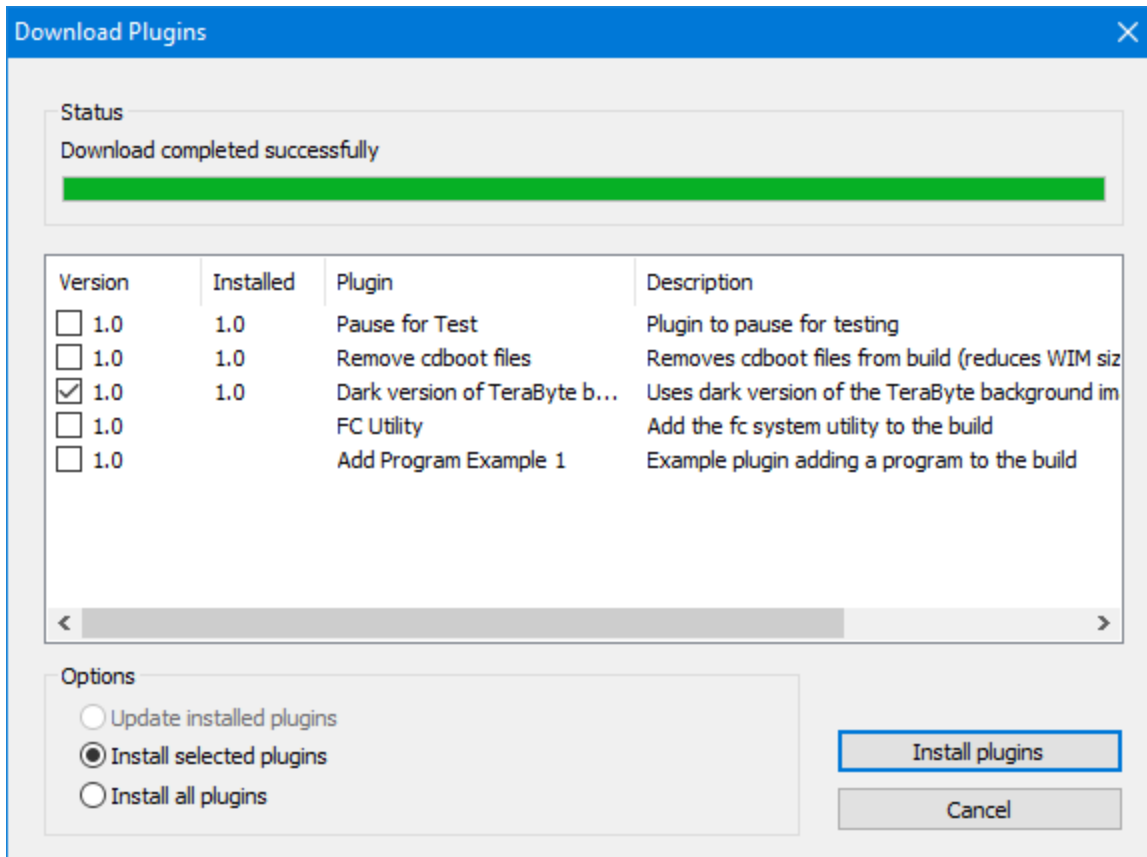
When creating a sub-folder name, don't start the name with two underscore characters.

To clear (reset) saved plugin data, delete the **PluginData** file in the build's **config** folder. The file will be recreated using default settings the next time plugins are loaded. Deleting this file will also reset all individual plugin options.

When processing a plugin, error messages and log output are saved in the plugin's log file (plugin's filename with .log appended) in the same folder as the plugin.

## Downloading Plugins

The Download Plugins dialog is accessible by clicking the **Download plugins** button on the **Plugins** tab in TBWinPE/RE Builder's Settings. If downloading is enabled, the current list of available plugins from TeraByte's website will be downloaded. Plugins can then be installed or updated as desired. If downloading is disabled you will only be able to install already existing plugins (those previously downloaded or included with the program).



Downloaded plugins are saved in the **Plugins\\_\_Download** folder. Installed plugins are located in the **Plugins\\_\_TeraByte** folder.

### Options

#### Update installed plugins

This option will be available if updated versions are available to install. The currently installed versions will be replaced with the newer versions. (This only affects plugins in the \_\_TeraByte sub-folder.)

#### Install selected plugins

Select to install just the the selected plugins (those checked in the list). If an installed plugin is selected it will be replaced. (This only affects plugins in the \_\_TeraByte sub-folder.)

### Install all plugins

Select to install all available plugins. Currently installed plugins will be replaced. (This only affects plugins in the \_\_TeraByte sub-folder.)

**Note:** The downloaded and installed plugins are separate from custom/user plugins. If you wish to modify one of these plugins you should copy it to the Plugins folder or a sub-folder so it won't be replaced when updating.

If a plugin has a long description or ReadMe file, it can be viewed by right-clicking on the plugin and selecting the desired action.

When selections are made, click the **Install plugins** (or **Update plugins**) button to install (or update) the plugins. Plugins will be reloaded.

## Creating Plugins

The Create Plugin dialog is accessible by clicking the **Create plugin** button on the **Plugins** tab in TBWinPE/RE Builder's Settings. This dialog provides an easy way to create a new plugin with basic options set. Once created, the plugin will be added to the installed list and can be edited as needed.

**Note:** Plugin files can also be created by copying an existing plugin file and editing it as desired.

The screenshot shows the 'Create Plugin' dialog box. It has a blue title bar with the text 'Create Plugin' and a close button (X). The dialog is divided into two main sections: 'Details' and 'Support Options'.  
In the 'Details' section, there are four input fields:

- 'Plugin Name': A text box containing 'New plugin name'.
- 'Author': An empty text box.
- 'Plugin File': An empty text box with a 'Browse...' button to its right.
- 'Description': A text box containing 'New plugin'.

The 'Support Options' section contains three lists of checkboxes:

- 'Build Modes':
  - TBWinRE
  - TBWinPE
- 'Architectures':
  - AMD64
  - x86
- 'Build Types':
  - Boot Media
  - IFW WinRE
  - IFW BootWIM
  - Update WinRE
  - Boot File

At the bottom of the dialog are two buttons: 'Create' and 'Cancel'.

### Details

#### Plugin Name

Enter a name for the plugin. Short names are recommended as they will display in the list without needing to resize it. This field is required.

#### Author

Enter the author of the plugin (e.g. name or company name).

## Plugin File

Enter the plugin filename. If just the filename is entered, it will be placed in the Plugins folder. To place the plugin in a sub-folder (recommended if it will use multiple files), click the **Browse...** button and select the folder and filename.

The filename and path will be validated before the plugin is created. The filename must have the .tbplg extension (it will be added if missing) and must not already exist.

## Description

Enter a short description for the plugin. This description is displayed in the list of plugins. A short description is recommended here (a longer description or ReadMe file can also be used if needed).

## Support Options

### Build Modes

Select the build modes that will be supported by the plugin. An unsupported build mode will display a warning when checking requirements.

### Architectures

Select the build architectures that will be supported by the plugin. An unsupported architecture will display a warning when checking requirements.

### Build Types

Select the build types that will be supported by the plugin. An unsupported build type will display a warning when checking requirements.

**Note:** The plugin should be scripted to check and handle the modes, architectures, and types as needed if required (for example, different files may need to be used in x86 vs. AMD64 builds).

When ready, click **Create** to create the plugin, which will then be added to the list of plugins. Plugins will be reloaded.

## Plugin File Format

The plugin file is a normal text file with the **.tbplg** extension. Specifically, a hybrid INI / code file: the beginning of the file is a normal INI type layout with sections followed by the plugin code script.

The contents of a minimal plugin file are shown below:

```
[TBPlugin]
Name=My Test Plugin
Author=
Description=My plugin for testing
Version=1.0
Website=
ReadmeFile=
WindowsVer=10.0
BuilderVer=1.38
WIMVer=10.0.0
TBScript=0
TBOSDTVer=
BuildModes=TBWinRE, TBWinPE
BuildTypes=BootMedia, IFWBootWIM
BuildArchitectures=AMD64

[Description]

[RequiredFiles]

[WinPEPackages]

[Code]

sub Process
    log "Plugin running..."
end sub
```

Lines that begin with a ; (semi-colon) are comments and will be ignored.

Details on the individual file sections are provided below.

### [TBPlugin]

This section contains the plugin's settings, details, and version requirements. The **Name** option must exist. **Description** and **Version** should exist. Other values are optional (defaults will be used if not specified).

This section must exist. Versions should be specified in **###.###** format (e.g. 10.0, 10.0.22631, etc.).

Section items:

**Name**=Test Plugin name for plugin



|   |   |
|---|---|
| <b>Description</b> =Plugin Description        | short description   |
| <b>Version</b> =1.0                           | version of plugin   |
| <b>Website</b> =https://www.websitedomain.com | website link (if any); must start with 'https://'                               |
| <b>ReadmeFile</b> =readme.txt                 | name of readme file in plugin's folder; should be text (.txt) file              |
| <b>WindowsVer</b> =10.0                       | minimum Windows version required (6.1 = Win7, 6.3 = Win8.1, 10.0 = Win10, etc.) |
| <b>BuilderVer</b> =1.38                       | minimum Builder version required  |
| <b>TBOSDTVer</b> =2.0.20                      | minimum TBOSDT version required (only applies if TBScript=1)                    |
| <b>TBScript</b> =1                            | enable if TBScript is required (.tbs)   |
| <b>WIMVer</b> =10.0.19000.23.5                | minimum WIM version required  |
| <b>BuildModes</b> =TBWinRE,TBWinPE            | build modes supported: TBWinPE, TBWinRE   |
| <b>BuildTypes</b> =BootMedia,IFWBootWIM       | build types supported: BootMedia, IFWBootWIM, IFWWinRE, UpdateWinRE, BootFile   |
| <b>BuildArchitectures</b> =AMD64,x86          | build architectures supported: AMD64, x86                                       |

**Note:** The plugin should be scripted to check and handle the required versions, modes, architectures, and types as needed (for example, different files may need to be used in x86 vs. AMD64 builds).

### [Description]

This section contains the long description text and can consist of multiple lines of text. This section is provided for plugins that need to provide more details than the short description field, but not enough to require a separate ReadMe file.

This section is optional.

Example:

```
[Description]
Plugin long description text goes here
and can use multiple lines.

Another paragraph of long description text.
```

### [RequiredFiles]

List of files required by the plugin (one per line). Use full drive letter path for system files (environment variables can be used). Files without a drive letter are assumed to originate in the plugin's folder.

The plugin will fail the requirements check if any files are missing.

This section is optional.

Example:

```
[RequiredFiles]
readme.txt
customappfiles.zip
```

## [WinPEPackages]

List of WinPE package names, one per line, no paths (with or without .cab extension). This section can be used when adding packages via the script (**WinPEPackages** subroutine) isn't necessary.

WinPE packages will only be included in TBWinPE builds. Packages should be listed in the order in which they should be installed (some packages have dependencies).

For builds using Windows 10 or higher, compatible .msu packages can also be added (specify the full path). Adding this type of package can substantially increase the time required to create the build as well as the build's final size. Additionally, due to the number of changes that can be made by these types of updates, thoroughly testing the build is recommended to ensure the desired functionality is retained. Installing MSU packages is a good example of why checking versions before adding can be helpful (for one version adding it may be required, for a later version the changes may already be included in the WIM).

**Note:** MSU packages will only be installed in TBWinRE builds if allowed (same as from the WinREPackages.txt file).

This section is optional.

Example:

```
[WinPEPackages]
WinPE-WMI
e:\MSUPackages\Security\update.msu
```

## [Code]

This is the plugin script code section and must be the last INI section in the file (after whichever of the above sections are used).

This section must exist.

Lines that begin with ; or // are comments. In a code line, text after // is a comment.

Commands are normally one per line. To continue a command line to the next line, end the line with \ and continue on the next line. The lines will be considered as a single line for processing.

Multiple commands can be placed on one line if separated with a ; character.

Commands, functions, keywords, and variable names are not case sensitive.

## Process Subroutine

This is the main plugin subroutine and must exist for each plugin.

Example:

```
sub Process
    log "Processing plugin..."
; ..
; ..
end sub
```

## WinPEPackages Subroutine

This subroutine is used to add WinPE packages that require scripting (for example, checking if supporting files exist or the WIM version is supported before adding). Alternatively, it can be used instead of the **[WinPEPackages]** section even for packages that are always added.

WinPE packages will only be included in TBWinPE builds. Packages should be listed in the order in which they should be installed (some packages have dependencies).

For builds using Windows 10 or higher, compatible .msu packages can also be added (specify the full path). Adding this type of package can substantially increase the time required to create the build as well as the build's final size. Additionally, due to the number of changes that can be made by these types of updates, thoroughly testing the build is recommended to ensure the desired functionality is retained. Installing MSU packages is a good example of why checking versions before adding can be helpful (for one version adding it may be required, for a later version the changes may already be included in the WIM).

**Note:** MSU packages will only be installed in TBWinRE builds if allowed (same as from the WinREPackages.txt file).

This subroutine is optional.

**Note:** The registry hives for the build's WIM file are not loaded when the **WinPEPackages** subroutine is called. Registry calls should not be made into the hive paths.

Example (adding one WinPE package and one .msu package):

```
sub WinPEPackages
    WinPEPackage "WinPE-HTA"
    WinPEPackage "e:\MSUPackages\Security\update.msu"
end sub
```

## Variables & Strings

Variables are created by assigning a value to them or specifying them to receive a return value. All variables are global.

Variable names must start with a letter (a to z) and can contain letters and numbers. Variable names are not case sensitive.

Examples:

```
installpath="@tbMountPath@\Program Files\MyProgram"
count=10
RegGetValue "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion" "EditionID" wineid
IniRead "@tbPluginPath\options.ini" "Options" "Height" "0" heightval
```

Variables can have sub-vars (e.g. **myvar.count**) and elements (e.g. **myvar[2]**). When using a variable, either a sub-var or element can be specified, but not both at the same time (**myvar.count[2]** and **myvar[2].count** are both invalid).

The first element index is 0. Elements must be added in order (0, 1, 2, etc.). Each variable has a **.size** sub-var that contains the number of elements. Elements can be added to the end by using the correct index value, **[ ]**, or **.size** (if the variable already exists).

Examples:

```
item[0] = "first"
item[1] = "second"

myvar[] = "first element"
myvar[] = "second element"
myvar[myvar.size] = "third element"
```

Sub-vars/elements of sub-vars/elements can't be accessed directly (as mentioned above), but are available internally. Assign to a new variable if access is needed.

Variables can be used inside strings by surrounding them with the **@** character or they can be separated with a space or comma when combining into a string. Note that if an element is used inside a string, the element index must be a literal number and no spaces can be used (e.g. **@myvar[2]@** is supported, **@myvar[ 2 ]@** and **@myvar[myvarindex]@** are not supported). If the variable is not found (unassigned), the string will contain the text as entered.

For example, the output from both of these log commands is the same:

```
log "This is the build's path: @tbBuildPath@"
log "This is the build's path: " tbBuildPath
```

Strings should be surrounded by quotes. Inside quotes can be specified by using double quotes. Both double and single quotes are supported and can be switched between as needed.

Examples:

```
log "This is a string."
log 'This is a string.'
log "Left side ""quoted"" right side."
```

```
log 'Left side 'quoted' right side.'
log "Left side 'quoted' right side."
log 'Left side "quoted" right side.'
```

The starting index for strings is 0 (e.g. **Chr(0,name)** returns the first character of **name**).

Where supported, strings can be combined by specifying consecutive strings. Variables and function results can also be included (values will be converted to strings, if applicable). In addition to variable assignment, the **Log** and **WriteTextFile** commands support combined strings.

```
myvar = "part 1", "-", "part 2" // 'part 1-part 2'
myvar=3
myvar = "myvar: " myvar // 'myvar: 3'
name = "John Smith"
myvar = "First name: " left(name,4) ", Last name: " mid(name,5)
// 'First name: John, Last name: Smith'
```

## Built-in Variables

The following variables are built-in and available for use unless otherwise noted. All of these variables are read-only.

|                                |   |
|--------------------------------|---|
| <b>Error</b>                   | error value (number) -- returned by some commands                       |
| <b>SystemError</b>             | Windows system error code returned by some commands (number)            |
| <b>tbResult</b>                | default result var for some command results (various types)             |
| <b>tbWindowsVer</b>            | Windows version (string)  |
| <b>tbWindowsVer.Major</b>      | major version (number)  |
| <b>tbWindowsVer.Minor</b>      | minor version (number)  |
| <b>tbWindowsVer.Build</b>      | build version (number)  |
| <b>tbWindowsVer.ReleaseID</b>  | release ID (string)   |
| <b>tbWindowsVer.DisplayVer</b> | display version (string)  |
| <b>tbBuildPath</b>             | path to build's folder (the folder containing TBWinPE.exe) (string)     |
| <b>tbConfigPath</b>            | path to build's Config folder (string)                                  |
| <b>tbISOPath</b>               | path to build's ISO folder (string) -- (when applicable for build type) |
| <b>tbMountPath</b>             | path to build's Mount folder (string)                                   |
| <b>tbPluginPath</b>            | path to plugin's folder   |
| <b>tbRegSystem</b>             | registry path to WIM's system hive <sup>1</sup> (string)                |
| <b>tbRegSoftware</b>           | registry path to WIM's software hive <sup>1</sup> (string)              |

|  |  |
|--|--|
| <b>tbRegDefault</b>                      | registry path to WIM's default hive <sup>1</sup> (string)                                      |
| <b>tbMode</b>                            | build mode: "TBWinRE" or "TBWinPE" (string)  |
| <b>tbMode.ID</b>                         | build mode: 0=TBWinRE, 1=TBWinPE (number)  |
| <b>tbBuildType</b>                       | build type: "UpdateWinRE", "BootMedia", "BootFile", "IFWBootWIM", "IFWWinRE" (string)          |
| <b>tbBuildType.ID</b>                    | build type: 0=UpdateWinRE, 2=BootMedia, 3=BootFile, 4=IFWBootWIM, 5=IFWWinRE (number)          |
| <b>tbArch</b>                            | build architecture: "AMD64" or "x86" (string)  |
| <b>tbArch.ID</b>                         | build architecture: 0=x86, 1=AMD64 (number)  |
| <b>tbx64</b>                             | contains "64" for AMD64 builds, empty "" for x86 (string)                                      |
| <b>tbPaths</b>                           | contains paths to TeraByte programs and PE build source (number)                               |
| <b>tbPaths.IFW</b>                       | path to Image for Windows installation (string)  |
| <b>tbPaths.TBView</b>                    | path to TBView/TBIMount/TBIHD installation (string)  |
| <b>tbPaths.TBOSDT</b>                    | path to TBOSDT installation (string)   |
| <b>tbPaths.Source</b>                    | path to WinPE source files (ADK/AIK) (string)  |
| <b>tbBuilderVer</b>                      | version of TBWinPE/RE Builder (FILEEVER)   |
| <b>tbBuilderVer.Ver</b>                  | version of TBWinPE/RE Builder (string)   |
| <b>tbBuilderVer.v1 / .v2 / .v3 / .v4</b> | major, minor, build/sub-version1, sub-version 2 (numbers)                                      |
| <b>tbWimVer</b>                          | version of WIM used for build (string)   |
| <b>tbWimVer.Major</b>                    | major version (number)   |
| <b>tbWimVer.Minor</b>                    | minor version (number)   |
| <b>tbWimVer.Build</b>                    | build (number)   |
| <b>tbWimVer.SpBuild</b>                  | service pack build (number)  |
| <b>tbWimVer.SpLevel</b>                  | service pack level (number)  |
| <b>tbWimVer.FileVer</b>                  | internal file version <sup>2</sup> (FILEEVER)  |
| <b>tbWimVer.v1 / .v2 / .v3 / .v4</b>     | internal file version <sup>2</sup> : major, minor, build/sub-version1, sub-version 2 (numbers) |

<sup>1</sup> The registry path does not include the root (i.e. "HKLM\"). However, when used with the registry commands, the root will be inserted to expand the path as necessary. These will not be expanded when used in other areas (such as log output).

<sup>2</sup> These variables are not available when checking plugin requirements.



## Commands

This section covers variable assignment, calculations, and available plugin commands.

**Note:** In most cases, command parameters can be separated with a space. However, some separations require using a , (comma) or ( ) (parentheses) to get the correct results. Using commas is recommended unless creating simple joined strings.

## Assignment

Variables are assigned a value using the = (equals) character. To clear a variable, don't specify a value.

Examples:

```
var = "string value"
var = 3 + 5
var = var + 1
var = "text " varA " more text"

// clear value of a variable
var =
```

## Calculations

Order of operations is left to right. Use ( ) as necessary to group operations.

Basic operations are supported:

|                         |                                      |
|-------------------------|--------------------------------------|
| <b>+</b>                | add                                  |
| <b>-</b>                | subtract                             |
| <b>*</b>                | multiply                             |
| <b>/</b>                | divide                               |
| <b>&gt;</b>             | greater than                         |
| <b>&lt;</b>             | less than                            |
| <b>&gt;=</b>            | greater than or equal                |
| <b>&lt;=</b>            | less than or equal                   |
| <b>= / ==</b>           | equals (both formats supported)      |
| <b>!= / &lt;&gt;</b>    | not equal (both formats supported)   |
| <b>or /   </b>          | boolean or (both formats supported)  |
| <b>and / &amp;&amp;</b> | boolean and (both formats supported) |

Variable values will be converted as necessary (if possible), but some types are not directly comparable to other types or have values not valid for calculations (registry variables, for example).



If calculations are used along with other parameters to create a string (such as with the log command), make sure to separate the calculation from the following parameters with a comma (,) or surround the entire calculation with ( ). Additionally, the calculation should not be the first item when assigning a variable a string with calculations (a numeric or boolean value is expected in this case and will cause an error).

Examples:

```
log "text here" 2+2, " more text here"
log "text here" (2+2) " more text here"
myvar = "Value=" (2+2)
```

## Commands

This section lists details on the available plugin commands.

[ ] indicates an optional parameter.

Switches (indicated by the / character) must immediately follow the command and be before any required command parameters.

**Note:** Some commands return an error code in the **Error** variable and may return a system error code in the **SystemError** variable. Error value of 0 is success/non-error. **SystemError** values are standard Windows error code values.

---

### FileCopy source\_file dest\_file

Copies **source\_file** to **dest\_file**. Checks for valid path. Wildcards (\*?) can be used in filename of **source\_file**.

When copying source files using wildcards, **dest\_file** is assumed to be a folder. When copying a single file and **dest\_file** is a folder, the file will be copied into the folder using the same filename as the source file.

Example:

```
prgpath = "%ProgramFiles(x86)%\ABC"
FileCopy "@prgpath\abcprogram.exe" "@tbMountPath\Program
Files\ABC\abcprogram.exe"
```

---

### FileDelete [/a] del\_file

Deletes **del\_file**. Checks for valid path. Wildcards (\*?) can be used in filename.

Switch **/a** resets attributes to normal before deleting.

---

### FileRename filename newname

Renames a file or folder. Checks for valid path. Wildcards not supported.

---

### FindFile file\_path [result\_var]

Searches for matching files. Wildcards supported.

Returns results in elements of **TBResult** or **result\_var** (if specified).

Example:

```
FindFile "@tbMountPath@\\Windows\\a*.exe"
```

---

### FolderCreate folder\_path

Creates **folder\_path** (complete path to folder). Checks for valid path. Wildcards not supported.

---

### FolderCopy source\_folder dest\_folder

Copies **source\_folder** to **dest\_folder**, including sub-folders. Checks for valid path.

---

### FolderDelete [/s] del\_folder

Deletes **del\_folder**. Checks for valid path. Wildcards not supported.

Switch **/s** makes it recursive.

**Note:** Folder must be empty to delete without using **/s** switch.

---

### FindFolder folder\_path [result\_var]

Searches for matching folders. Wildcards supported.

Returns results in elements of **TBResult** or **result\_var** (if specified).

Example:

```
FindFolder "@tbMountPath@\\Program Files\\T*"
```

---

### RegCreateKey [/32] key\_string

Creates **key\_string**. Checks for valid path.

Switch **/32** uses WoW6432 access.

**Error** = 0 on success.

Example:

```
RegCreateKey "@tbRegSoftware@\\MyNewKey"
```

---

### RegOpenKey [/32] key\_string

Opens **key\_string**. Checks for valid path.

Switch **/32** uses WoW6432 access.

This command is really only useful to check if a key exists. **Error** = 0 on success.

---

### RegDeleteKey [/32] key\_string

Deletes **key\_string**. Checks for valid path.

Switch **/32** uses WoW6432 access.

**Error** = 0 on success.

Example:

```
RegDeleteKey "@tbRegSoftware@\MyNewKey"
```

---

### RegSetValue [/32] [/c] key\_string value\_name value\_data

Sets the value data (**value\_data**) for **key\_string value\_name**. Checks path is valid.

Switch **/32** uses WoW6432 access.

Switch **/c** creates the key if it doesn't exist.

**value\_data** should be a reg\_ variable type. REG\_SZ (string) is used if not reg\_ type.

Example:

```
mykey = "@tbRegSoftware@\TestKey"  
RegSetValue /c mykey "Description" "This is a test string."  
RegSetValue mykey "Size" RegDWORD(1)
```

---

### RegGetValue [/32] [/exp] key\_string value\_name [result\_var]

Returns registry data for **value\_name** in **TBResult** or **result\_var** (if specified).

Switch **/32** uses WoW6432 access.

Switch **/exp** will expand REG\_EXPAND\_SZ data.

Example:

```
mykey = "@tbRegSoftware@\TestKey"  
RegGetValue mykey "Description" myregvar  
log "Description text: " myregvar  
RegGetValue mykey "Size"  
log "Size: " tbResult
```

---

### RegDeleteValue [/32] key\_string value\_name

Deletes the registry value **value\_name** at **key\_string**. Checks path is valid.

Switch **/32** uses WoW6432 access.

Example:

```
RegDeleteValue "@tbRegSoftware@\TestKey" "Size"
```

---

### RegCopyTree [/32] [/32d] key\_string\_source key\_string\_dest

Copies the registry tree at **key\_string\_source** to **key\_string\_dest**. Checks path is valid.

Switch **/32** uses WoW6432 access.

Switch **/32d** uses WoW6432 access with **key\_string\_dest**.

---

### RegDeleteTree [/32] key\_string

Deletes the registry tree at **key\_string**. Checks path is valid.

Switch **/32** uses WoW6432 access.

---

### IniRead infile section varname [default\_value] [result\_var]

Reads standard INI type files. Checks path is valid.

Places result in **TBResult** or **result\_var** (if specified).

Example:

```
IniRead "@tbPluginPath\options.ini" "Options" "Count" "0" count  
log "Count: " count
```

---

### IniWrite infile section varname var\_value

Writes to standard INI type files. Checks path is valid.

Example:

```
IniWrite "@tbPluginPath\options.ini" "Options" "Count" "2"
```

---

### IniReadSection infile section [result\_var]

Reads a standard INI file section. Checks path is valid.

Places result in **TBResult** or **result\_var** (if specified).

Data format: The string value of **result\_var** is the section name. The section items are in the variable's elements as raw strings.

Example:

```
IniReadSection "myinifile.ini" "Options" data
// data      : "Options"
// data[0]   : "Path=C:\"
// data[1]   : "Maximize=1"
```

---

### IniWriteSection infile section section\_data

Writes a standard INI file section. Checks path is valid.

The section items of **section** are the elements of the **section\_data** variable as raw strings.

Example:

```
data = "Options"
data[0] = "Path=C:\"
data[1] = "Maximize=1"
IniWriteSection "myinifile.ini" "Options" data
```

---

### IniDeleteSection infile section

Deletes a standard INI file section. Checks path is valid

Example:

```
IniDeleteSection "myinifile.ini" "Options"
```

---

### If [not] (bool\_value) ... endif

Process **If** section.

Example:

```
b = true
if b
    log "Inside 'if' section"
endif
```

---

### While [not] (bool\_value) ... end while / wend

Process **While** section. **LoopMax** is used.

Example:

```
x = 0
while x < 5
```

```

    log "x is " x
    x = x+1
wend

```

---

### For `index_var, start, stop, [step] ... next`

Process **For** section. **LoopMax** is used. **step** is 1 if not specified. Negative step can be used.

**Note:** If negative step value is used, it must be separated from the **stop** value with a comma.

Example:

```

for i, 1, 5
    log "i value: " i
next

for i,10,0,-2
    log "i value: " i
next

```

---

### Call `subroutine_name`

Calls **subroutine\_name** as a subroutine.

Format: **sub sub\_name .... end sub**

Subroutines don't support parameters. If necessary, normal variables can be used (all are global).

Example:

```

sub Process
    call MySub
end sub

sub MySub
    log "Inside 'MySub' subroutine."
end sub

```

---

### Log `log_string log_string log_string ...`

Logs **log\_string(s)**... to plugins's log file. Functions and variables can also be used.

Example:

```

log "Build's path: " tbBuildPath
log "Build's path: @tbBuildPath@"

```

---

### ReadTextFile `[/trim:#] [/line:#] [/max:#] file_path [result_var]`

Reads a text file into elements of **TBResult** or **result\_var** (if specified).

Default max line count is 500.

Switch **/trim** trims CRLF, spaces, tabs from start/end of each line read. **/trim:1** just trims CRLF at end.

Switch **/line:#** starts reading at specified line number.

Switch **/max:#** maximum number lines to read. **/max** will read all lines.

### WriteTextFile [/n] [/c] file\_path write\_string

Writes **write\_string** to **file\_path** file. Checks if path is valid.

Switch **/c** will create the file (UTF-8, no-BOM is used).

Switch **/n** will add newline chars (CRLF) to end when writing.

### MenuItem [/plg] [/opt:#] name path [path64] [workingdir] [parameters] [icon]

Creates specified menu item in the TBLauncher.ini file (after existing items).

Switch **/plg** will configure to add to **Tools | Plugin Items** menu instead of main menu area.

A sub-menu can be created by using **|** in the name. For example: "MyProgram|Run My Program"

If a program has multiple menu options they can be grouped together this way.

Switch **/opt:#** will add an **Options=#** entry for the menu item.

Bit-flag: 0=auto-select (default), 1=Use ShellExecute, 2=Use CreateProcess, 8=Append to end of Tools menu (**/plg** items only).

Higher values override lower values (for bit-flags 1, 2).

Example:

```
MenuItem /plg "My Program" "X:\Program Files\MyProgram\myprog.exe"
```

### AutoRun [/opt:#] name path [path64] [workingdir] [parameters]

Creates specified **AutoRun** item in the TBLauncher.ini file (after existing items).

AutoRun items are launched on startup directly after InitScript.cmd.

Switch **/opt:#** will add an **Options=#** entry for the menu item.

Bit-flag: 0=auto-select (default), 1=Use ShellExecute, 2=Use CreateProcess, 4=CreateProcess (wait for finish)

Higher values override lower values.

**Note:** Care should be taken to avoid hanging the startup (process should finish quickly).

Example:

```
AutoRun "My Setup Program" "X:\Program Files\MyProgram\setup.exe" "" "" "/c"
```

---

### Break

Breaks from loop (**while**, **for**).

---

### Continue

Continues a loop (**while**, **for**).

---

### Return

Returns from a subroutine.

---

### Exit

Exits plugin.

---

### SetFileAttr [/r] [/h] [/s] file\_path

Sets the attributes for **file\_path**. Checks if path is valid.

Switch **/r**=ReadOnly, **/h**=Hidden, **/s**=System

If no attributes are specified, attributes are set to normal.

---

### WinPEPackage package\_name

Adds a WinPE package to the list for adding to build. **package\_name** can include .cab extension or not. Also supports .msu packages. Must specify the full path (e.g. E:\MSUPackages\mypackage.msu). MSU packages will only be installed in TBWinRE builds if allowed (same as from the WinREPackages.txt file).

**Note:** The registry hives for the build's WIM file are not loaded when the **WinPEPackages** subroutine is called. Registry calls should not be made into the hive paths.

**Note:** Only packages added in the **WinPEPackages** subroutine will be added to the build (packages added in the **Process** subroutine are ignored because package processing has already taken place).



**AddDriver [/s] driver\_folder****AddDriver [/s] driver\_file**

Copies a driver folder into the build. **driver\_folder** and **driver\_file** must specify full path.

If a folder is specified, the folder is copied. If an .inf file is specified, the folder containing the .inf file is copied.

Drivers are copied into ##### sub-folders in build's 'Drivers\\_\_PluginDrivers\_\_' folder. For example: 'Drivers\\_\_PluginDrivers\_\_\00000', then '..\00001', etc.

Switch **/s** also copies sub-folders.

**Note:** Drivers are added along with any other drivers included in the build after processing of all plugins has completed.

**ExportVar var\_name [export\_name]**

Adds variable **var\_name** to export list. Uses **export\_name** instead of default name (if specified). Variables in this list are exported as environment variables when using the **RunScript** command.

Default export names are as follows:

| Variable Name | Environment Variable Name |
|---------------|---------------------------|
| MyVar         | MyVar                     |
| MyVar.Path    | MyVar_Path                |
| MyVar[0]      | MyVar_0                   |

Plugin variables listed below are automatically exported:

| Variable Name       | Environment Variable Name              |
|---------------------|--|
| <b>tbMode</b>       | TBWinPE_Mode                           |
| <b>tbBuildType</b>  | TBWinPE_BuildType                      |
| <b>tbArch</b>       | TBWinPE_Arch                           |
| <b>tbx64</b>        | TBWinPE_x64                            |
| <b>tbBuildPath</b>  | TBWinPE_BuildPath                      |
| <b>tbConfigPath</b> | TBWinPE_ConfigPath                     |
| <b>tbMountPath</b>  | TBWinPE_MountPath (where appropriate)  |
| <b>tbISOPath</b>    | TBWinPE_ISOPath (where appropriate)    |
| <b>tbPluginPath</b> | TBWinPE_PluginPath (where appropriate) |

---

### RunScript [/show] [/max:#] [/log] script\_file

Runs a script (.cmd, .ps1, .tbs). Variables set to export will be exported as environment variables before running script.

Switch **/show** will show the window for the script that is run.

Switch **/max:#** will set maximum time in seconds to wait for script to finish (default 60 sec.). The process will be terminated if time expires. Does not apply if **/show** is used.

Switch **/log** will write console output to script's log file (ignored if **/show** used).

**tbResult** contains the process exit code.

The **Error** variable will be 0 if process ran successfully or 1 if there was an error. In case of error, **SystemError** is the system error code.

Example:

```
myname = "John Smith"
ExportVar myname
scriptfn = "@tbPluginPath\testscript.cmd"
WriteTextFile /c /n scriptfn "@echo off"
WriteTextFile /n scriptfn "set"
RunScript /log scriptfn
if (error=0); log "RunScript success"; else; log "RunScript error"; endif
```

---

### TakeOwnership file\_path

Takes ownership of **file\_path** file. Checks path is valid.

This is necessary for some files inside the WIM to allow editing/replacing, etc.

**Note:** Care should be taken to avoid changing ownership of the running Windows system's files.

---

### Sleep time\_ms

Sleeps for **time\_ms** in ms (milliseconds, 1000=1 second).

**Note:** Unnecessary use of this command is not recommended as it will slow down creation of the build. This command is provided for testing scenarios that require a delay.

---

### Unzip zip\_file dest\_folder

Unzips **zip\_file** to **dest\_folder** (must be .zip file). Checks **dest\_path** is valid.

The **Error** variable will be 0 if successful or 1 if there was an error. **SystemError** will be 2 if **zip\_file** wasn't found.

## Functions

This section covers available plugin functions. Functions return a specific value that can be used as a parameter or assigned to a variable.

Function parameters should be separated by commas ( , ). If a string value is needed as a parameter, it must be a single string (combining strings, strings & variables, etc., is not supported as they would be seen as multiple parameters).

---

### Len(string)

Returns length of **string** (number of characters in **string**).

---

### Left(string, count)

Returns string containing **count** of characters starting from index 0 of **string**.

---

### Right(string, count)

Returns string containing **count** of characters starting from last index of **string**.

---

### Mid(string, start\_index, [count])

Returns string containing **count** of characters starting from **start\_index** of **string**. If **count** is not specified (or zero) it will include to end of string.

---

### Chr(byte)

### Chr(index, var)

Returns a string containing the character for the **byte** value 0-255.

If **index** and **var** are specified, returns a string containing the character at **index** position of the string value of **var**.

---

### FindStr(string\_find, string\_search)

Returns the index for the found string **string\_find** in **string\_search**. Returns -1 if not found.

---

### RFindStr(string\_find, string\_search, [offset])

Returns the index for the found string **string\_find** in **string\_search**, searching from the right with an **offset** from the start of the string (if specified). Returns -1 if not found.

---

### GetFilename(path\_string)

Returns the filename (or folder name) portion of **path\_string**.

---

### GetPath(path\_string)

Returns the path portion of the **path\_string**.

---

### FileSize(file\_path)

Returns the size of the **file\_path** file if successful. If error, returns -1.

---

### FileExists(file\_path)

Returns true if file (or folder) **file\_path** exists, false if not found. Wildcards (\*?) supported in filename or folder if folder is last (e.g. "C:\TBWin?E").

---

### FolderExists(folder\_path)

Returns true if **folder\_path** is a folder and it exists, false if not found. Wildcards not supported.

---

### FileVersion(file\_path)

Returns a FILEVER variable containing the file version of **file\_path**.

Example:

```
fvPrg = FileVersion("@programpath\program.exe")
```

---

### IsSet(var)

Returns true if **var** is set to a value, false if not set. An unassigned variable will not cause an error when used with this function.

---

### RegVarType(reg\_var)

Returns the registry variable type of **reg\_var**.

0=not set or not a registry variable

1=reg\_str, 2=reg\_expstr, 3=reg\_multistr, 4=reg\_DWORD, 5=reg\_QWORD, 6=reg\_binary,  
7=reg\_none

---

### RegExpStr(string)

Returns a reg\_expstr variable set to **string**.

Example:

```
regExp = RegExpStr("%ProgramFiles%\MyPath\MyProgram.exe")
```

---

### RegDWORD(DWORD)

Returns a reg\_DWORD variable set to value of **DWORD**.

Example:

```
regDW = RegDWORD(1024)
```

---

### RegQWORD(QWORD) RegQWORD(lowpart,highpart)

Returns a reg\_QWORD variable set to **QWORD** or **lowpart/highpart**.

Example:

```
regQW = RegQWORD(16,1024)
```

---

### RegMultiStr([string],[string],...)

Returns a reg\_multistr variable containing the specified **string** items (accessible as elements).

Example:

```
regMulti = RegMultiStr("Item A","Item B","Item C","Item D")
```

---

### RegNone()

Returns a reg\_none variable.

---

### RegBinary(["hex"],[byte],[byte],...]

Returns a `reg_binary` variable containing the specified **byte** values. If **"hex"** is the first parameter, the bytes are decoded as two-char hex (0A, C2, etc.) without need for 0x prefix. This makes it easy to copy in registry exported binary data from a .reg file.

Bytes are accessible as elements.

Example:

```
regBin = RegBinary("hex", 1, 2, 3, 0A, FE)
```

---

### CompareFileVersion(file\_ver\_a, string\_ver/file\_ver\_b)

Returns number result for compare:

<0 = **file\_ver\_a** is less than specified **string\_ver/file\_ver\_b** (-1)

0 = **file\_ver\_a** is equal to specified **string\_ver/file\_ver\_b** (0)

>0 = **file\_ver\_a** is greater than specified **string\_ver/file\_ver\_b** (1)

**file\_ver\_a/b** is a FILEVER var (e.g. from **FileVersion()** function).

**string\_ver** is a string in the "0.0.0.0" format.

---

### CompareWindowsVersion(string\_ver)

Compares **string\_ver** against the running Windows version.

Returns a number result for compare:

<0 = Windows is less than specified version (-1)

0 = Windows is equal to specified version (0)

>0 = Windows is greater than specified version (1)

**string\_ver** uses "0.0.0.0" format (e.g. "10.0", "10.0.19045", etc.)

---

### CompareWimVersion(string\_ver)

Compares **string\_ver** against the WIM used for the build.

Returns a number result for compare:

<0 = WIM less than specified version (-1)

0 = WIM is equal to specified version (0)

>0 = WIM is greater than specified version (1)

**string\_ver** uses "0.0.0.0.0" format: major.minor.build.spbuild.splevel

## Options

This section covers available plugin options. These are set using the **Option** command and control aspects of the plugin when running.

Example:

```
Option ExpandVars false
```

---

### ExpandVars true/false

Expands inline variables in strings (format: **@varname@**). Enabled by default.

---

### ExpandEnvVars true/false

Expands inline environment variables in strings (format: **%varname%**). Enabled by default.

---

### DisableWoW true/false

Disables WoW folder redirection for certain path operations. Disabled by default (redirection will take place).

---

### CaseSensitive true/false

Sets case sensitivity when comparing strings. Disabled by default.

---

### LoopMax loop\_max

Sets the **LoopMax** value to **loop\_max**, at which point a loop (**while**, **for**) will abort to prevent infinite loops hanging the program and leaving the build in an undesirable state. The plugin script will exit with an error if the loop is aborted.

**loop\_max** must be between minimum (10) and maximum (1000000) values. Default is 100.

If it's necessary to use a higher value to perform an operation, it's recommended to reset to the default value after the loop has finished.

---

### PathCheck true/false

Checks path is valid for destination file and registry paths when enabled. Valid paths are those in the allowed locations. This check is to help prevent unwanted file and registry changes outside of the valid build locations. Reading/accessing other locations isn't blocked when this option is enabled. Enabled by default.

**Note:** Certain paths will still be checked even when this option is disabled.

**Note:** Paths in external scripts are not checked.

Valid paths: build's ISO folder (when applicable for build type), build's mount folder, plugin's folder

Valid registry paths: WIM's SYSTEM hive, WIM's SOFTWARE hive, WIM's DEFAULT hive

**Note:** When developing plugins, care should be taken to avoid making changes to system files/folders outside of the WIM file. For example, when changing a Windows registry setting for the build, make sure to change the setting in the WIM's mounted hive and not the running system's setting.



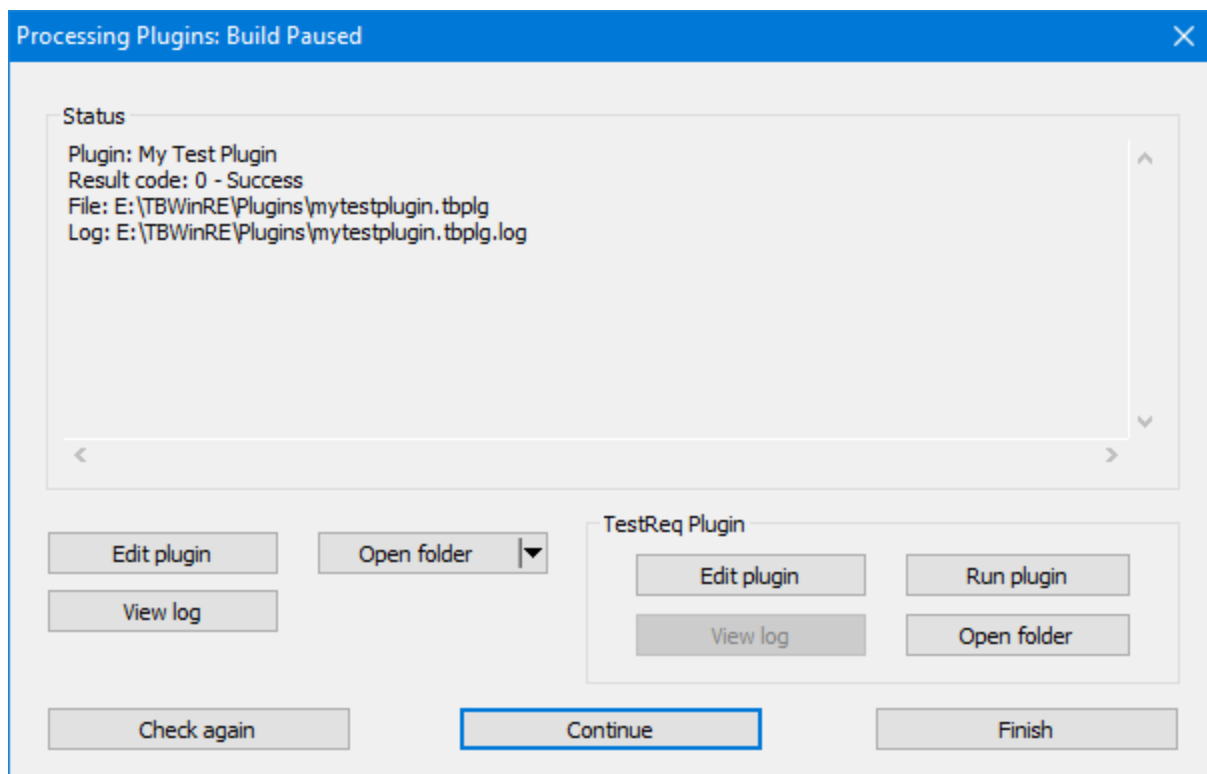
## Developer Mode

When Developer Mode is enabled, the build process will pause when checking plugin requirements and during the build step when plugins are being run. By default, the build will be paused only if a plugin error occurs. Enable the **Pause on requirements check success (DevMode)** and **Pause on build success (DevMode)** options to also pause on success (which can be helpful when testing and developing plugins).

**Note:** Developer Mode is automatically disabled for unattended builds.

During this paused state, the plugins can be stepped through one at a time to determine if the plugin is working as expected. Plugins can be edited and run again as necessary before continuing to the next plugin.

When developing or testing plugins, it may be helpful to have the build pause before the plugin being worked on is checked or run. To do this, make sure there is another plugin processed first and **Pause on build success (DevMode)** is enabled. For example, the **Pause for Test** plugin, which does nothing, can be used for this purpose. Then click **Continue** when ready to run the plugin being worked on.



The status of the plugin just processed is displayed. If an error occurred, the error will also be displayed.

The following actions are available while paused:

### Edit plugin

Opens the current plugin in the editor. The file can be edited as necessary and saved before checking or running again.

**Note:** It is not necessary to close the editor before checking or running the plugin again. You can keep the editor open for further changes and simply save the file prior to checking/running it.

### View log

Views the log output for the current plugin. This can be examined for diagnostic log entries, errors, etc.

### Open folder

Opens the current plugin's folder in File Explorer.

### Check again / Run again

Checks or runs the current plugin again.

**Note:** Some actions and/or results may be different on additional runs depending on how the plugin handles things.

### Continue

Continues to the next plugin. If the current plugin has any active warnings/errors, it will be disabled.

**Note:** During the build step, a failed plugin may not leave the build in the desired state. For example, files may be missing if the plugin aborted prior to copying them into the build.

### Finish

Disables Developer Mode for this build pass and finishes processing all remaining plugins.

## TestReq Plugin / TestBuild Plugin Sections

These sections of the dialog are available if the corresponding options are enabled in settings. The TestReq plugin is available during the requirements check. The TestBuild plugin is available during the build step.

### Edit plugin

Opens the TestReq/TestBuild plugin in the editor. The file can be edited as necessary and saved before running again.

**Note:** It is not necessary to close the editor before checking or running the plugin again. You can keep the editor open for further changes and simply save the file prior to checking/running it.

### View log

Views the log output for the TestReq/TestBuild plugin. This can be examined for diagnostic log entries, errors, etc.

### Run plugin

Runs the TestReq/TestBuild plugin. The TestReq plugin has access to test folders and registry hives to allow testing code and many operations without needing an active build step.

## Open folder

Opens the TestReq/TestBuild plugin's folder.

## Pause During Requirements Check

During the requirements check, the selected plugins are not run (only the requirements are checked). This step is used to make sure the support options are configured correctly (build modes supported, build types supported, required files exist, etc.). The plugins can be edited and rechecked as necessary.

The TestReq plugin (if enabled) can be run normally here as it has access to test folders and registry hives. This provides a method to easily test plugin code without needing to have a build step active. The test folders include the **ISO** and **mount** folders (accessible via the **tbISOPath** and **tbMountPath** variables). The **tbRegSystem**, **tbRegSoftware**, and **tbRegDefault** variables have access to a mounted hive for testing registry operations (mount location is HKLM\TBWinPE\_TestReq). For example, code can be created and tested to copy files to specific locations in the mount folder as well as configuring registry items (entries for added programs, configuration settings, etc.).

## Pause During Build Step

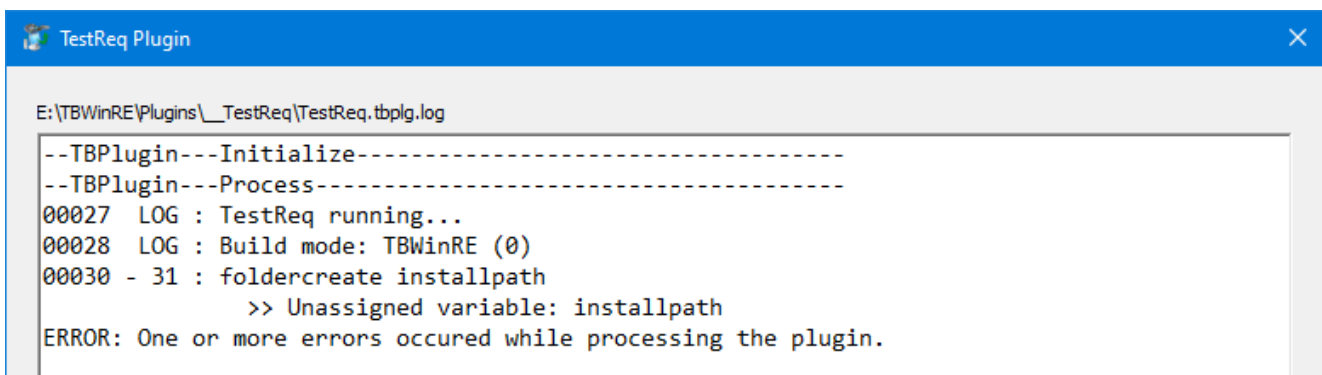
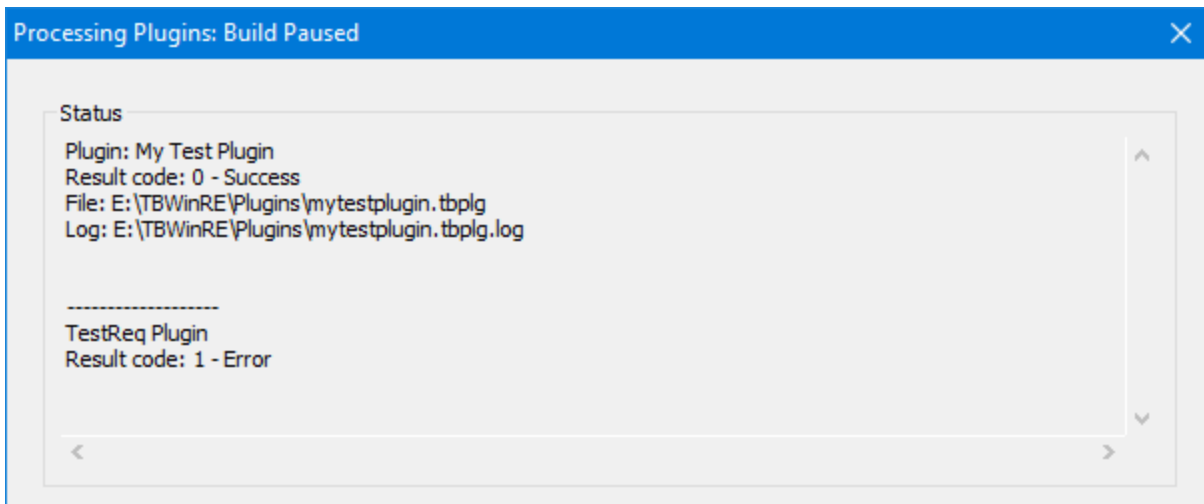
The pause during the build step is mostly the same as during the requirements check. However, there are some important differences:

- The TestBuild plugin is used instead of the TestReq plugin. TestBuild has access to the mounted WIM and registry hives.
- The selected plugins are run and have access to the build's mounted WIM and registry hives.

**Note:** Some operations are only supported prior to or after the pause point in the build. For example, adding WinPE packages has already taken place and adding drivers takes place after all plugins have run. If necessary, these operations would need to be tested in normal plugins and completed builds.

## Plugin Errors

Plugin errors will be displayed in the status area of the dialog as well as in the plugin's log file. In the example below, the TestReq plugin aborted with an error.



**Note:** If the **Show log after running test plugin (DevMode)** option is enabled, the TestReq log will automatically be displayed after running the plugin.

The log shows the line number (00030), the error code (31), the plugin script code where the error occurred, and the error description (unassigned variable, in this case).

In this example, the plugin code looks like this:

```

sub Process
    log "TestReq running..."
    log "Build mode: " tbMode " (" tbMode.ID ")"
    instalpath = "@tbMountPath@\Program Files\MyProgram"
    foldercreate installpath
    filecopy "@tbPluginPath@test\*.*" installpath
    log "Files copied"
end sub

```

The error is corrected by fixing the typo in the **installpath** variable name where the path is assigned to it.

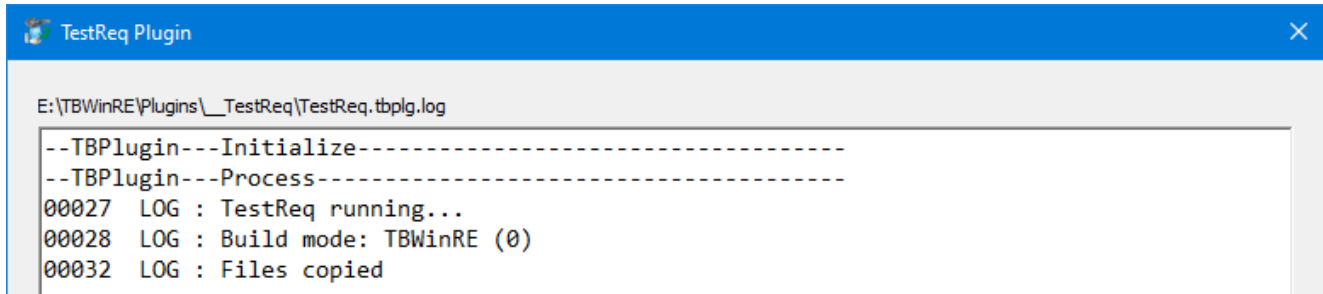
```

sub Process
    log "TestReq running..."
    log "Build mode: " tbMode " (" tbMode.ID ")"
    installpath = "@tbMountPath@\Program Files\MyProgram"
    foldercreate installpath
    filecopy "@tbPluginPath@test\*.*" installpath

```

```
    log "Files copied"  
end sub
```

Save and run the plugin again:



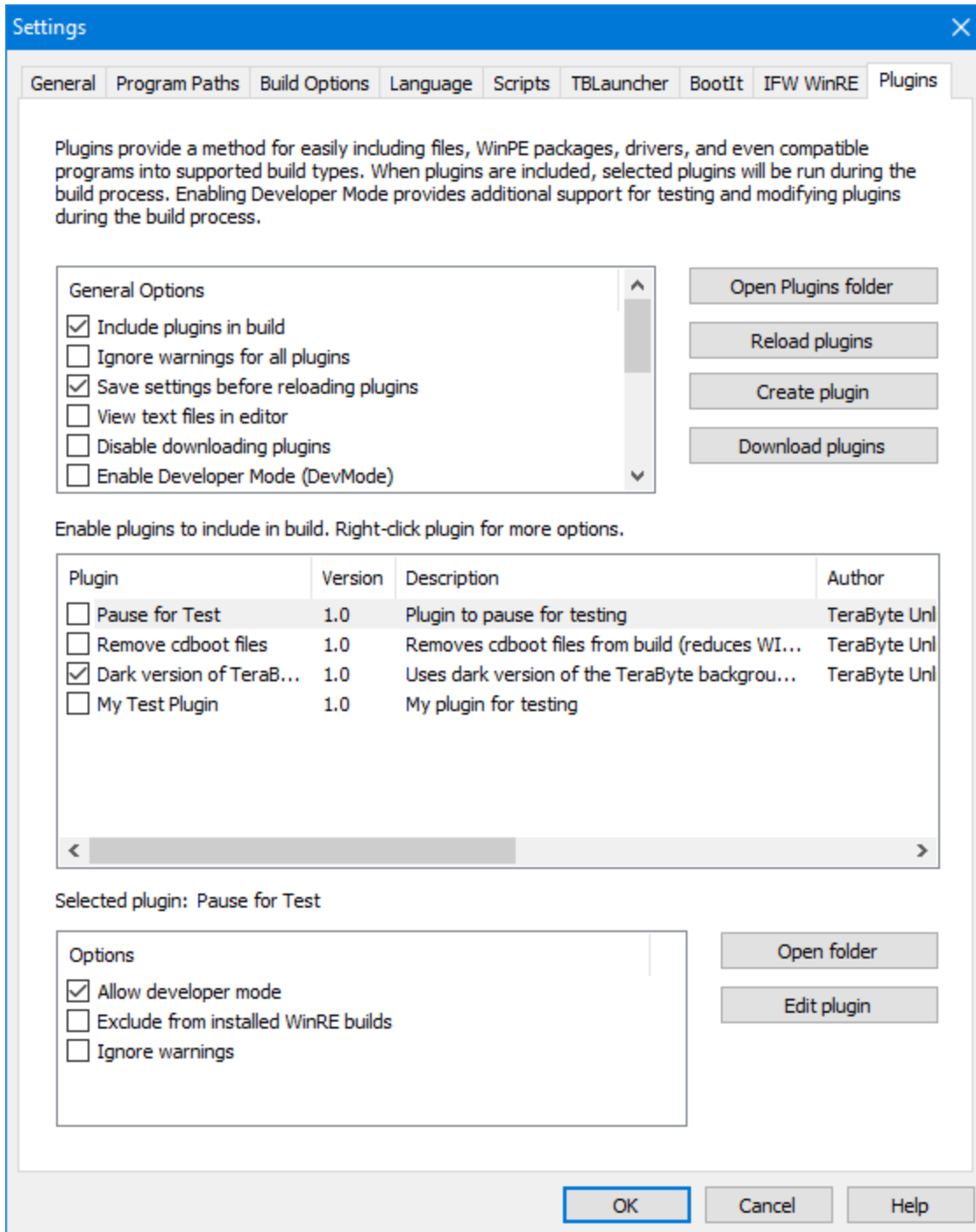
The screenshot shows a window titled "TestReq Plugin" with a close button in the top right corner. The window contains a text area with the following log output:

```
E:\TBWinRE\Plugins\_TestReq\TestReq.tbplg.log  
--TBPlugin---Initialize-----  
--TBPlugin---Process-----  
00027 LOG : TestReq running...  
00028 LOG : Build mode: TBWinRE (0)  
00032 LOG : Files copied
```

Finish by checking that the files were correctly copied into the **installpath** folder.

## Plugin Settings

Plugin settings are available on the **Plugins** tab in TBWinPE/RE Builder's Settings:



The **Open Plugins folder** button will open the build's Plugins folder. This folder contains the plugins that can be used by the build. Multiple plugins can exist in this folder, but sub-folders are recommended for plugins that require multiple files to keep things better organized and prevent issues

where multiple plugins use files with the same names. To delete a plugin, delete the plugin's files (or folder) and reload the plugins. Plugins can also be deleted using the context menu of the list.

The **Reload plugins** button will reload plugins from the Plugins folder. This is useful if changes have been made (files edited, added, removed, etc.).

The **Create plugin** button will bring up a dialog to enter basic information for the plugin (name, filename, description) and select the build modes/types/architectures supported. Other details/settings, required versions, etc., would need to be added to the plugin file by editing it. See [Creating Plugins](#)<sup>(6)</sup> for more information.

The **Download plugins** (or **Install plugins**) button will allow downloading (if enabled) and installing plugins from TeraByte's website. Plugins already installed can be updated if a new version is found. See [Downloading Plugins](#)<sup>(4)</sup> for more information.

## General Options

### Include plugins in build

Enable to include selected plugins in the build. Selected plugins will not be included in the build if this option is not enabled. This option allows you to easily toggle including/excluding plugins from builds without needing to change the selected state of the individual plugins.

### Ignore warnings

Enable to ignore warnings when checking requirements for plugins. Warnings include errors and unsupported plugins. Basically, if the plugin doesn't complete the requirements check successfully it will trigger a warning (the plugin will be disabled and not used if the build is continued).

**Note:** This option is automatically enabled for unattended builds.

### Save settings before reloading plugins

Enable to have the current plugin settings automatically saved before plugins are reloaded. Applies to both manual and automatic reloads.

**Note:** Saved settings will not revert if the settings dialog is canceled.

### View text files in editor

Enable to view text files (such as logs, ReadMe files, etc.) in the text editor instead of the built-in viewer. By default, Notepad is used as the editor.

### Disable downloading plugins

Disables downloading plugins from TeraByte's website. When downloading is disabled, only plugins already downloaded or included with the program will be available to install.

### Enable Developer Mode (DevMode)

Enables developer mode. This allows pausing the requirements and build steps to manage plugins during the build process. Additionally, test plugins can be used to aid in developing plugins. At least one plugin must be enabled for the build to enter developer mode. This option would normally only be enabled when developing or testing plugins. See [Developer Mode](#)<sup>(33)</sup> for more information.

**Note:** This option is automatically disabled for unattended builds.

### Pause on requirements check success (DevMode)

Enable to have the build pause when a requirements check is successful. By default, the build will pause only when there is an error/warning. Pausing on success can be helpful to access the test plugin as well as allowing stepping through the plugins one at a time.

### Pause on build success (DevMode)

Enable to have the build pause when the plugin's build step is successful. By default, the build will pause only when there is an error. Pausing on success can be helpful to access the test plugin as well as allowing stepping through the plugins one at a time.

### Allow TestReq plugin (DevMode)

Enable to allow access to the TestReq plugin during the plugin requirements check step. This plugin can be used for testing requirements and code to aid in developing plugins. Since the build is not mounted at the time this plugin is run, test folders and registry hives are used to allow most operations to be tested.

### Allow TestBuild plugin (DevMode)

Enable to allow access to the TestBuild plugin during the build step. This plugin can be used for testing build code and has full access to the current build like a normal plugin (this includes the mounted WIM and the WIM's registry hives).

### Reset TestReq registry hive (DevMode)

Enable to reset/clear the registry hives used by the TestReq plugin on each build run. This can be helpful for testing registry commands in a known/clear state. The reset takes place each time the plugin requirements step is started.

### Reset TestReq folders (DevMode)

Enable to reset/clear the test folders used by the TestReq plugin on each build run. This can be helpful for testing file/folder commands in a known/clear state. The reset takes place each time the plugin requirements step is started.

### Show log after running test plugin (DevMode)

Enable to automatically show the test plugin's log (applicable for both TestReq and TestBuild) after it's run. This can be helpful to quickly see the results while testing without needing to click the button to view the log every time.

**Note:** This option does not apply if the **View text files in editor** option is enabled. Refresh or reload in the editor as needed.

## Plugins

Select (check) the plugins in the list to include them in the build. Selected plugins will only be included if the **Include plugins in build** option is enabled. Plugins will be processed in the order they appear in



the list. If plugins are added that require being in certain locations (e.g. must be run before/after a different plugin), set the order as required prior to creating a build.

Additional options are accessible by right-clicking on a plugin (only applicable commands will be displayed):

|                            |  |
|----------------------------|--|
| <b>Move to top</b>         | Moves plugin to the top of the list.                                 |
| <b>Move up</b>             | Moves plugin up one in the list.                                     |
| <b>Move down</b>           | Moves plugin down one in the list.                                   |
| <b>Move to bottom</b>      | Moves plugin to the bottom of the list.                              |
| <b>Refresh</b>             | Refreshes (reloads) the plugin.                                      |
| <b>Open folder</b>         | Opens the plugin's folder.   |
| <b>Edit plugin</b>         | Opens the plugin in the editor (by default, Notepad is used).        |
| <b>View log</b>            | View the plugin's log file.  |
| <b>View description</b>    | View the plugin's long description.                                  |
| <b>View ReadMe file</b>    | View the plugin's ReadMe file.                                       |
| <b>Open plugin website</b> | Opens the website page specified in the plugin.                      |
| <b>Delete plugin</b>       | Deletes (removes) the plugin. You will be prompted for confirmation. |

In addition to the context menu, drag & drop can also be used to order the plugins.

### Individual Plugin Options / Options for Selected Plugin

These options apply to each plugin separately and may override general plugin options or internal plugin settings, allowing the user to adjust certain options without changing them for every plugin.

#### Allow developer mode

Enable to allow developer mode for the plugin. If this is disabled, developer mode will be ignored for the plugin even if the general option is enabled. This option is enabled by default. When testing or developing plugins, you may wish to disable this option for plugins that don't require pausing the build to avoid having to step through them.

#### Exclude from installed WinRE builds

Enable to exclude the plugin from builds that modify the installed WinRE (UpdateWinRE, IFWWinRE). This option will override the plugin's internal setting allowing you to prevent individual plugins from being included in those build types (e.g. if they prevent the build from completing successfully due to increased WIM size).

#### Ignore warnings

Enable to ignore warnings for the plugin. If enabled, errors and warnings will be ignored for the plugin regardless of the general option (warnings are still logged). For example, if a particular

plugin is included, but isn't supported for all build types, you can enable the option to ignore the warning while leaving the general option enabled.

The **Open folder** button will open the plugin's folder.

The **Edit plugin** button will open the plugin in the editor (by default, Notepad is used).

### Additional Plugin INI Options

Several additional options are available that are not currently accessible in the program's settings. These should be placed in the **[Plugins]** section of the TBWinPE.ini file.

#### Editor

Use this option to specify the path to the text editor you wish to use for editing the plugin files. By default, Notepad is used.

Example:

```
Editor=c:\program files\myeditor\myeditor.exe
```

#### EditorParameters

Use this option if the editor requires any additional command line parameters. By default, only the file being edited is passed as a command line parameter. In the specified text, **%s** will be replaced with the file being edited.

Example:

```
EditorParameters=/f "%s"
```

## Obtaining Technical Support

The primary support communication method will be use of online services. The most recent versions of software and information will be available on the TeraByte Unlimited website:

[www.terabyteunlimited.com](http://www.terabyteunlimited.com)

Registered users that require technical support should try to use the following email address as the primary communication method:

[support@terabyteunlimited.com](mailto:support@terabyteunlimited.com)

Pre-sale information and technical support for unregistered users is available by email only.

In all cases, registered or not, TeraByte Unlimited reserves the right to refuse any communication method that would incur a cost.

## Trademarks

Microsoft and Windows are registered trademarks of Microsoft Corporation.

All other trademarks and copyrights referred to are the property of their respective owners.