



## Paquet Builder Guide

Build professional Windows installers  
for files and applications

# Paquet Builder

## User Manual

---

Version 2026

G.D.G. Software

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

---

## Table of Contents

- 1. Build Better Windows Installer Packages**
  - 1.1 Start With a Practical Workflow
  - 1.2 Browse by Area
  - 1.3 What You Can Build
  - 1.4 Recommended Next Steps
- 2. Introduction to Paquet Builder**
  - 2.1 What Makes It Different
  - 2.2 Typical Use Cases
  - 2.3 Core Capabilities
  - 2.4 Designed for Beginners and Power Users
  - 2.5 Continue Reading
- 3. Getting Started**
  - 3.1 Navigating the Interface
  - 3.2 Getting Started
- 4. Contact Information**
  - 4.1 What to Include in a Support Request
- 5. Create Packages**
- 6. Quick Tutorial: Create a Simple Package**
  - 6.1 Continue From Here
- 7. Create a Project**
  - 7.1 Starting a New Project
  - 7.2 Choose the Right Package Type
  - 7.3 At a Glance
  - 7.4 Continue From Here
- 8. General Settings**
- 9. About Your Package**
  - 9.1 Package Title
  - 9.2 Company Information
  - 9.3 Related Pages
- 10. File Manager**
  - 10.1 Adding Files
  - 10.2 Managing Files
  - 10.3 Component Manager
  - 10.4 File Actions
  - 10.5 Related Pages
- 11. Select Installation Scope**
  - 11.1 Scope Modes
  - 11.2 Impact on Installer Behavior
  - 11.3 Scope Options
  - 11.4 User Choice — Under the Hood
  - 11.5 Runtime Scope Variables
  - 11.6 Directive Support
  - 11.7 Best Practices
  - 11.8 Related Pages
- 12. Set Main Destination Folder**
  - 12.1 Default Destination Folder
  - 12.2 Folder Prompt Message
  - 12.3 Folder Dialog Options
  - 12.4 Additional Option
  - 12.5 Related Pages
- 13. Select Interface Theme**
  - 13.1 Interface Themes
  - 13.2 Theme Comparison
  - 13.3 Related Pages
- 14. Modify Welcome Screen**
  - 14.1 Welcome Screen Previews
  - 14.2 Editing the Dialog Text
  - 14.3 Variables and Resource Strings
  - 14.4 Templates
  - 14.5 Multilanguage Packages
  - 14.6 Related Pages
- 15. Display Prompt Messages**
  - 15.1 Prompt Messages
  - 15.2 Formatting and Variables
  - 15.3 Related Pages
- 16. Readme and License Agreement**
  - 16.1 How Each Screen Works
  - 16.2 Editing the Text
  - 16.3 Multilanguage Support
  - 16.4 Related Pages

- 17. Select Components**
  - 17.1 Runtime Preview
  - 17.2 How It Works
  - 17.3 Adjusting Component Order
  - 17.4 Customizing the Prompt Text
  - 17.5 Related Pages
- 18. Edit Final Screen**
  - 18.1 Runtime Previews
  - 18.2 Editing the Screen Text
  - 18.3 Checkboxes and Additional Tasks
  - 18.4 Multilanguage Support
  - 18.5 Related Pages
- 19. Program, Setup Execution**
  - 19.1 Post-Extraction Action
  - 19.2 Additional Options
  - 19.3 Related Pages
- 20. Functionality**
- 21. File Extraction Settings**
  - 21.1 Overwrite Modes
  - 21.2 Advanced Options
  - 21.3 Decompression Engines
  - 21.4 Related Pages
- 22. Edit Progress Monitor**
  - 22.1 Progress Dialog Style
  - 22.2 Status Text
  - 22.3 Options
  - 22.4 Related Pages
- 23. Shortcut Management**
  - 23.1 Using the Shortcut Manager
  - 23.2 Start Menu Folder
  - 23.3 Shortcut Properties
  - 23.4 Useful Variables
  - 23.5 Internet Shortcuts
  - 23.6 Related Pages
- 24. Write Registry Entries**
  - 24.1 Managing Entries
  - 24.2 Import / Export .reg Files
  - 24.3 Entry Editor
  - 24.4 Accessing an Alternate Registry View
  - 24.5 Important Considerations
  - 24.6 Related Topics
- 25. Language and Localization**
  - 25.1 Managing Languages
  - 25.2 Language Detection
  - 25.3 Setting the Language Programmatically
  - 25.4 Role of the Default Language
  - 25.5 Related Pages
- 26. Protect Package Files**
  - 26.1 File Encryption and Passwords
  - 26.2 Additional Protection
  - 26.3 Related Pages
- 27. Configure Uninstaller**
  - 27.1 Core Settings
  - 27.2 Additional Options
  - 27.3 Silent Uninstaller
  - 27.4 Related Pages
- 28. Build Settings**
- 29. Set output EXE file path**
  - 29.1 Building your package
  - 29.2 About Archive Caching
  - 29.3 Build Options
  - 29.4 Compilation Log
  - 29.5 Cancel
  - 29.6 Related Pages
- 30. Modify File Compression**
  - 30.1 Storing Path Information
  - 30.2 Compression Level
  - 30.3 Compression Options
  - 30.4 Related Pages

- 31. Digitally Sign Packages — Code Signing**
  - 31.1 Signed vs. Unsigned
  - 31.2 Choose Your Signing Method
  - 31.3 Obtaining a Code Signing Certificate
  - 31.4 Configuring the Certificate
  - 31.5 Digest Algorithms
  - 31.6 Additional Options
  - 31.7 Related Topics
- 32. Multi-volume Archive**
  - 32.1 Why Use Multi-volume Archives?
  - 32.2 Volume Size
  - 32.3 Output Filenames
  - 32.4 Extracting a Multi-volume Archive
  - 32.5 GSplitted Advanced File Archiver
- 33. Edit Package Resources**
  - 33.1 Icons
  - 33.2 Version Information
  - 33.3 Related Pages
- 34. Advanced Build Options**
  - 34.1 Compress EXE Stub
  - 34.2 Build Folder Options
  - 34.3 Loader Temporary Resources
  - 34.4 Related Pages
- 35. Developer Options**
  - 35.1 Content Viewer (Dialog Rendering)
  - 35.2 Additional C Scripts
  - 35.3 Tutorial: Call a C Function
  - 35.4 Related Pages
- 36. After Build**
  - 36.1 Post-Build Options
  - 36.2 Splitting Large Packages
  - 36.3 Related Pages
- 37. How to Sign Installers with SignTool and Paquet Builder**
  - 37.1 1. Prerequisites
  - 37.2 2. Enabling Digital Signing in Paquet Builder
  - 37.3 3. Understanding SignTool Parameters in Paquet Builder
  - 37.4 4. Signing Your Installer Package
  - 37.5 5. Alternative: Signing with a Specific Certificate
  - 37.6 Multiple Digital Signatures in Paquet Builder
  - 37.7 Defining the Path to SignTool in Paquet Builder
  - 37.8 Related Topics
- 38. How to Sign Installers with Azure Artifact Signing in Paquet Builder**
  - 38.1 1. Introduction
  - 38.2 2. Why Choose Azure Artifact Signing?
  - 38.3 3. Prerequisites
  - 38.4 4. Required Installations
  - 38.5 5. Configuring Azure Artifact Signing in Paquet Builder
  - 38.6 Configuring SignTool and Azure Artifact Signing Paths in Paquet Builder
  - 38.7 Related Topics
- 39. How to Sign Installers with JSign in Paquet Builder**
  - 39.1 Prerequisites
  - 39.2 Configuring JSign in Paquet Builder
  - 39.3 Command Placeholders
  - 39.4 Examples
  - 39.5 Directive Support
  - 39.6 Comparison with Other Signing Methods
  - 39.7 Related Topics
- 40. Go Further**
  - 40.1 Core Logic & Variables
  - 40.2 Folder & Component Management
  - 40.3 System Integration & UI
- 41. Working with Variables**
  - 41.1 How Variables Work
  - 41.2 Defining Custom Variables
  - 41.3 Built-in Variable Categories
  - 41.4 Best Practices
  - 41.5 Related Pages

- 42. Global Variable Reference**
  - 42.1 Profile Resolution
  - 42.2 Package and Archive Information
  - 42.3 Core Session Paths
  - 42.4 User Profile Folders (Current User)
  - 42.5 Common Folders (All Users)
  - 42.6 System Folders
  - 42.7 Operating System and Hardware
  - 42.8 Installation Scope
  - 42.9 Runtime State and Logic
  - 42.10 Command-line Parameters
  - 42.11 Localization
  - 42.12 OS Version Reference
  - 42.13 Special Formatting
  - 42.14 Windows Known Folders
  - 42.15 Regarding %ISADMIN% and %ISPOWERUSER%
  - 42.16 See Also
- 43. Organizing Files into Components**
  - 43.1 Component Management
  - 43.2 Component Properties
  - 43.3 Advanced Usage
  - 43.4 Related Pages
- 44. Automating Component Creation**
  - 44.1 Running the Wizard
  - 44.2 What the Wizard Does
  - 44.3 Related Pages
- 45. Individual File Properties**
  - 45.1 Accessing File Properties
  - 45.2 Customizing File Destination
  - 45.3 Conditional File Extraction
  - 45.4 Uninstaller Options
  - 45.5 Reset Properties
  - 45.6 Related Pages
- 46. Resource Strings and Localization**
  - 46.1 Using Resource Strings
  - 46.2 The Resource String Editor
  - 46.3 Creating Multilingual Packages
  - 46.4 Technical Details
  - 46.5 Related Pages
- 47. Designing Rich Content**
  - 47.1 The Editor
  - 47.2 Key Features
  - 47.3 Adding Images
  - 47.4 Related Pages
- 48. Managing the Package Uninstaller**
  - 48.1 What the Uninstaller Removes
  - 48.2 Enabling the Uninstaller
  - 48.3 Advanced Options
  - 48.4 UAC and Elevation
  - 48.5 Related Pages
- 49. User Account Control (UAC) and Elevation**
  - 49.1 Requested Execution Levels
  - 49.2 Best Practices
  - 49.3 Related Pages
- 50. AI Assistant Overview**
  - 50.1 Key Capabilities
  - 50.2 How It Works
  - 50.3 Proposed Action Cards
  - 50.4 Context Awareness
  - 50.5 Dark and Light Theme
  - 50.6 Getting Started
  - 50.7 Related Topics
- 51. Setting Up the AI Assistant**
  - 51.1 Quick Setup
  - 51.2 Available Providers
  - 51.3 API Key Security
  - 51.4 Choosing a Model
  - 51.5 Advanced Settings
  - 51.6 Mock Mode
  - 51.7 Independent Storage
  - 51.8 Related Topics
- 52. AI Providers and Models**
  - 52.1 Cloud Providers
  - 52.2 Local AI Providers
  - 52.3 Choosing the Right Model
  - 52.4 Cost Considerations
  - 52.5 Provider-Specific Notes
  - 52.6 Related Topics

**53. What the AI Assistant Can Do**

- 53.1 Project Settings
- 53.2 File and Component Management
- 53.3 Variable Management
- 53.4 Localization and Resource Strings
- 53.5 Registry Entries
- 53.6 Shortcuts
- 53.7 Screen Content
- 53.8 Custom Actions
- 53.9 Documentation Search
- 53.10 UI Navigation
- 53.11 Application Actions
- 53.12 Screenshot Capture
- 53.13 Complete Tool Reference
- 53.14 Related Topics

**54. Using the AI Assistant — Examples and Tips**

- 54.1 Opening the AI Assistant
- 54.2 The Welcome Screen
- 54.3 Prompt Examples
- 54.4 Reviewing and Applying Changes
- 54.5 Context-Aware Suggestions
- 54.6 Tips for Best Results
- 54.7 Dark and Light Theme
- 54.8 Clearing the Conversation
- 54.9 Related Topics

**55. Using Local AI Models**

- 55.1 Supported Local Providers
- 55.2 Setting Up a Local Provider
- 55.3 Recommended Local Models
- 55.4 Limitations of Local Models
- 55.5 Custom Host and Port
- 55.6 Hybrid Approach
- 55.7 Related Topics

**56. MCP Server Overview**

- 56.1 What is MCP?
- 56.2 Use Cases
- 56.3 AI Assistant vs. MCP Server
- 56.4 Available Tools (20+)
- 56.5 Getting Started
- 56.6 Related Topics

**57. Setting Up the MCP Server**

- 57.1 How to Obtain the MCP Server
- 57.2 Installation
- 57.3 Configuration
- 57.4 Verifying the Connection
- 57.5 Typical Workflow
- 57.6 Example: Creating an Installer with Claude Code
- 57.7 Protocol Details
- 57.8 Troubleshooting
- 57.9 Related Topics

**58. MCP Server Tool Reference**

- 58.1 Tools at a Glance
- 58.2 Project Management
- 58.3 Settings
- 58.4 Collections
- 58.5 Custom Actions
- 58.6 Build
- 58.7 Documentation and Filesystem
- 58.8 Example Session
- 58.9 Related Topics

**59. Custom Action System****60. Custom Actions Overview**

- 60.1 Why Use Custom Actions
- 60.2 When Actions Run
- 60.3 Key Rules to Remember
- 60.4 Undo and Redo
- 60.5 Modern Rendering for Dialog Actions
- 60.6 Explore Further

**61. Custom Action Catalog**

- 61.1 User Interface
- 61.2 Variables and Logic
- 61.3 File System
- 61.4 Execution
- 61.5 Registry and Configuration
- 61.6 XML Editing
- 61.7 Shortcuts and System

- 62. Managing Custom Actions**
  - 62.1 UI Features
  - 62.2 Event Indicators
  - 62.3 Working with Actions
  - 62.4 Import and Export
  - 62.5 Undo and Redo
  - 62.6 Keyboard Shortcuts
  - 62.7 Related Pages
- 63. Editing Action Properties**
  - 63.1 Editor Layout
  - 63.2 Editing Values
  - 63.3 Inserting Dynamic Content
  - 63.4 Related Pages
- 64. Custom Action Templates**
  - 64.1 Importing a Template
  - 64.2 Built-in Templates
  - 64.3 Creating Custom Templates
  - 64.4 Template Directories
  - 64.5 Sharing Templates
  - 64.6 Related Pages
- 65. Ask User (Text Input)**
  - 65.1 Properties
  - 65.2 Behavior
- 66. Ask User for One or More Choices**
  - 66.1 Properties
  - 66.2 Behavior
  - 66.3 Additional Notes
- 67. Ask User for a Choice (Multiple Options in Drop Down)**
  - 67.1 Properties
  - 67.2 Syntax Rules for Comboltems
- 68. Ask User for a Folder**
  - 68.1 Properties
  - 68.2 Behavior
- 69. Ask User for a Choice**
  - 69.1 Properties
  - 69.2 Behavior
- 70. Call C Function**
- 71. Check if a File/Folder Exists**
  - 71.1 Properties
- 72. Check if a Process Is Running**
  - 72.1 Properties
- 73. Check if a Registry Key Exists**
  - 73.1 Properties
- 74. Conditional If/Then/Else**
  - 74.1 How to Use
  - 74.2 Properties of "Begin Conditional If/Then"
  - 74.3 Properties of "End Conditional If/Then"
  - 74.4 Properties of "Else Conditional If/Then"
  - 74.5 Error Detection for Missing "End If/Then"
- 75. Copy a File/Folder**
  - 75.1 Properties
- 76. Create a Folder**
  - 76.1 Properties
- 77. Create an Internet Shortcut**
  - 77.1 Properties
- 78. Create a Shortcut**
  - 78.1 Properties
- 79. Define a Label**
  - 79.1 Properties
  - 79.2 See Also
- 80. Delete a Folder**
  - 80.1 Properties
- 81. Do an Arithmetic Operation**
  - 81.1 Properties
  - 81.2 Left/Right Variables
  - 81.3 Available operations
- 82. Execute a BAT File**
  - 82.1 Properties
- 83. Execute a Dialog Scenario**
  - 83.1 Properties
- 84. Execute a Program**
  - 84.1 Properties
- 85. Exit Package**
- 86. Find One String Within Another**
  - 86.1 Properties
- 87. Get File Information**
  - 87.1 Properties
  - 87.2 Available information
- 88. Get Full Path of a Windows Known Folder**
  - 88.1 Properties
  - 88.2 KNOWNFOLDERID constants

- 89. Go to a Label**
  - 89.1 Properties
  - 89.2 Example of using Go to label
  - 89.3 Restrictions
- 90. Insert an Uninstall Command**
  - 90.1 Properties
  - 90.2 Available uninstall commands
  - 90.3 Example for ucRemoveFromIni
  - 90.4 Upgrading from Paquet Builder 2.9
  - 90.5 Tips
- 91. Install MSI Package or Update**
  - 91.1 Properties
  - 91.2 Other MSI actions
- 92. Perform Operation on a Variable**
  - 92.1 Properties
  - 92.2 Available operations
  - 92.3 Examples for varStringReplace
- 93. Open a Document File**
- 94. Read Data (or Variables) from a File**
- 95. Read an Entry from the Registry**
  - 95.1 Reading destination path from registry
- 96. Read a Key from an .ini File**
- 97. Refresh Windows Shell Icons**
- 98. Register a Shared File**
- 99. Remove a Registry Key or Entry**
- 100. Remove Single File or Using Wildcard**
- 101. Rename a File**
- 102. Replace Variables in File**
  - 102.1 Notes
  - 102.2 Example
- 103. Restart Computer**
- 104. Self-Register a File (ActiveX, OLE)**
- 105. Set File Attributes**
- 106. Show an Information Panel**
- 107. Show a License Agreement**
- 108. Show a Message Box**
- 109. Show a Readme Dialog**
- 110. Show/Hide a Wait Message Dialog**
- 111. Sleep (Pause Program Execution)**
- 112. User Comment**
- 113. Write an Environment Variable**
  - 113.1 Example 1: Set a new application directory variable
  - 113.2 Example 2: Add to system PATH
- 114. Write a Registry Entry**
  - 114.1 Example
- 115. Write Text (or Variables) to a File**
- 116. Write a Value to an .ini File**
- 117. XML - Begin Editing File or String**
  - 117.1 Examples
  - 117.2 Best practices
- 118. XML - End Editing File (and Save)**
- 119. XML - Read Attribute Value from Node**
  - 119.1 Behavior when the node does not exist
  - 119.2 PathToNode explanation
- 120. XML - Read Value from Node**
  - 120.1 Behavior when the node does not exist
  - 120.2 PathToNode explanation
- 121. XML - Write Node Attribute**
  - 121.1 PathToNode explanation
- 122. XML - Write Node**
  - 122.1 Behavior when the node does not exist
  - 122.2 PathToNode explanation
- 123. Automation and Command Line**
  - 123.1 Automation via MCP Server
- 124. Command-line options and switches for Paquet Builder**
  - 124.1 What is a switch?
  - 124.2 Launching Paquet Builder from the command line
  - 124.3 Supported switches
  - 124.4 Examples
  - 124.5 See also

- 125. About Directive Files**
  - 125.1 Introduction to Directive Files
  - 125.2 How Directive Files Work
  - 125.3 [General] Section Parameters
  - 125.4 [Source] Section
  - 125.5 [Components] Section
  - 125.6 Directive Variables
  - 125.7 [Templates] Section — Screen Content
  - 125.8 [SignToolCmd] Section — Custom SignTool Commands
  - 125.9 [JSignCmd] Section — JSign Commands
  - 125.10 Define Custom Directive Variables
  - 125.11 Example: Simple Directive File Without Source
  - 125.12 Related Topics
- 126. How to execute directive files in Paquet Builder?**
- 127. Console Command-line Compiler**
  - 127.1 Benefits of the Console Compiler
  - 127.2 How to Obtain the Console Command-line Compiler
  - 127.3 Setting Up the Console Command-line Compiler
  - 127.4 Using the Console Command-line Compiler
  - 127.5 Additional Notes
- 128. Exit codes for Paquet Builder**
- 129. Command-line switches for packages**
  - 129.1 Accessing the command line for packages and uninstallers
  - 129.2 Defining command-line switches
- 130. Exit code for packages and installers**
  - 130.1 Exit code meaning
  - 130.2 Default exit code values for Paquet Builder packages
- 131. About 7-Zip Format**
  - 131.1 Key Features
  - 131.2 Compression Methods
  - 131.3 Filters
  - 131.4 LZMA Characteristics
  - 131.5 Technical Notes
  - 131.6 Related Pages
- 132. Working with Microsoft Windows Installer Packages**
  - 132.1 Creating an MSI Setup Package
  - 132.2 Advanced MSI Actions
  - 132.3 Compressing MSI Files with 7-Zip
  - 132.4 Additional Resources
- 133. About Some Setup Software**
  - 133.1 The Problem
  - 133.2 Solutions
  - 133.3 Related Pages
- 134. Environment Options**
  - 134.1 Settings
  - 134.2 AI Assistant
  - 134.3 Related Pages
- 135. Technical Documentation**
  - 135.1 System Requirements
  - 135.2 Package Size Limit
  - 135.3 AI and Automation
- 136. Automatic Updates**
  - 136.1 Automatic Update Checks
  - 136.2 Manual Update Check
  - 136.3 Update Dialog
  - 136.4 Additional Packages
  - 136.5 Security
  - 136.6 Settings
  - 136.7 Related Pages

# 1. Build Better Windows Installer Packages

---

Paquet Builder combines self-extracting archives, installer logic, custom actions, code signing, and automation in one visual environment.

Use this documentation both as a guided manual and as a context-sensitive reference. When you press F1 inside Paquet Builder, the application opens the matching topic directly.

## Read it your way

If you are new to Paquet Builder, follow the guided topics below in order. If you already know the interface, jump straight to the setting or custom action you need.

## 1.1 Start With a Practical Workflow

### Create a project

Choose a package type, define the package identity, and save a reusable project file. [Open project creation](#) →

### Add files and screens

Set the destination folder, organize your files, and design the user-facing package screens. [Open the quick tutorial](#) →

### Add advanced behavior

Use custom actions, variables, registry operations, and runtime conditions without writing scripts. [Browse custom actions](#) →

### Automate the output

Compile from the command line, run directive files, or control projects through AI workflows. [Open automation topics](#) →

## 1.2 Browse by Area

## Create Packages →

Project types, first builds, and the shortest route from files to a working EXE.

## General Settings →

Package identity, destination folders, screens, and file management.

## Functionality →

Runtime behavior such as shortcuts, registry entries, languages, and extraction rules.

## Build Settings →

Output paths, compression, signatures, resources, and advanced build-time options.

## Go Further →

Variables, components, uninstallers, rich content, and deeper package customization.

## Custom Actions →

Visual runtime logic for dialogs, file operations, conditions, XML, INI, and registry work.

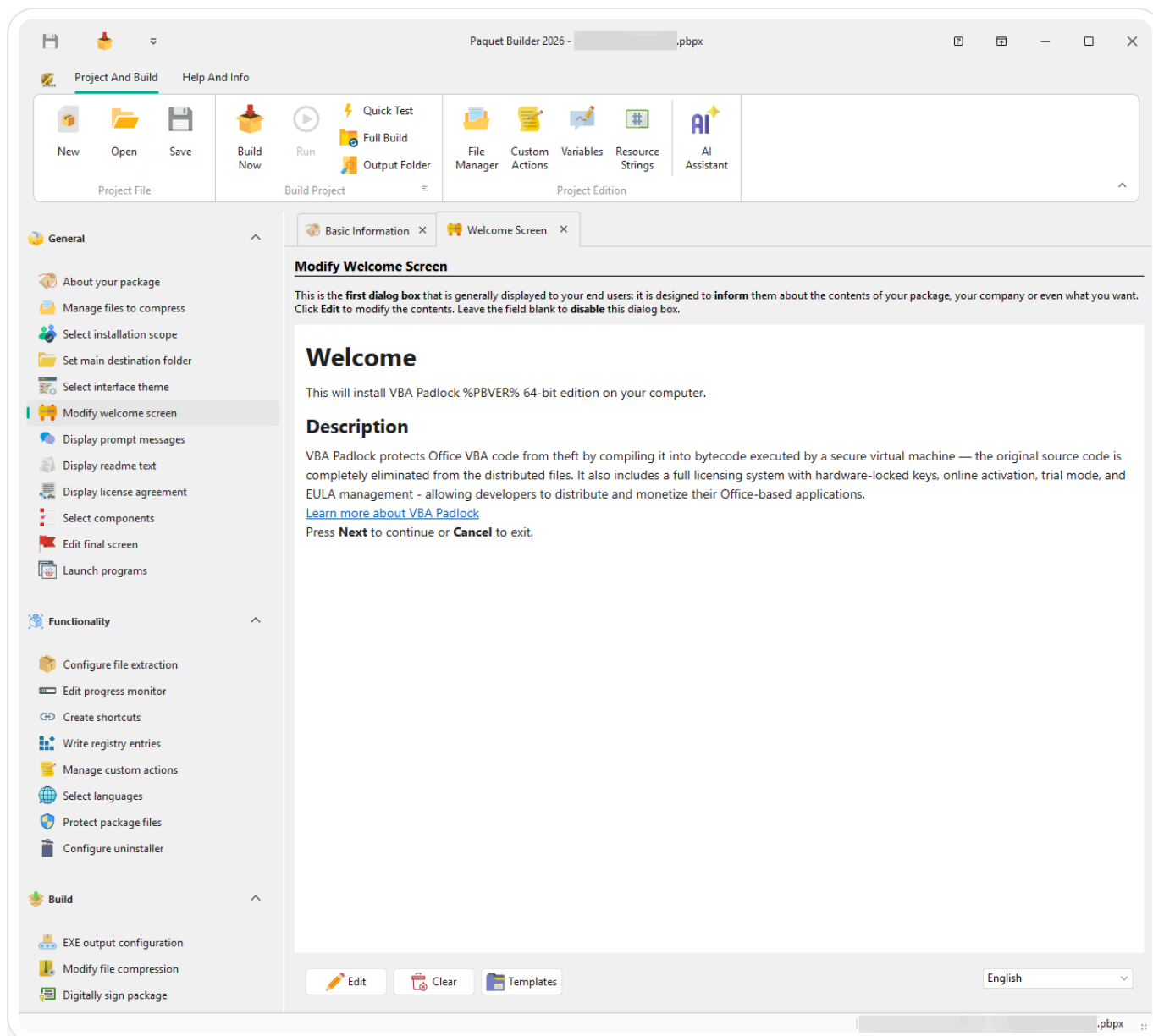
## AI Assistant →

Configure projects through natural language inside Paquet Builder.

## MCP Server →

Manage projects headlessly from Claude Code, Cursor, Windsurf, and other MCP clients.

## 1.3 What You Can Build



## Self-extracting archives

Package documents, media, binaries, or mixed content into compact, distributable `.exe` files.

## Software installers

Build setup flows with destination folders, welcome screens, readme pages, shortcuts, uninstallers, and post-install actions.

## Updates and maintenance

Ship patches, layered setups, MSI wrappers, and controlled cleanup actions for existing installations.



## Automated build workflows

Reuse projects in CI/CD pipelines, command-line builds, directive files, and AI-assisted configuration flows.

### 1.4 Recommended Next Steps

Start with the [Introduction](#) if you want the big picture. If you prefer learning by doing, jump straight to [Getting Started](#) and then follow [Create a Simple Package](#).

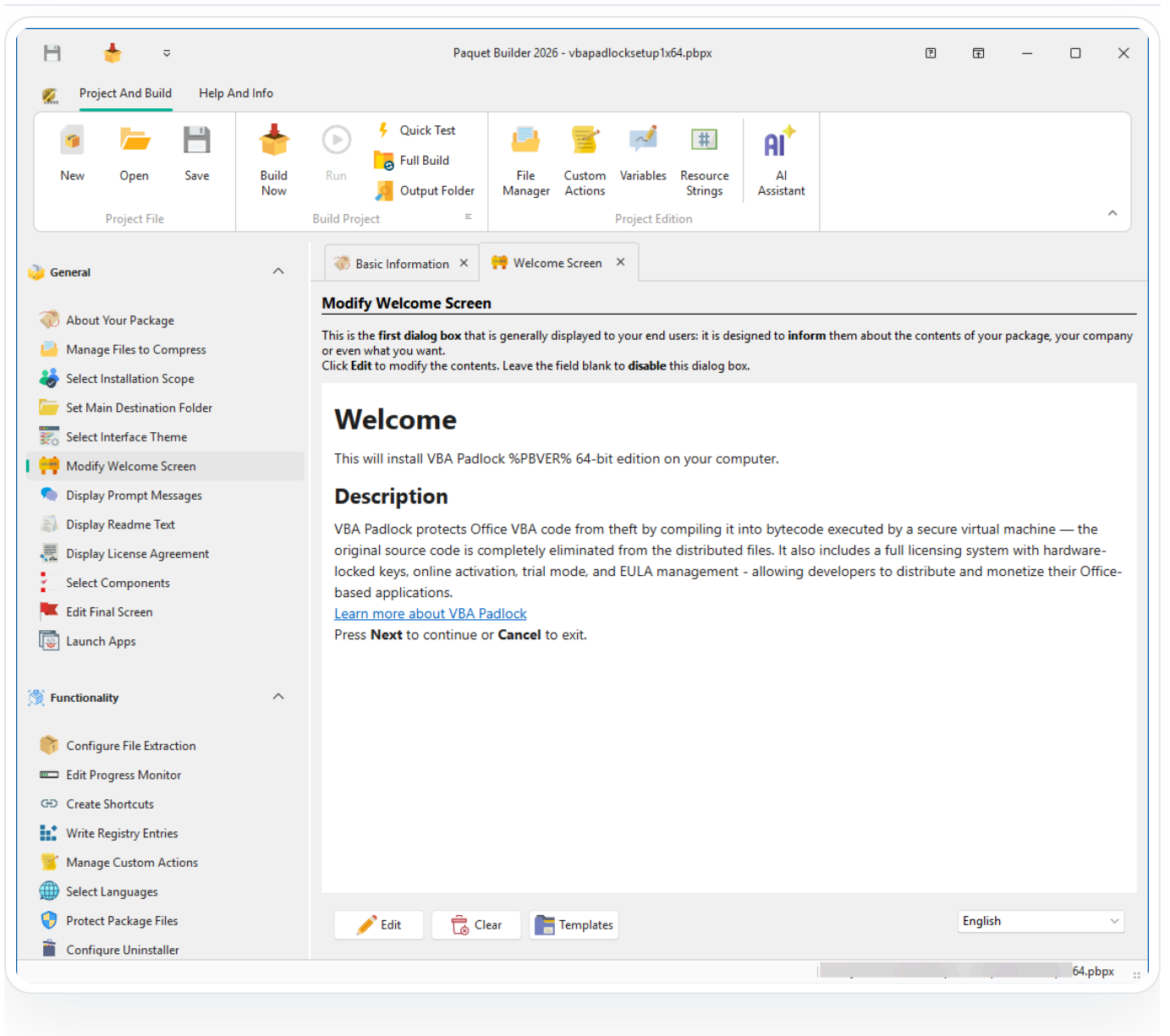
## 2. Introduction to Paquet Builder

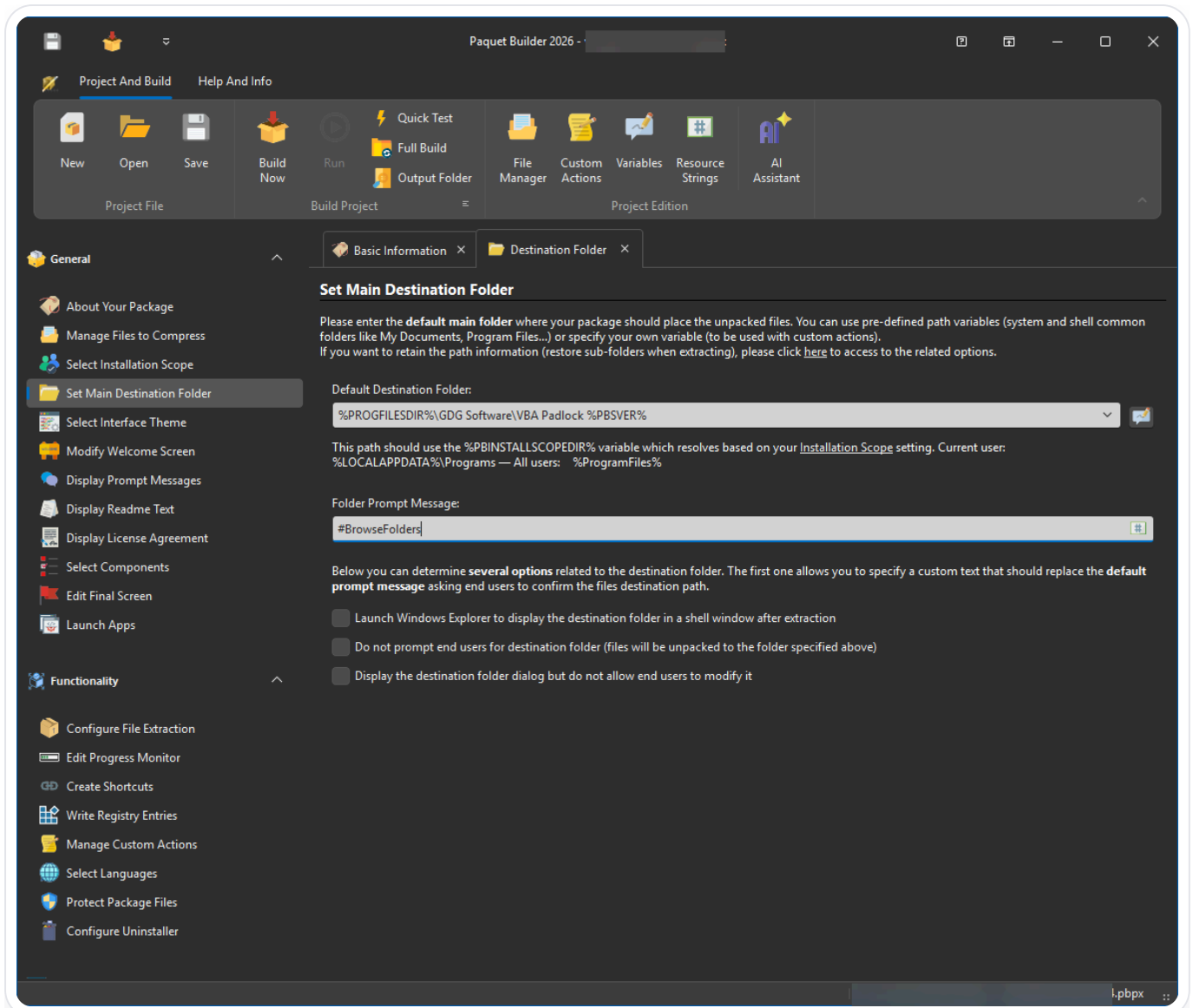
---

Paquet Builder sits between a self-extracting 7-Zip builder and a full Windows package authoring environment. It can be as simple as “add files and build an EXE” or as detailed as “design a multilingual installer with signing, custom dialogs, registry work, and scripted runtime behavior.”

### More than an archive wrapper

Paquet Builder is not limited to unpacking files. You can build setup routines, combine multiple installers, deliver updates, run post-install tasks, and automate project generation or compilation.





## 2.1 What Makes It Different



### Visual workflow

Create packages without scripting. Most tasks are configured through dedicated editors, dialogs, and checkboxes.



### Installer-grade behavior

Add welcome screens, readme pages, license agreements, shortcuts, uninstallers, and post-install execution.



## Advanced runtime logic

Use variables, conditions, custom actions, registry operations, XML editing, file checks, and user prompts.



## Modern delivery options

Sign packages, optimize compression, target x86, x64, or ARM64, and automate builds through CLI, directives, AI, or MCP.

## 2.2 Typical Use Cases

Paquet Builder can be used to:

- Package documents, media, or application binaries into a single `.exe`.
- Build software installers with wizard screens and custom behavior.
- Distribute updates, patches, or layered setup packages.
- Wrap MSI or MSP packages into a controlled delivery flow.
- Create multilingual packages with Unicode and right-to-left language support.
- Automate builds for internal tools, commercial software, or CI/CD pipelines.

## 2.3 Core Capabilities



### Files and components

Organize files into components, define destination paths, and let end users select optional content.



### Compression and output

Build compact archives with 7-Zip compression methods such as LZMA2, BCJ2, and PPMD, including multi-volume output.



### Security and trust

Protect output with passwords, integrity checks, Authenticode signatures, or Azure Artifact Signing.



## AI and automation

Configure projects with the built-in **AI Assistant** or manage them headlessly through the **MCP Server**.

## 2.4 Designed for Beginners and Power Users

You can create a working package in minutes: start a new project, add files, choose an output path, and click **Build Now**. After that, you can gradually enable advanced options such as custom actions, digital signatures, environment defaults, or automated build flows.

### Compatibility

Paquet Builder 2026 and the packages it creates are compatible with current Windows desktop and server environments, including x86, x64, and ARM64 targets where applicable.

## 2.5 Continue Reading

### Getting Started →

Learn how the interface is organized and where to begin.

### Create a Project →

Choose the right package type and create your first reusable project.

### Quick Tutorial →

Build a simple package step by step.

### Online Tutorials →

Watch additional guided walkthroughs on the website.

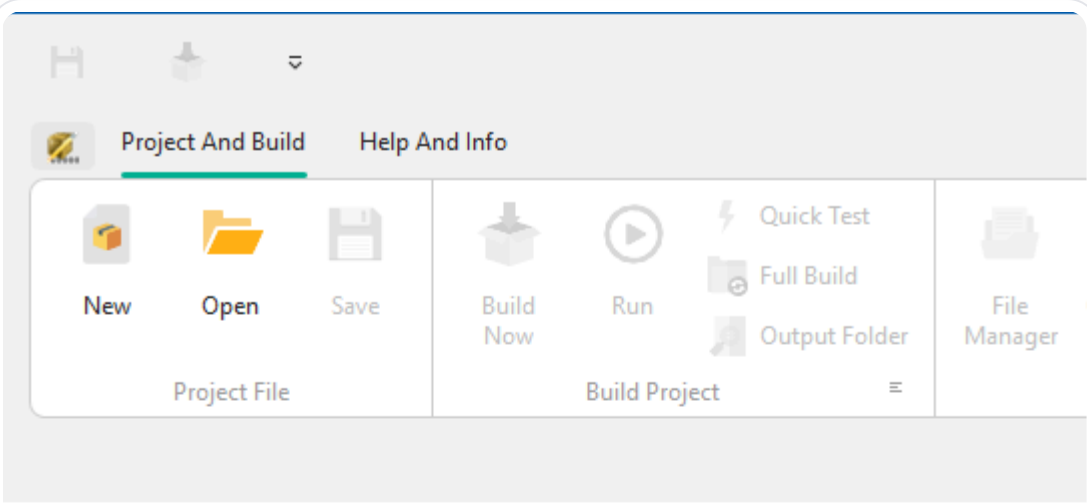
## 3. Getting Started

---

When Paquet Builder opens, you land on the welcome page where you can create a new project or reopen an existing one. From there, the product guides you through the full package workflow: define the package, add content, choose runtime behavior, and compile the final output.

### Context-sensitive help

Press F1 anywhere in Paquet Builder to open the topic that matches the current page or editor.



**Welcome to Paquet Builder**

Build professional **self-extracting packages and installers** in minutes. Package files, documents, or full applications with a workflow that stays simple at first and powerful when you need more control.

Customize every stage of the experience with **screens, components, custom actions, signing, and advanced build options**. Press **F1** at any time for context-sensitive help, or use the **AI Assistant** to build and configure your project in natural language.

Get started:

- [New Package](#)
- [Open Project](#)
- [Recent Projects](#)
- [Documentation](#)

## Note

Throughout this help, the words “package” or “installer” refer to the output created by Paquet Builder, most often a self-extracting .exe file.

## 3.1 Navigating the Interface

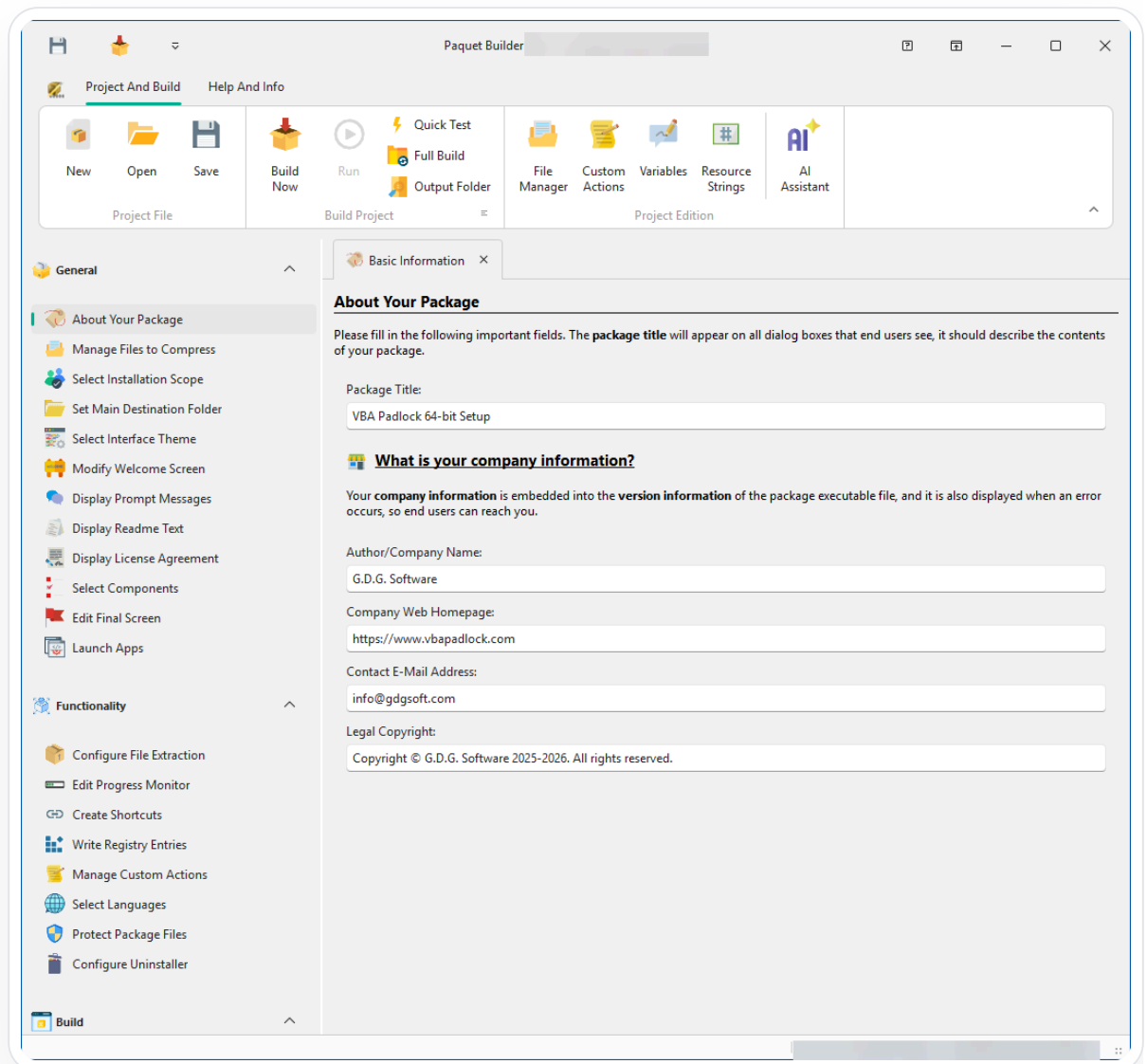
### 1 Start with a new or existing project

Create a new package from scratch or reopen a saved project file.

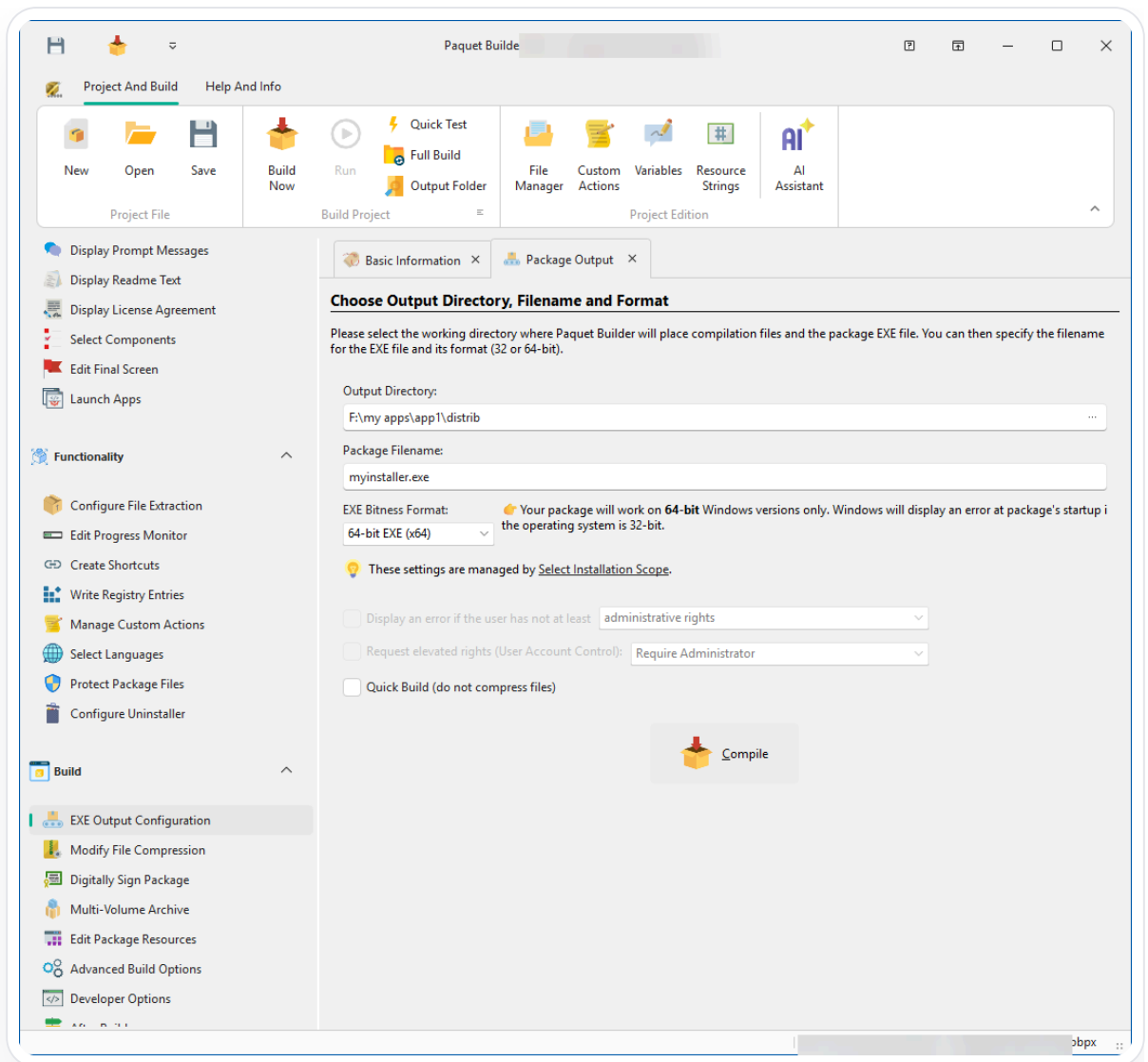
## 2 Use the left navigation to move through package settings

Paquet Builder groups most options into three main panels:

- **General** for the package identity, screens, file management, and basic user-facing behavior.

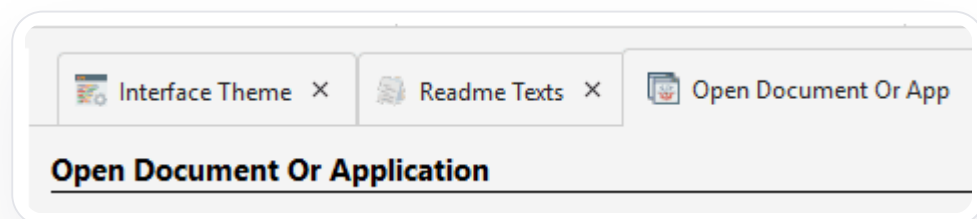


- **Functionality** for runtime behavior such as shortcuts, registry entries, or extraction logic.
- **Build** for output paths, signatures, compression, and advanced compilation options.



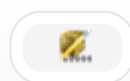
### 3 Open multiple pages in tabs

Clicking items in the navigation tree opens them in tabs, so you can compare settings or keep several topics open while working.



### 4 Use the application button for global commands

This menu gives you access to broader actions such as project-level commands and utilities.



## Project type matters

Some options and links vary depending on the **project type** you selected when the project was created.

## 3.2 Getting Started

### Follow the quick tutorial

The fastest path to understanding the workflow is to build a simple package from start to finish. [Open the tutorial →](#)

### Read about project types

Paquet Builder supports different kinds of packages, and the right starting point depends on what you want to ship. [Choose a project type →](#)

### Ask the AI Assistant

In Paquet Builder 2026, you can describe the package you want in natural language and let the assistant prepare settings for you. [Open AI Assistant overview →](#)

### Need more examples?

If you prefer guided walkthroughs and practical scenarios, continue with the online tutorials. [Browse tutorials →](#)

## 4. Contact Information

Use the links below for downloads, product questions, bug reports, and general support.

### Paquet Builder website →

Downloads, product pages, tutorials, and release information.

### Contact support →

Send questions, bug reports, or product feedback through the contact page.

### Community forum →

Ask questions, share ideas, and browse discussions around G.D.G. Software products.

### G.D.G. Software →

Company information and the broader product catalog.

### 4.1 What to Include in a Support Request

#### 1 Your full name and a valid email address

This makes it possible to reply and keep track of the conversation.

#### 2 The Paquet Builder version

Always include the product version you are using, especially when reporting unexpected behavior.

#### 3 Your environment details

For bug reports, add the Windows version, architecture, and any useful configuration details.

#### 4 A short description of the issue

Explain what you expected, what actually happened, and whether the issue is reproducible.

#### 5 Your user ID if you are a registered customer

Registered users should include their user ID to receive priority technical support.

## Before reporting a bug

If possible, mention the project type, the exact page or option involved, and any recent change you made before the issue appeared. That usually speeds up troubleshooting significantly.

Paquet Builder and related applications, documentation, and resources are copyright (c) G.D.G. Software, 2006-present. All trademarks and registered trademarks remain the property of their respective owners.

## 5. Create Packages

---

This section covers the first decisions you make in Paquet Builder: what kind of package to create, how to structure the project, and how to complete the first build successfully.

### Best entry point

If you are opening Paquet Builder for the first time, start with the quick tutorial. If you need to decide between package types first, read the project creation topic.

#### Create a Project [→](#)

Choose the package type that matches what you want to distribute.

#### Quick Tutorial [→](#)

Build a simple self-extracting package from start to finish.

## 6. Quick Tutorial: Create a Simple Package

---

Paquet Builder includes many features, so the easiest way to learn it is to build a simple package first. This quick tutorial walks you through the shortest path from a blank project to a working `.exe`.

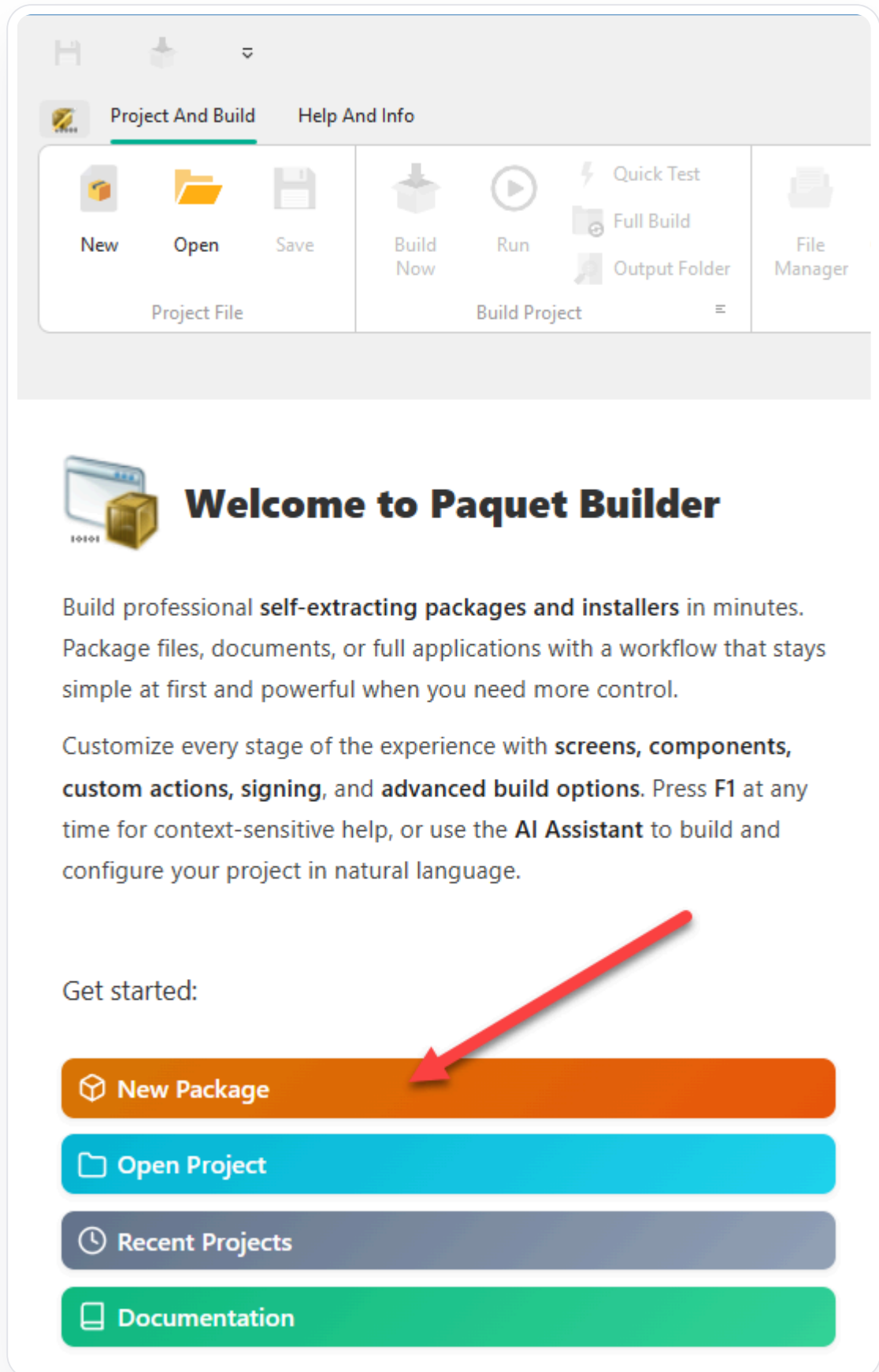
**Goal:** add source files to a package and compile the final output.

### Best use of this tutorial

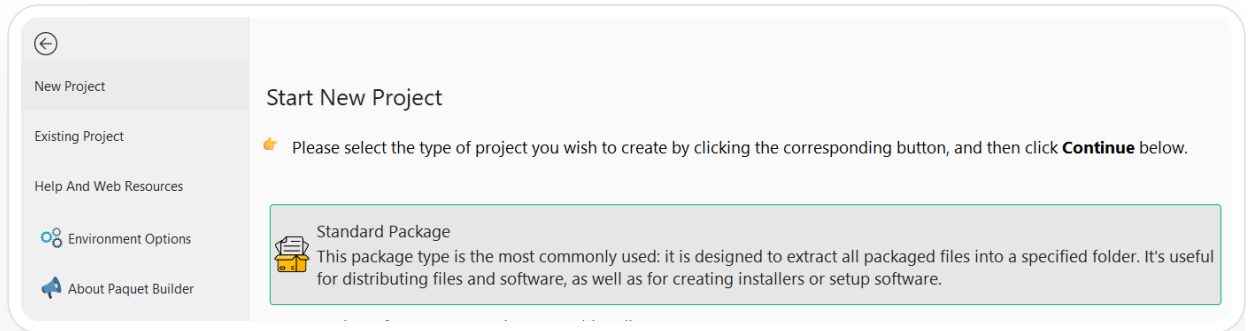
Keep Paquet Builder open while reading. Move through each step in the application, then come back here when you need the next screen.

## 1 Start Paquet Builder and create a new project

Open Paquet Builder. On the Home screen, click **New Package**.



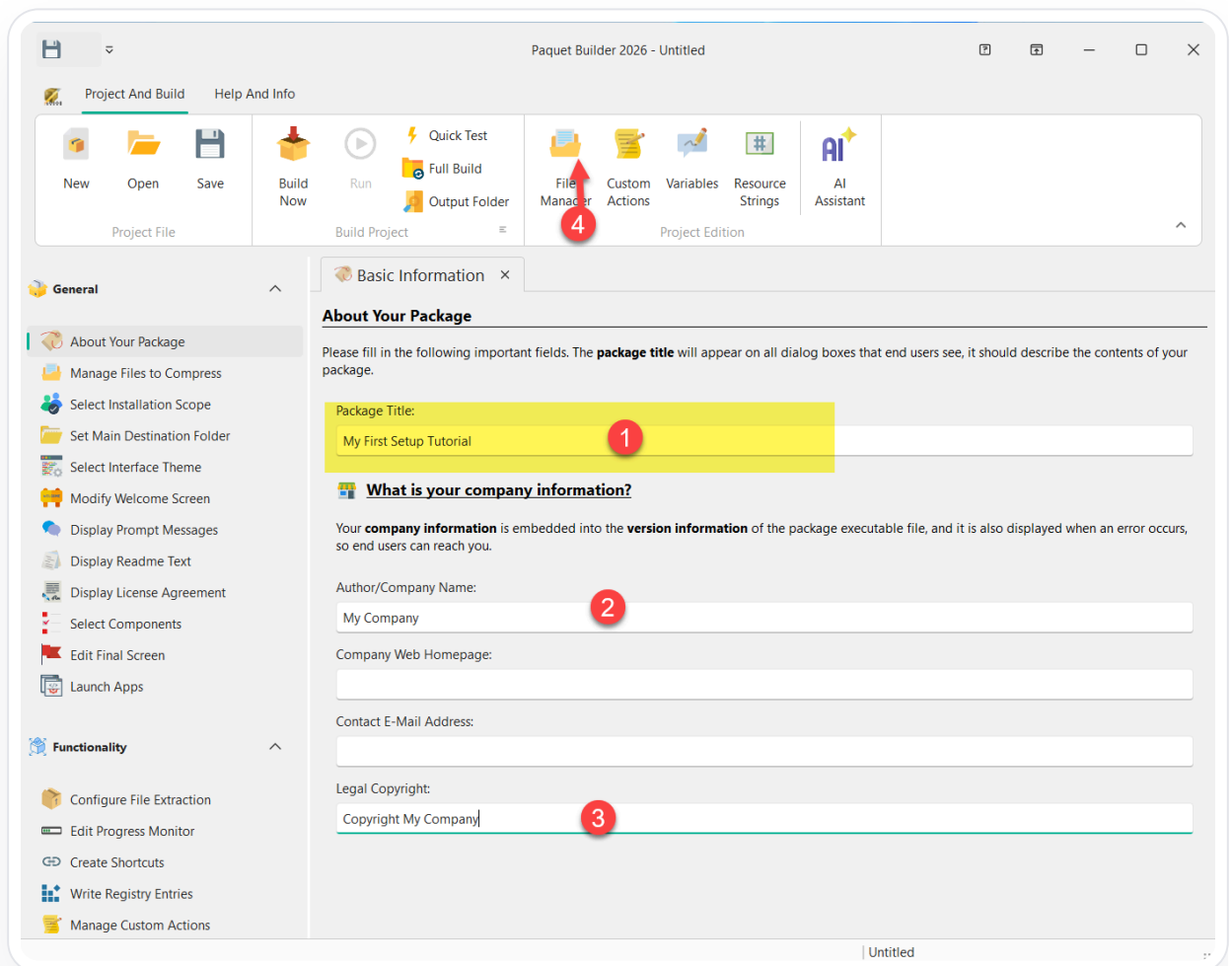
Select **Standard Package** — the most common type, used to extract files into a specified folder. Then click **Continue**.



## 2 Enter a package title

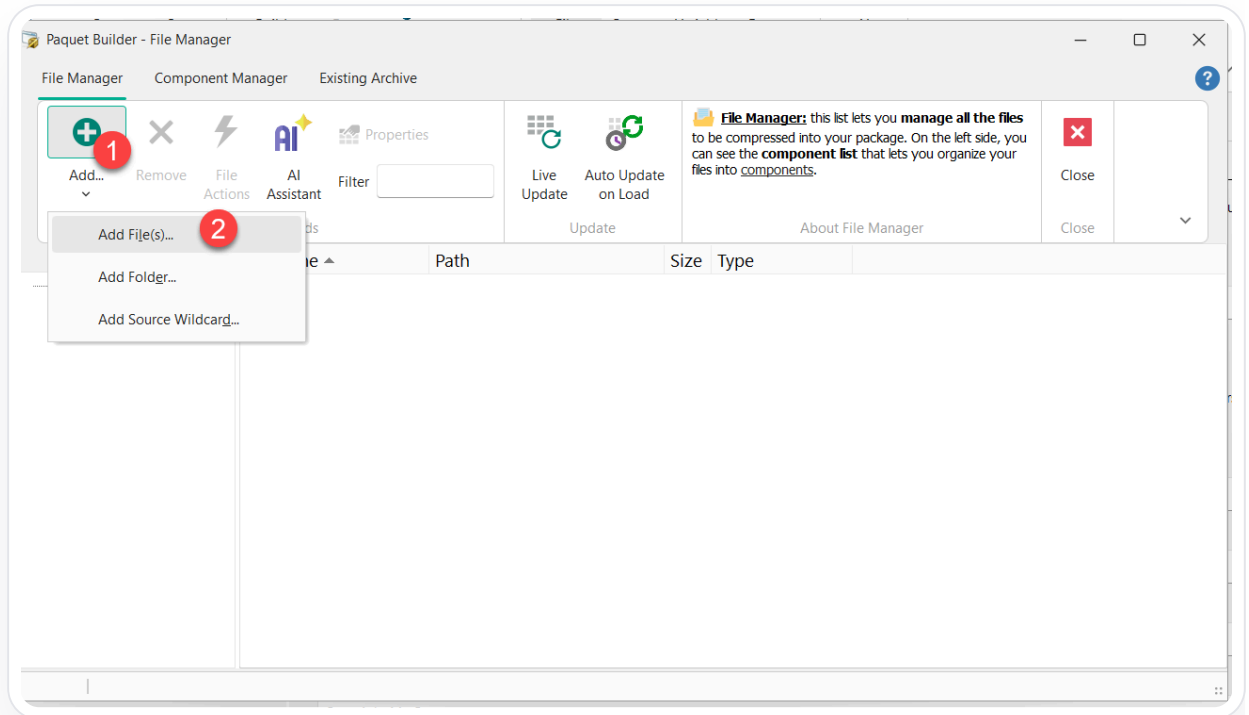
The project opens on the **About Your Package** page. Fill in the **Package Title** (e.g. *My First Setup Tutorial*) and optionally the **Author/Company Name** and **Legal Copyright** fields. This information is embedded in the executable's version metadata.

Use the **File Manager** button in the toolbar to open the File Manager at any time.

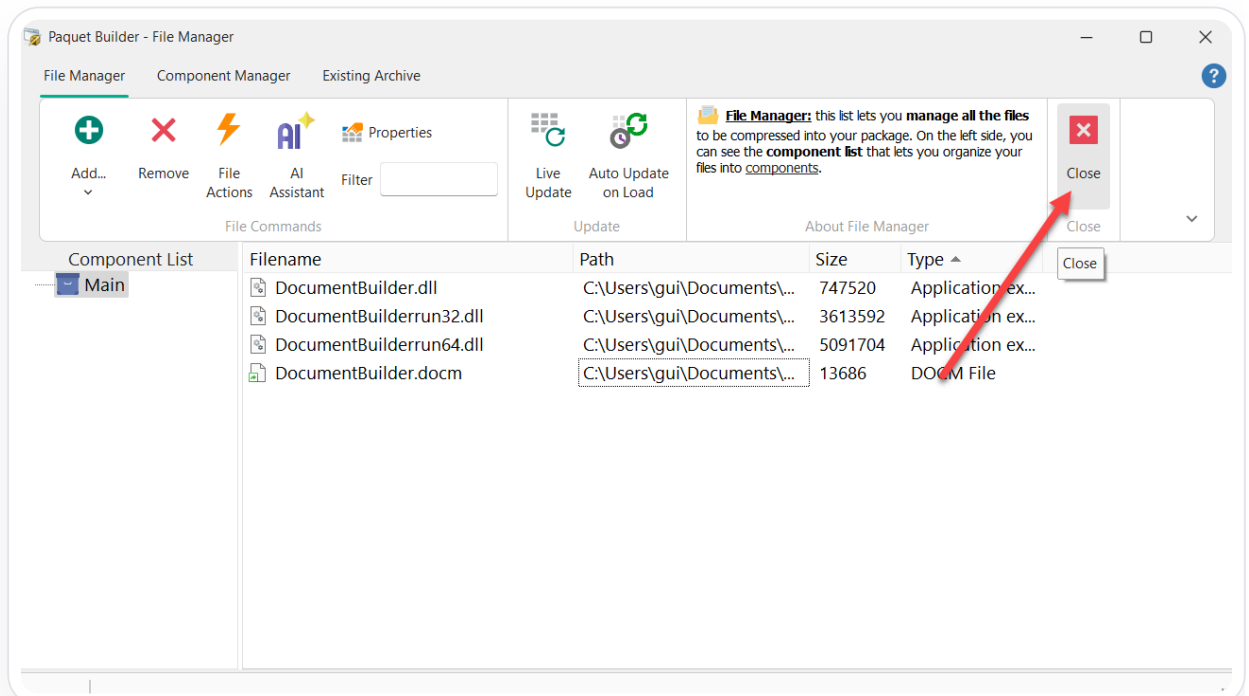


### 3 Add the files you want to distribute

In the **File Manager**, click **Add** then choose **Add File(s)** to browse for individual files, **Add Folder** to include an entire directory, or simply drag files from Windows Explorer into the list.



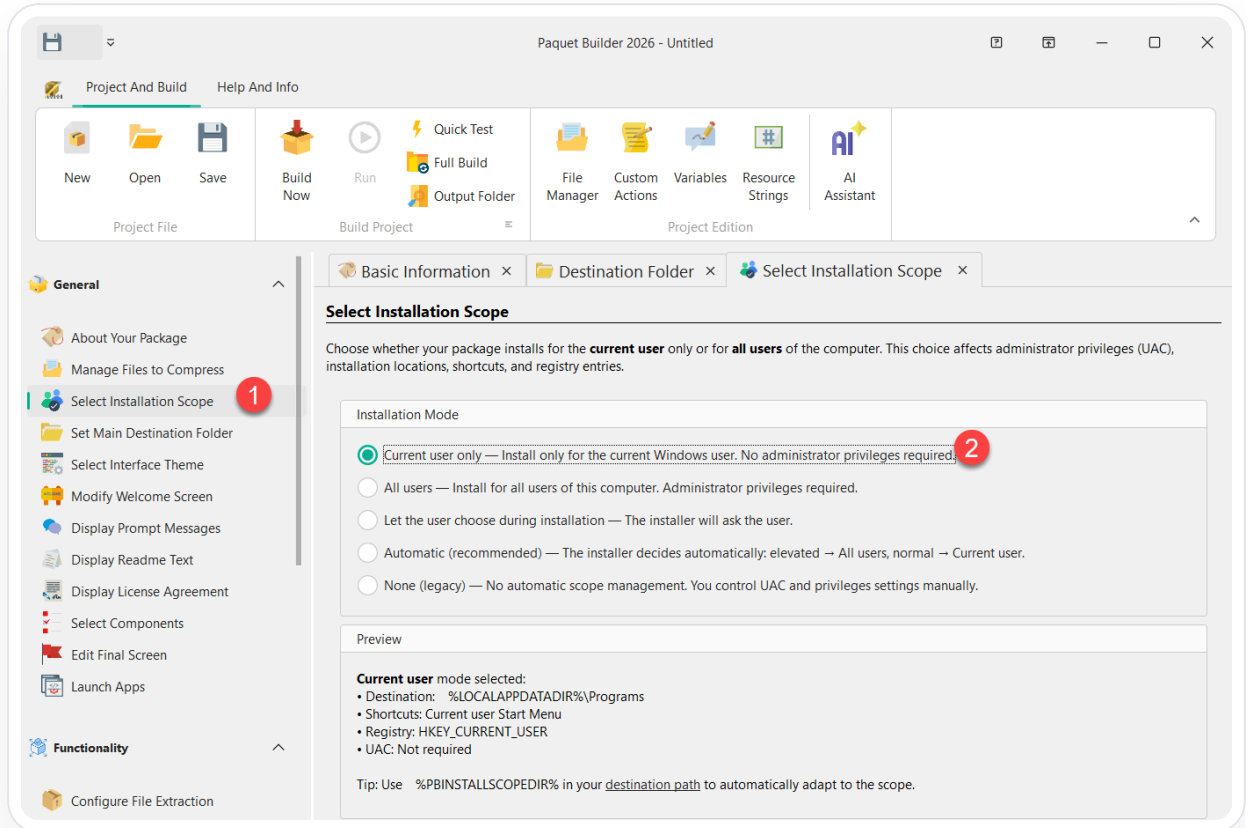
Once your files appear in the list, click **Close** to return to the main project view.



## 4 Select the installation scope

Navigate to **General > Select Installation Scope**. Choose whether the package installs for the **current user** only or for **all users**. This setting affects administrator privileges (UAC), destination paths, shortcuts, and registry entries.

For this tutorial, keep the default **Current user only** — no admin rights are required.

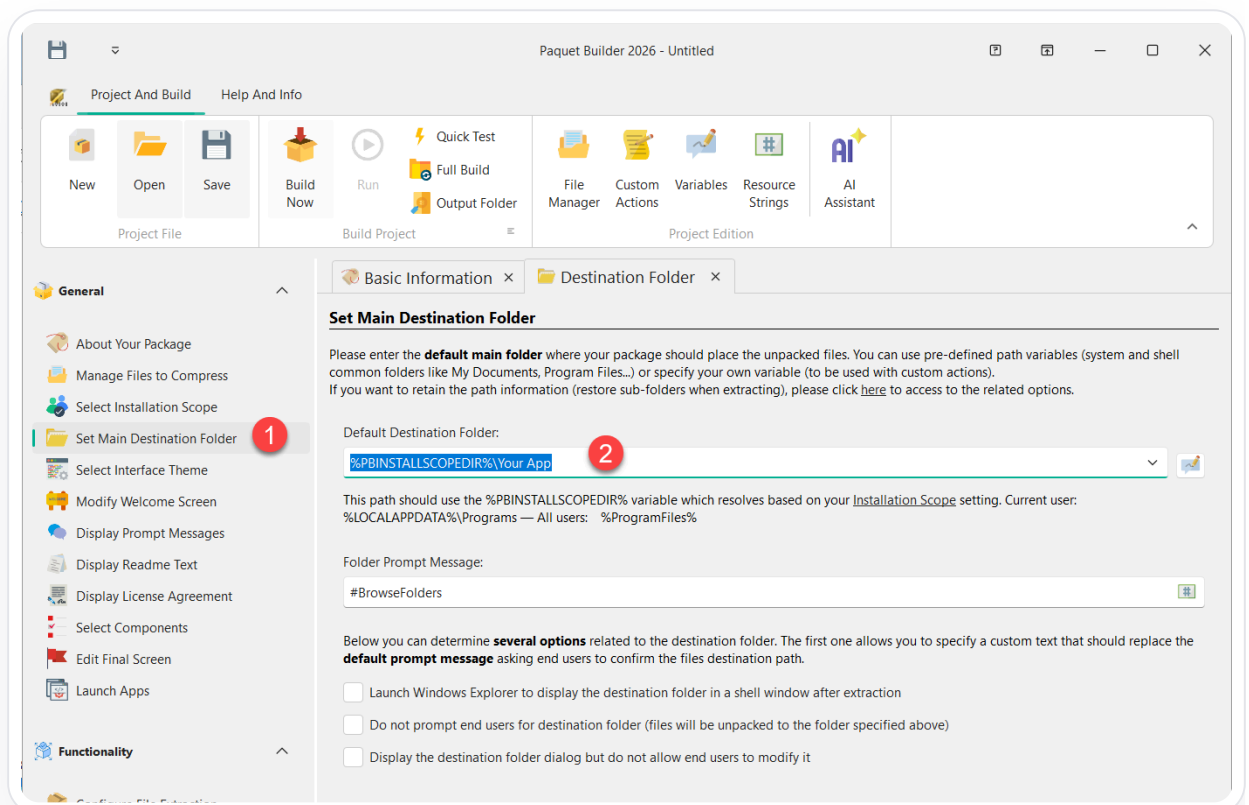


## 5 Set the destination folder

Navigate to **General > Set Main Destination Folder**. Enter the path where your files will be unpacked on the end user's machine. Use the `%PBINSTALLSCOPEDIR%` variable so the path adapts automatically to the installation scope you selected.

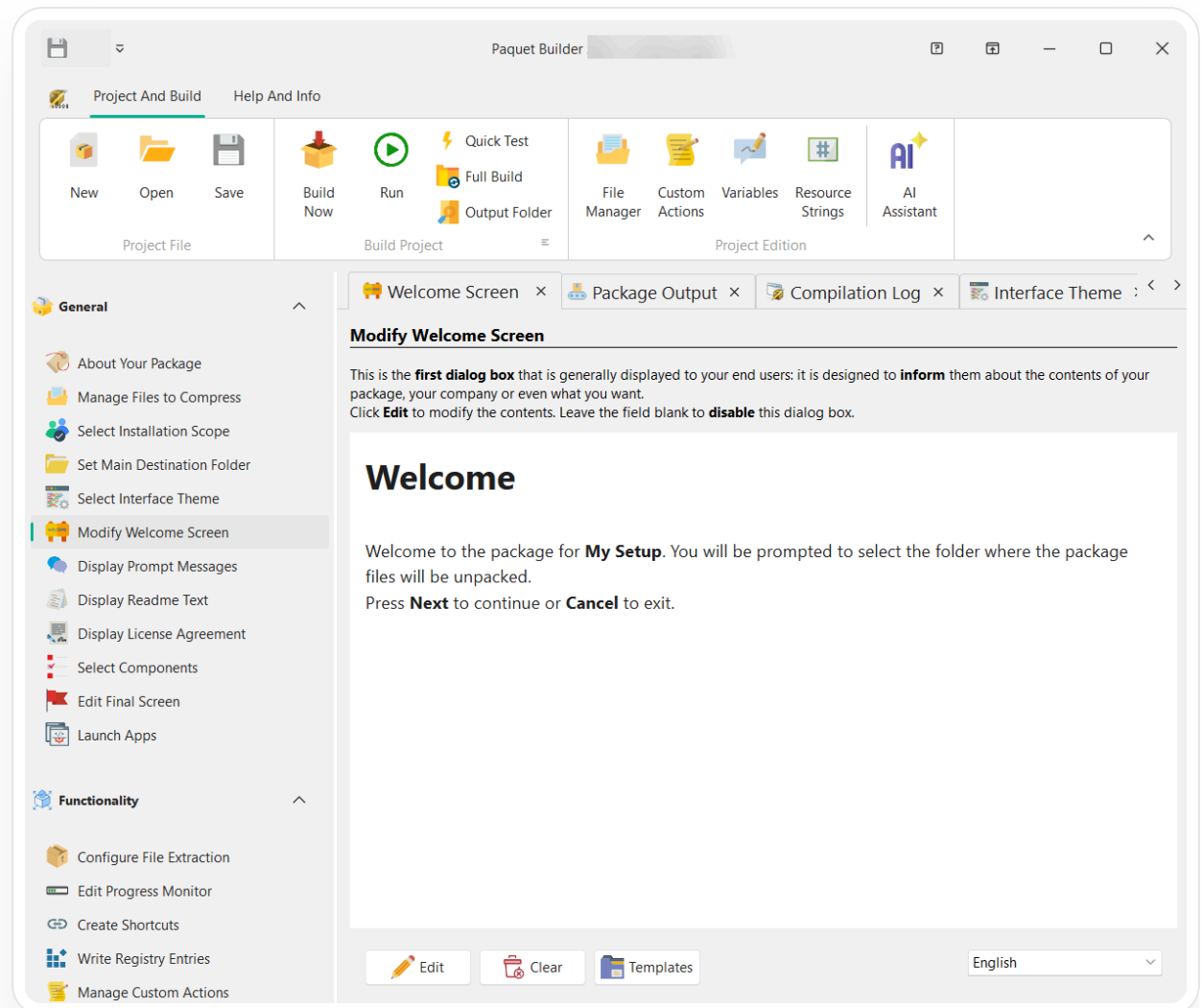
```
%PBINSTALLSCOPEDIR%\Your App
```

For a current-user install, this resolves to something like `C:\Users\John\AppData\Local\Programs\Your App`.

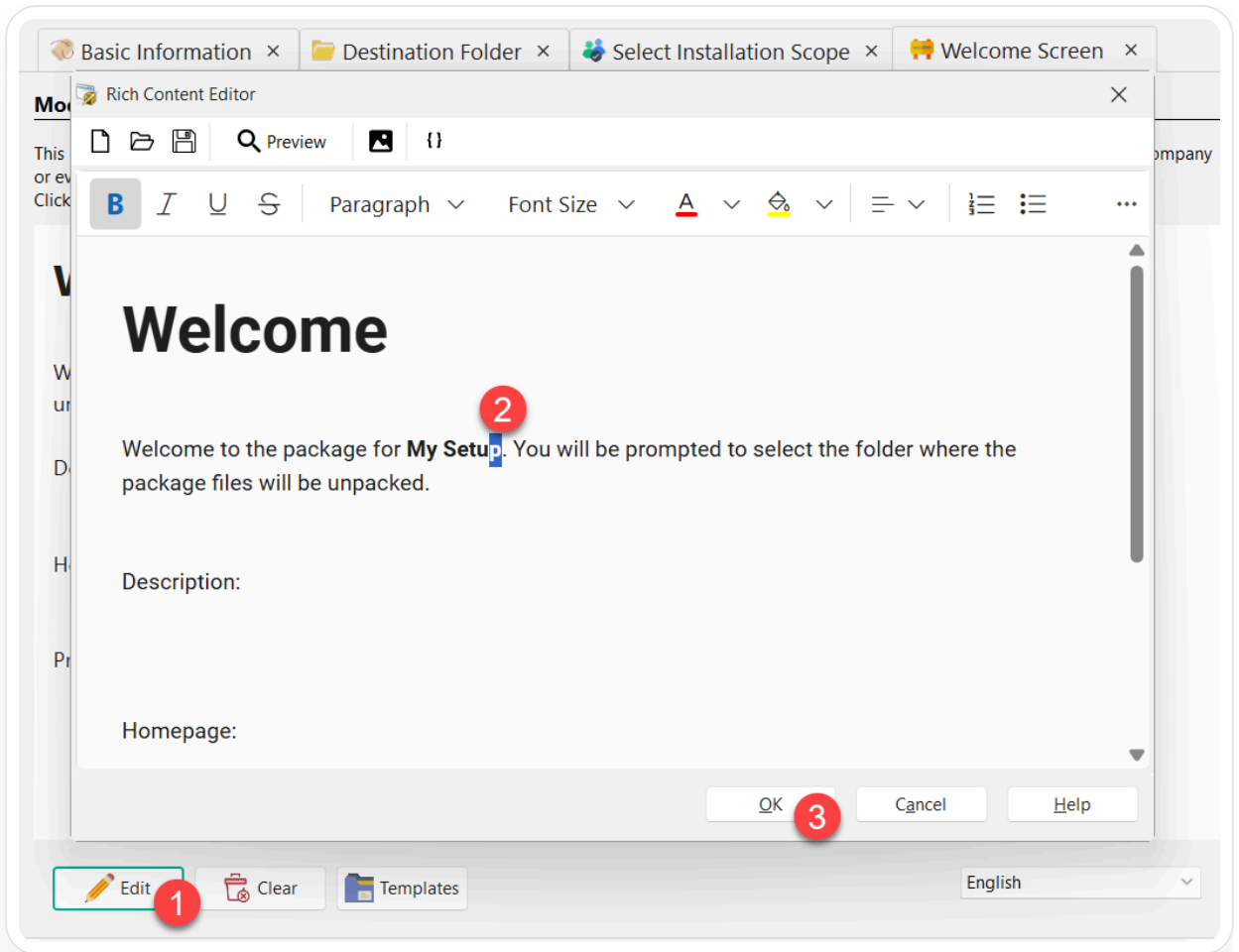


## 6 Add a welcome screen (optional)

Navigate to **General > Modify Welcome Screen**. Choose **Template — Package files will be extracted**, then click **Edit** to open the Rich Content Editor.

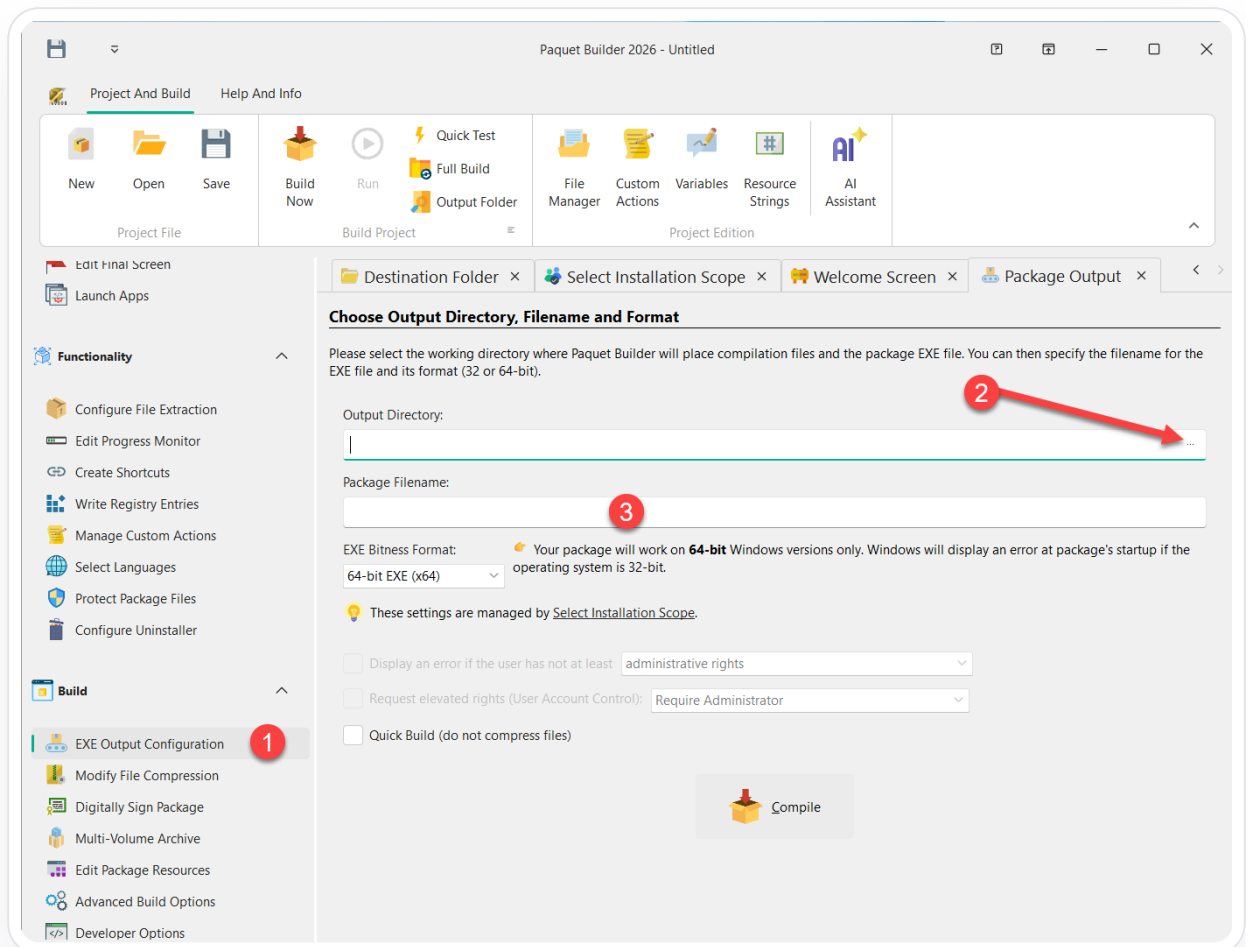


Write a short welcome message — the title and body text are fully customizable with rich formatting. Click **OK** when done.

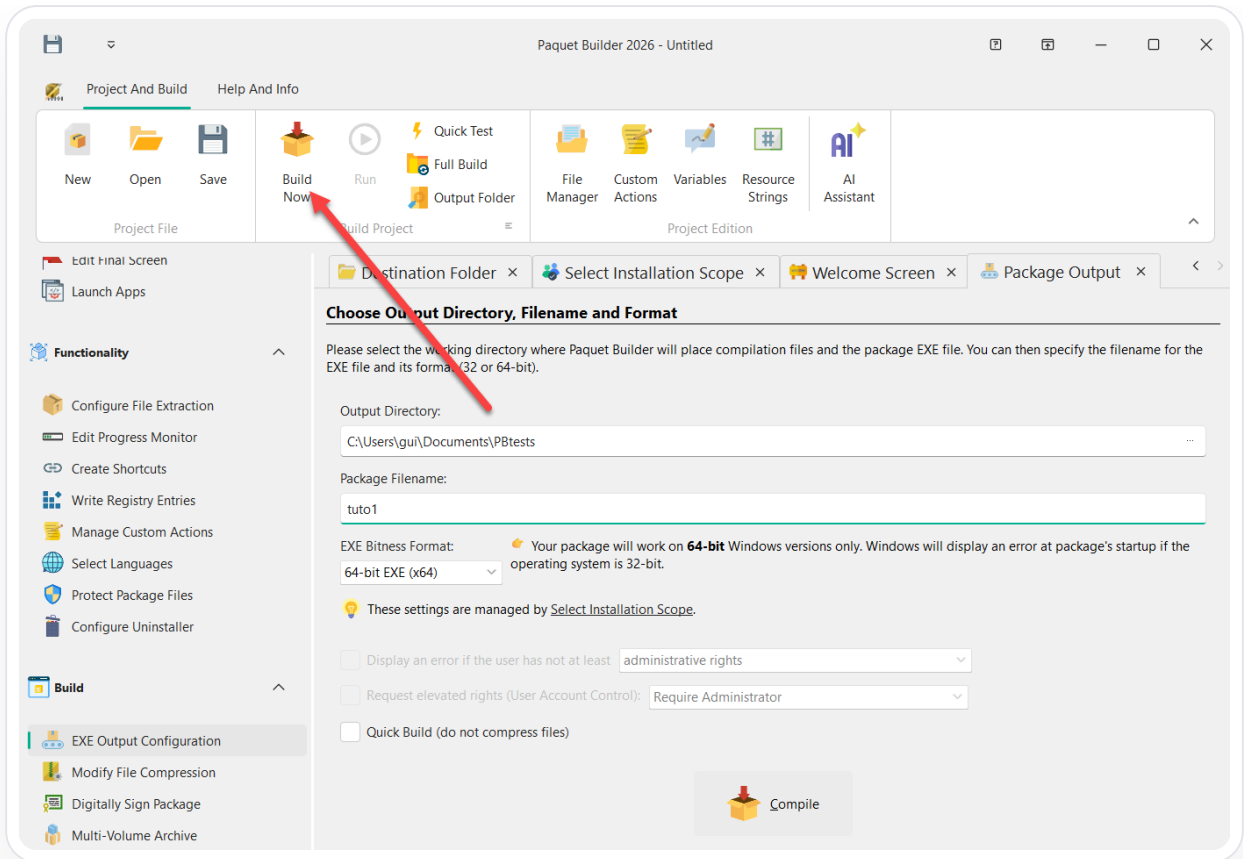


## 7 Configure the output and compile

Navigate to **Build > EXE Output Configuration**. Set the **Output Directory** (where the compiled file will be saved) and the **Package Filename**.

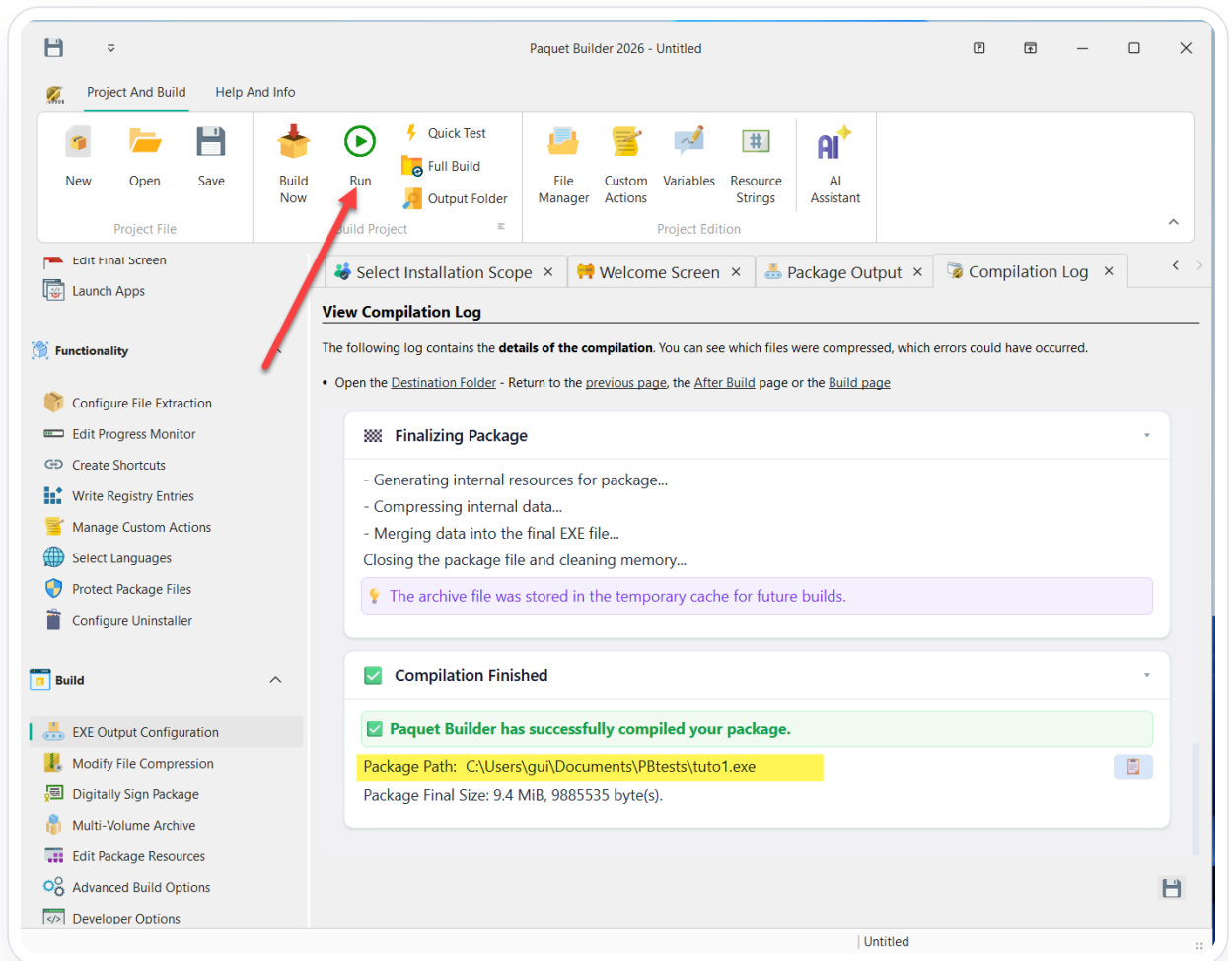


Click **Build Now** in the toolbar (or press the **Compile** button at the bottom of the page) to build the package.



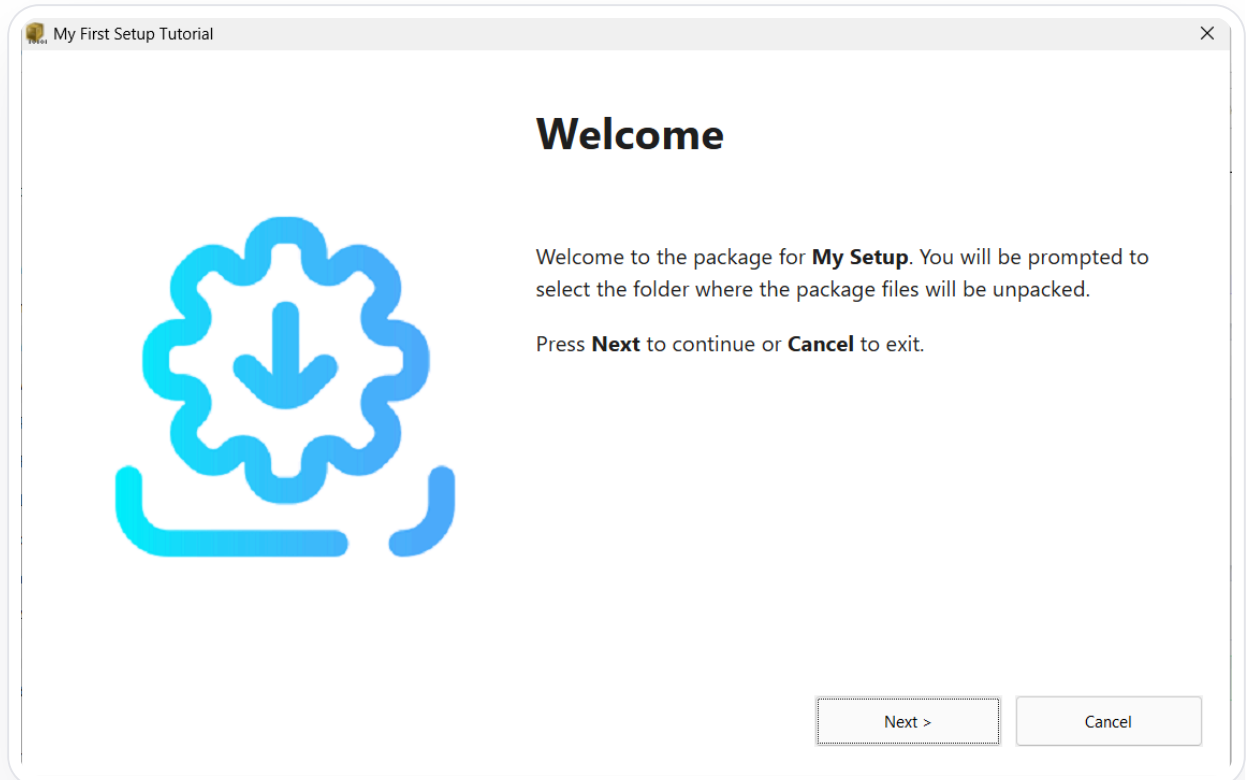
## 8 Check the compilation result

The **Compilation Log** tab opens automatically. When the build succeeds, you see a green confirmation with the package path and final file size. You can now click **Run** in the toolbar to test it.

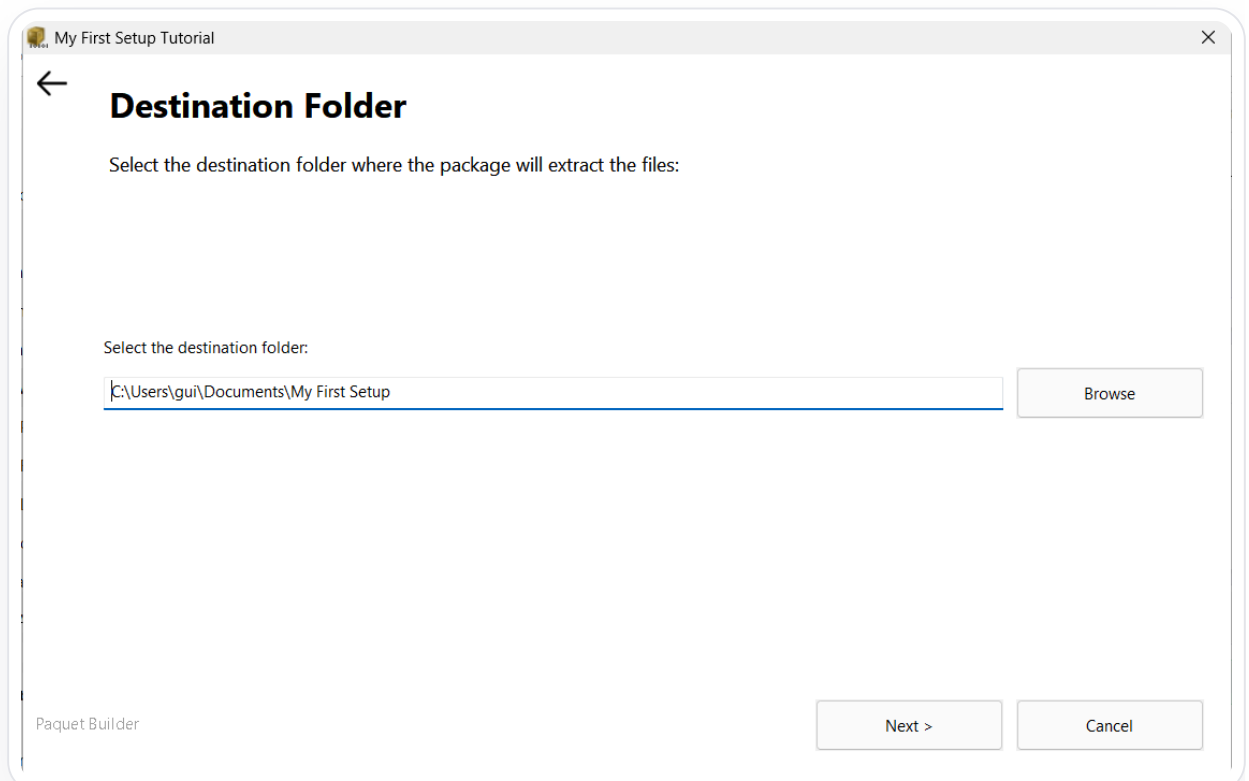


## 9 Test the result

Running the compiled .exe launches your package. The end user sees the **Welcome Screen** you configured:



After clicking **Next**, the user is prompted to confirm or change the **Destination Folder** before extraction begins:



Once the package builds successfully, you can continue by customizing the package appearance, adding additional screens, or enabling more advanced behavior.

## 6.1 Continue From Here

### Create a Project →

Learn the available package types and when to use each one.

### General Settings →

Customize package title, destination paths, screens, and file management.

### Functionality →

Add shortcuts, extraction rules, registry entries, and language support.

### More Tutorials →

Browse additional guided walkthroughs on the Paquet Builder website.

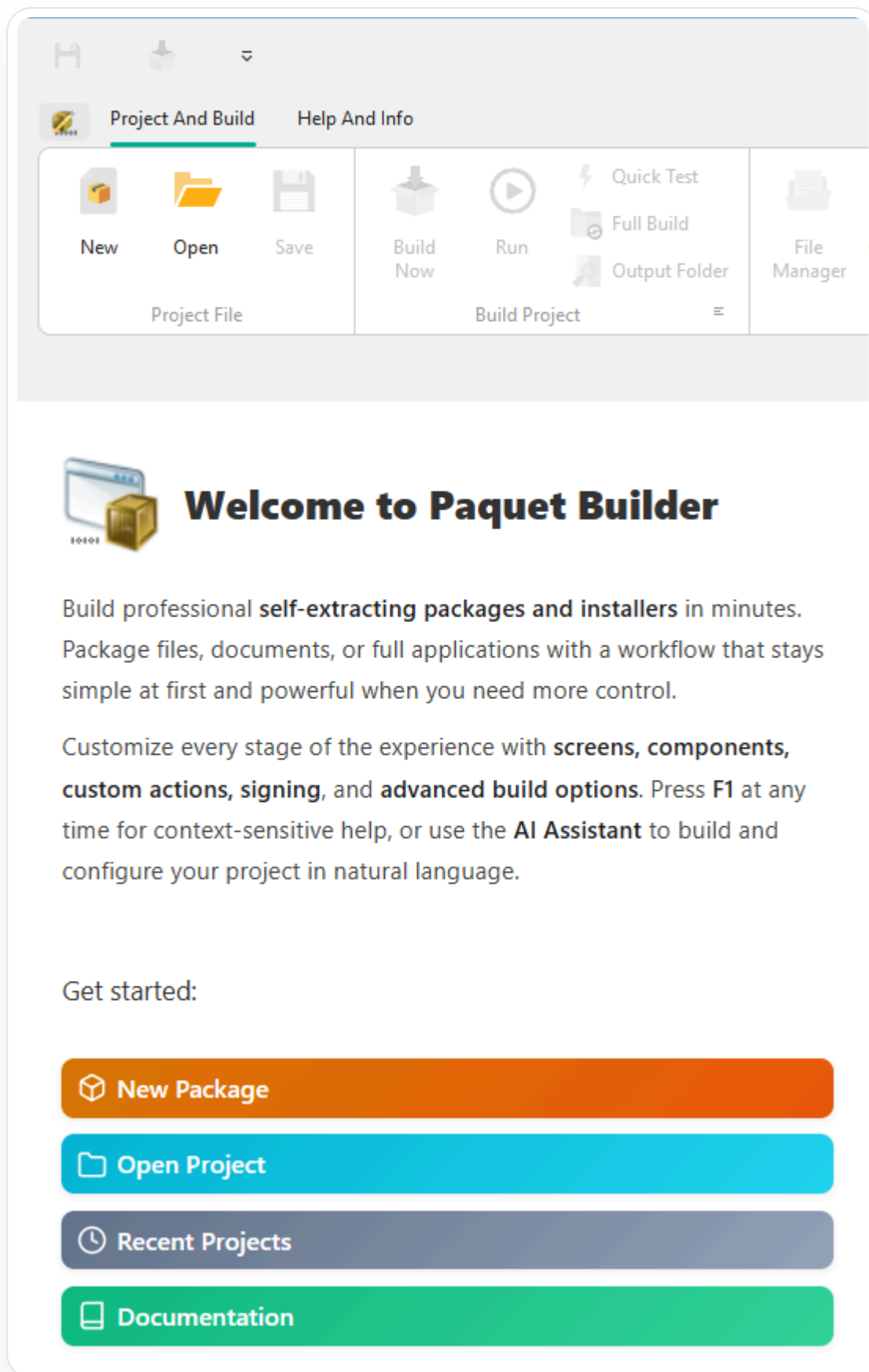
## 7. Create a Project

---

Every package starts with a **project file** (.pbpx). This file stores all of your settings — files to include, destination paths, screens, build options, and more. You can save, reopen, and share project files at any time.

### 7.1 Starting a New Project

- 1 Open Paquet Builder and click **New Package** on the Welcome Screen — or select **New Project** from the Application Menu at the top-left.



- 2 Select a **package type** (see below), then click **Continue**.

- 3 Fill in the **basic information** fields to get started.

## 7.2 Choose the Right Package Type

### Caution

The package type cannot be changed after the project is created. Read each description carefully before choosing.

## The default choice — recommended in 95% of cases.

A Standard Package extracts files into a folder chosen by you or the end user. It is ideal for distributing files, programs, and documents, and it supports the broadest set of features:



### File destinations

Assign **individual destination paths** to specific files or groups.



### Shortcuts

Create Start Menu and Desktop **shortcuts** automatically.



### Uninstaller

Generate a built-in **package uninstaller** with Add/Remove Programs registration.



### Full runtime logic

Use custom actions, registry operations, conditions, and user prompts.



### Tip

If you want to create a brand-new setup routine for a software program, **choose Standard Package** — not the Setup Routine type.

## For wrapping an existing installer into a single .exe.

Also known as “Package for a Multimedia Program,” this type is designed for software developers who already have a setup program (MSI, InstallShield, PowerPoint Viewer, etc.) and want to package it for easy distribution over the Internet, networks, or removable media.

How it works:

- Files are extracted to a **temporary folder**.
- The specified installation program is **launched automatically**.
- Once the program finishes, the temporary files are **removed**.



### Note

This type is recommended when you already have a working setup routine, such as a **Microsoft Windows Installer (MSI)** package, an InstallShield setup, or a multimedia presentation.

## For creating a standard 7-Zip archive (.7z).

This type compresses your files into a traditional 7-Zip archive without self-extracting functionality. A third-party tool such as 7-Zip is required to unpack the contents.

### Note

The resulting archive can still be used later in Paquet Builder through the “[Existing Archive](#)” option in the File Manager.

## 7.3 At a Glance

| Feature                     | Standard Package | Setup Routine  | Archive File |
|-----------------------------|------------------|----------------|--------------|
| Self-extracting .exe        | Yes              | Yes            | No (.7z)     |
| Custom destination folder   | Yes              | Temporary only | —            |
| Welcome screen and dialogs  | Yes              | Yes            | —            |
| Shortcuts and uninstaller   | Yes              | —              | —            |
| Custom actions and registry | Yes              | Limited        | —            |
| File-level destinations     | Yes              | —              | —            |

## 7.4 Continue From Here

### Quick Tutorial

Build your first package step by step.

### About Your Package

Fill in the package title and company metadata.

### General Settings

Configure destinations, screens, file management, and themes.

### Go Further

Explore components, variables, custom destinations, and advanced features.

## 8. General Settings

The **General** section is where most projects start. It contains the package identity, source content, destination folder, and the screens that shape the end-user experience.

### Good mental model

If a setting defines what the package is, what it contains, or what the user sees first, it is usually in **General Settings**.

#### About your package

Define package title, company information, and base metadata.

#### File Manager

Add, remove, and organize the files included in the package.

#### Select Installation Scope

Choose per-user or all-users installation to control paths, shortcuts, and privileges.

#### Main destination folder

Choose where files are unpacked or installed on the target system.

#### Welcome screen

Create the first screen users see when the package starts.

#### Readme and license agreement

Display informational or legal content before extraction begins.

#### Select components

Let users choose optional groups of files during installation.

#### Program setup execution

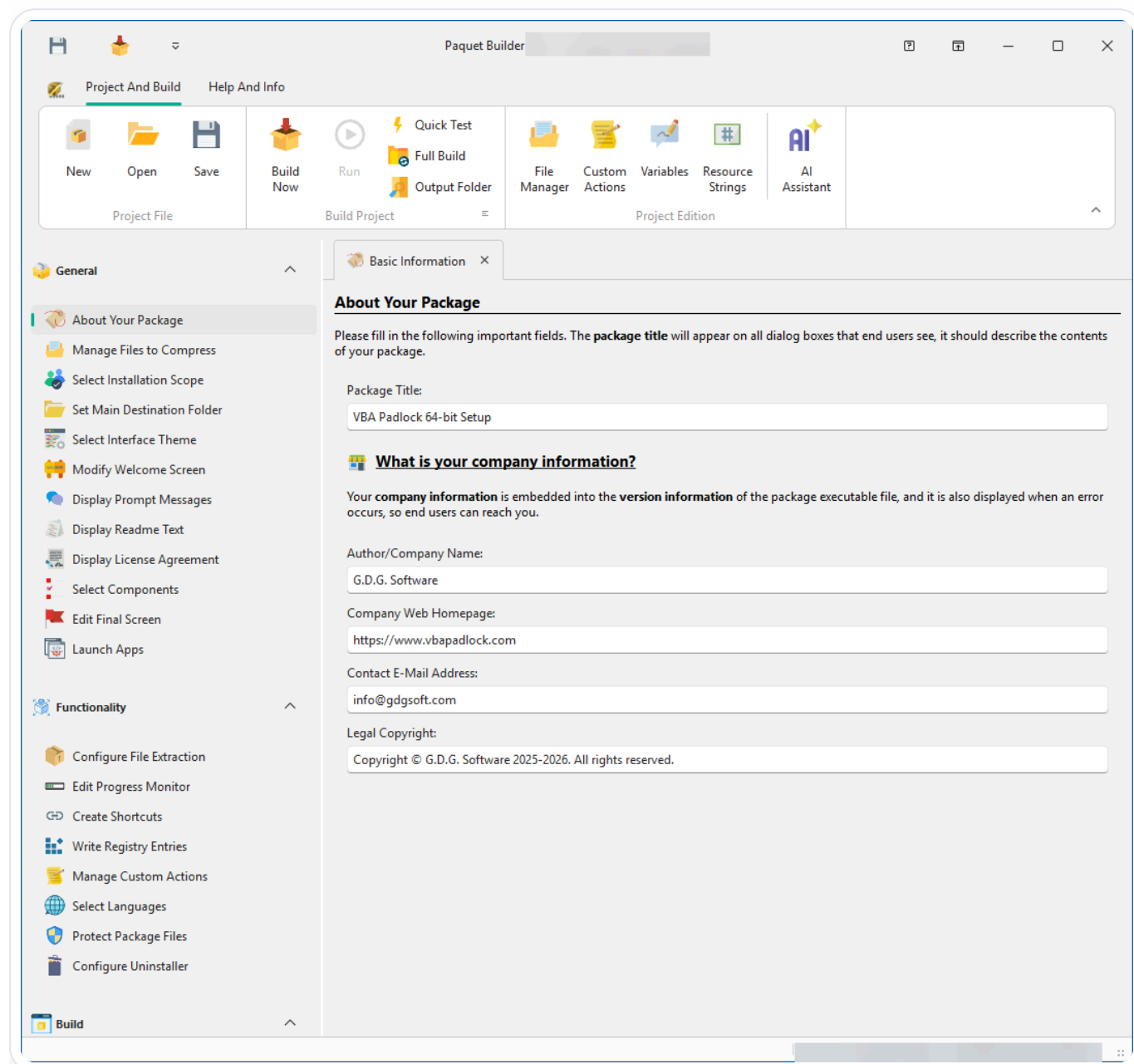
Run external setup programs after extraction when required.

#### End screen

Configure the final page displayed when the package completes.

## 9. About Your Package

This page defines the **basic identity** of your package — the title shown to end users and the company metadata embedded in the executable.



### 9.1 Package Title

The package title is **required**. It appears in window title bars, message boxes, and dialog headers during extraction. Choose a short, descriptive phrase:

*My Program Setup — My New Site Archive — Company Updater v3*

## 9.2 Company Information



### Author / Company Name

Identifies the publisher. Displayed in error messages, the About box, and [version information](#).



### Company Web Homepage

A URL where end users can find more information about your product or company.



### Contact E-Mail Address

Lets end users reach you for support. Highly recommended.



### Legal Copyright

Embedded in the [Version Information](#) of the package EXE and the [uninstaller](#) EXE. Use a format like: *Copyright 2026 by Your Company. All rights reserved.*

### Note

These details can be pre-set in the [Environment Options](#) so they are automatically applied each time you create a new package.



### Tip

At a minimum, fill in the **company name** — but providing all fields gives your package a more professional and trustworthy appearance to end users.

## 9.3 Related Pages

### Package Resources



Configure version information, icons, and manifest settings embedded in the EXE.

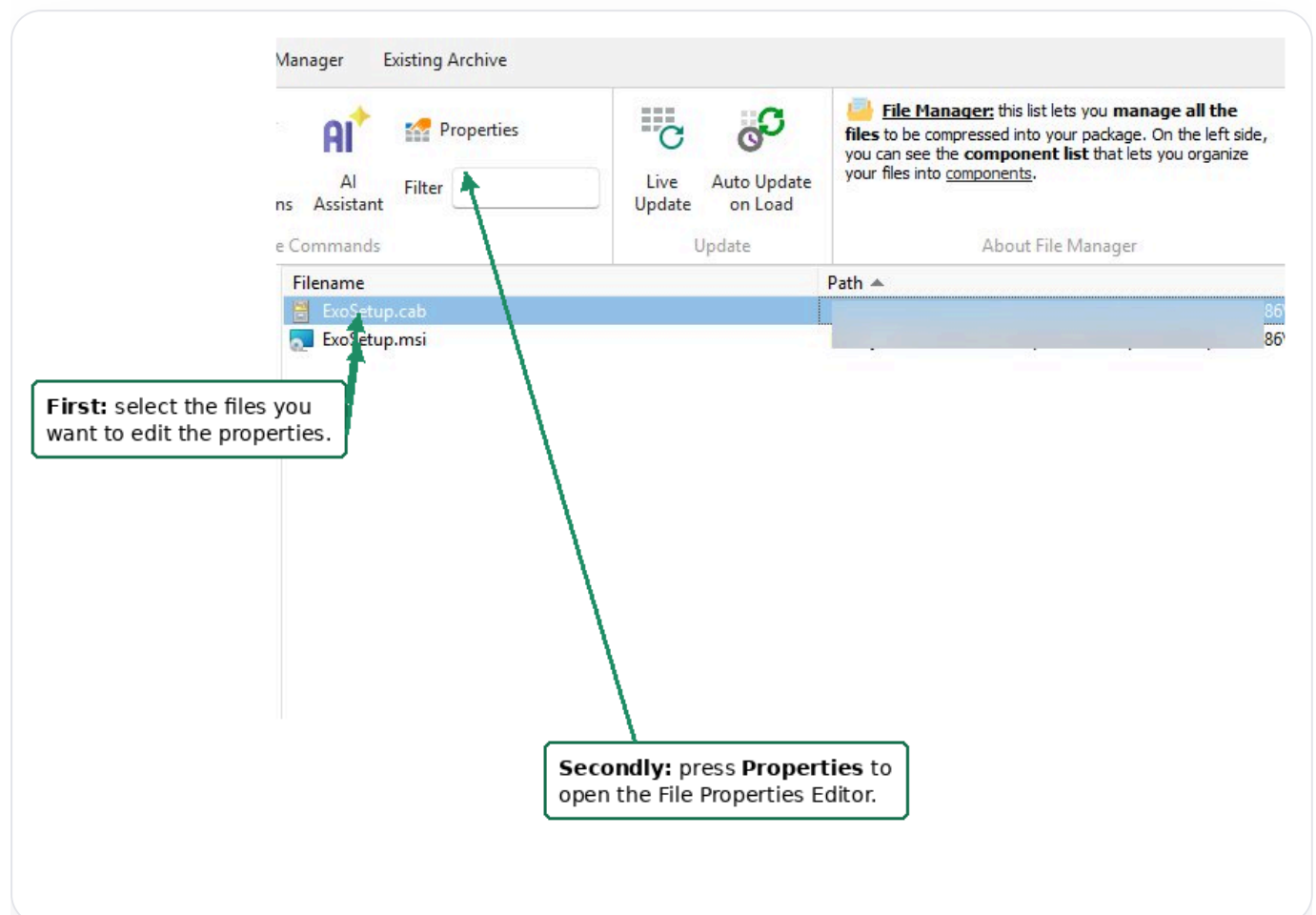
### Environment Options



Set default company information for all new projects.

# 10. File Manager

The File Manager is the central workspace where you select the files to include in your package. It is split into two panels: the **Component Manager** on the left and the **file list** on the right.



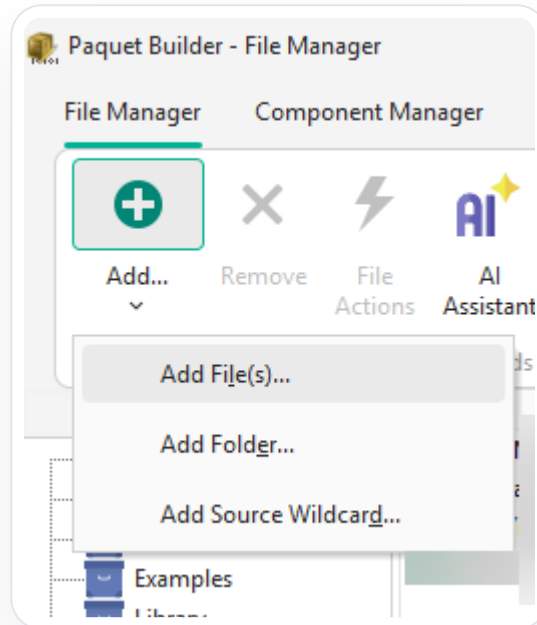
## Tip

**Components** are categories that group related files together. New projects start with a single "Main" component. Learn more about [components](#).

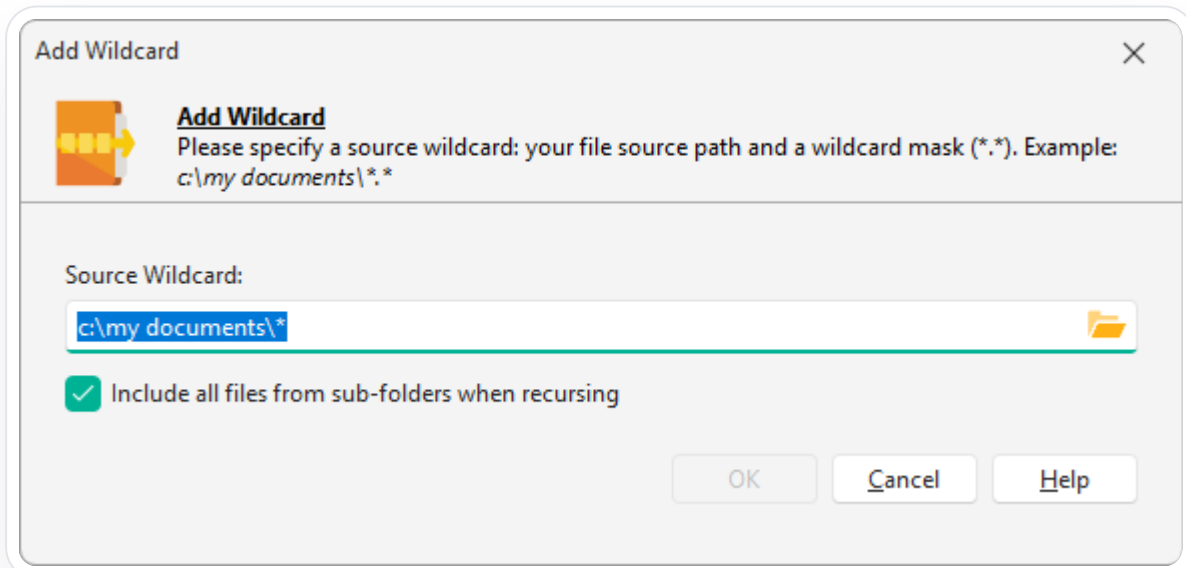
## 10.1 Adding Files

Click the **Add** button in the toolbar and choose an option:

- **Add File(s)** — browse for individual files.
- **Add Folder** — include an entire directory and optionally its sub-folders.
- **Add Source Wildcard** — define a mask (e.g. \*.dll) to match multiple files dynamically.



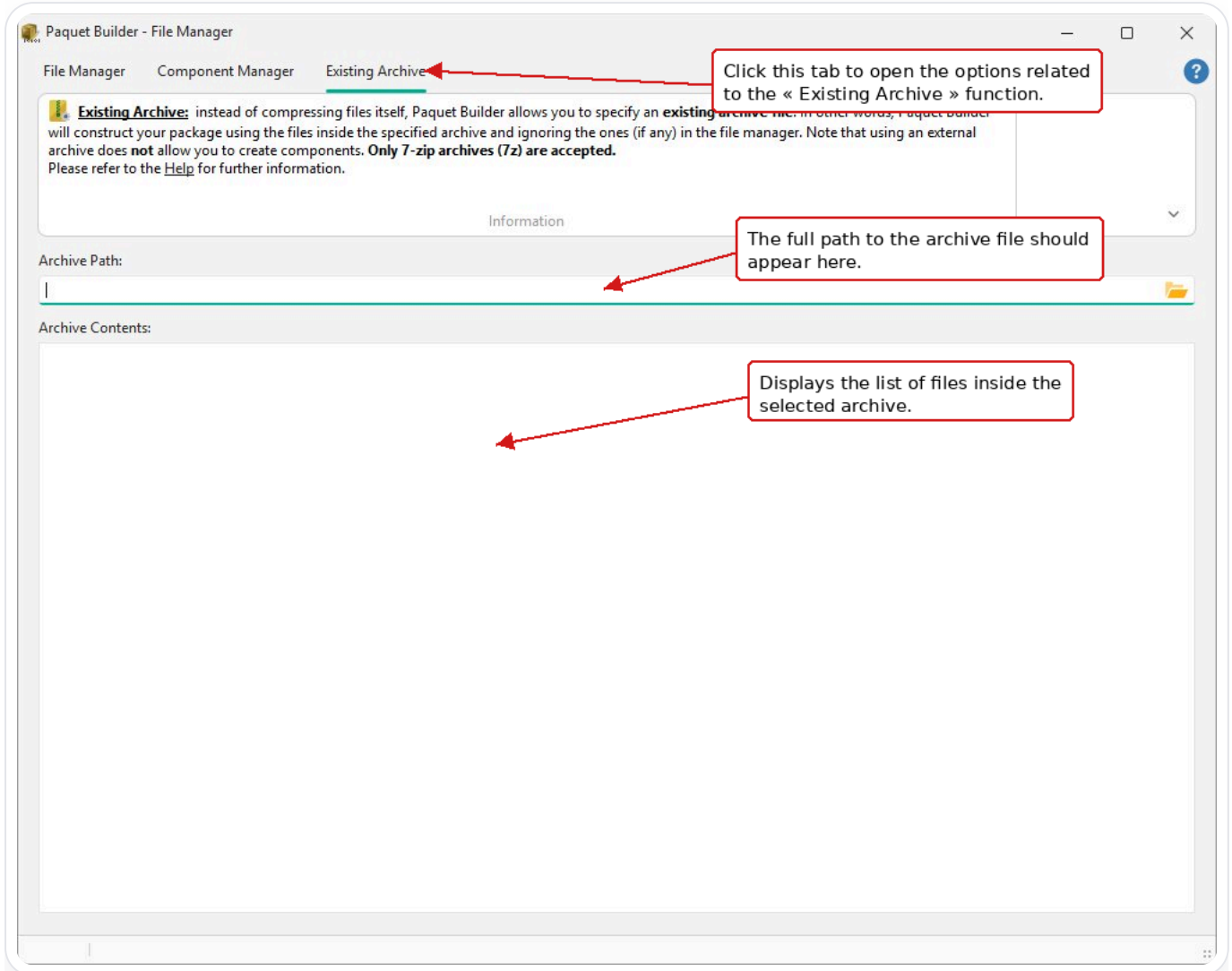
Wildcards let you define masks for files to include. This is useful for dynamic content that changes between builds.



When adding folders or wildcards, you can choose whether to **include sub-folders**. You can also configure components for **live-update** to refresh sub-folder contents automatically.

Drag files directly from **Windows Explorer** into the file list — they are added to the currently selected component.

Instead of individual files, you can use an existing **7-Zip archive** (.7z) as your package source. Open the **Existing Archive** tab and specify the archive path:



### ⚠ Caution

When an archive is used, files added to components are ignored. A warning appears in the compilation log.

### ⚠ Caution

This feature is incompatible with the [multi-volume archive option](#).

### 💡 Tip

The “Existing Archive” feature works well with the [Paquet Builder shell extension for Windows](#) — a free add-on that converts 7-Zip archives into Self-Extracting packages directly from Windows Explorer.

## 10.2 Managing Files

## Remove

Select files and click **Remove**, or right-click and choose **Clear file list** to remove all files at once.

## Sort and filter

Click column headers to sort by name, type, or size. Use the **Filter** field to narrow the list (e.g. \*.EXE).

## Properties

Select one or more files and use the **Properties** button, context menu, or double-click to edit **file properties**.

## Context menu

Right-click for commands like **Select All**, **File Information**, **Open**, or **Create a shortcut to...** (editable later in the **shortcut manager**).

## 10.3 Component Manager

The Component Manager (left panel) lets you organize files into logical groups:

- 1 Click **Add Component** in the ribbon and provide a title.
- 2 Select a component to view and manage its files in the right panel.
- 3 Use **Properties** to configure **component settings** (optional, required, folder-linked, etc.).

Additional actions:

- **Remove** — delete the selected component and its files.
- **Component Wizard** — let Paquet Builder **generate components automatically**.
- **List Update** — refresh **folder-linked components**.
- **Right-click** a component for options like **Rename Component**.

## 10.4 File Actions

The **File Actions** button provides registration options for shared and COM files:

| Action                                  | Description  |
|---|--|
| <b>Register as ActiveX / COM Server</b> | Creates a <b>custom action</b> to self-register an EXE or DLL for use by other COM applications.   |
| <b>Register as Shared DLL</b>           | Creates a <b>custom action</b> to mark the file as shared. During uninstallation, the system checks whether it is still in use before removing it. |

## 10.5 Related Pages

### Components →

Understand how components work and configure their properties.

### Auto-Create Components →

Use the Component Wizard to generate components from folder structures.

### File Destinations →

Assign individual destination paths to specific files.

### Create Shortcuts →

Configure Start Menu and Desktop shortcuts for your files.

# 11. Select Installation Scope

Installation scope determines **who** the installer targets on the machine. This single setting automatically controls destination paths, shortcut locations, registry root keys, and User Account Control (UAC) elevation requirements.

Access these settings from the **Installation Scope** page in the navigation panel.

The screenshot displays the Paquet Builder 2026 application window. The title bar reads "Paquet Builder 2026 - 1.pbpx". The interface is divided into several sections:

- Top Toolbar:** Contains icons for "Project And Build" (New, Open, Save) and "Help And Info".
- Main Workspace:**
  - Left Navigation Panel:** Lists various settings categories:
    - General:** About your package, Manage files to compress, **Select installation scope** (highlighted), Set main destination folder, Select interface theme, Modify welcome screen, Display prompt messages, Display readme text, Display license agreement, Select components, Edit final screen, Launch programs.
    - Functionality:** Configure file extraction, Edit progress monitor, Create shortcuts, Write registry entries, Manage custom actions, Select languages, Protect package files, Configure uninstaller.
    - Build:** EXE output configuration, Modify file compression, Digitally sign package, Multi-volume archive.
  - Right Panel (Tabs):** Includes "Welcome Screen", "Package Output", "Select Installation Scope" (active), "Code Signing", and "Compilation Log".
  - Content Area:**
    - Select Installation Scope:** A heading followed by a description: "Choose whether your package installs for the **current user** only or for **all users** of the computer. This choice affects administrator privileges (UAC), installation locations, shortcuts, and registry entries."
      - Installation Mode:** A group box containing five radio button options:
        - Current user only — Install only for the current Windows user. No administrator privileges required.
        - All users — Install for all users of this computer. Administrator privileges required.
        - Let the user choose during installation — The installer will ask the user.
        - Automatic (recommended) — The installer decides automatically: elevated → All users, normal → Current user.
        - None (legacy) — No automatic scope management. You control UAC and privileges settings manually.
      - Advanced Options:** A group box containing two checkboxes:
        - Automatically request elevation (UAC) when needed
        - Fail if administrator privileges are not available
      - Preview:** A text area showing the selected mode's configuration:
        - All users mode selected:**
          - Destination: %PROGFILESDIR%
          - Shortcuts: All Users Start Menu
          - Registry: HKEY\_LOCAL\_MACHINE
          - UAC: Administrator privileges required
        - Tip:** Use %PBINSTALLSCOPEDIR% in your destination path to automatically adapt to the scope.

## 11.1 Scope Modes

### ⚠ Caution

The scope mode cannot be changed at runtime by the end user — except when using **User Choice**, which presents a dialog. Choose the mode that best fits your deployment scenario.

Targets the active user's profile. No administrator rights required.

| Setting       | Value                   |
|---------------|-------------------------|
| Destination   | %LOCALAPPDATA%\Programs |
| Shortcuts     | Per-user folders        |
| Registry root | HKEY_CURRENT_USER       |
| UAC elevation | Not required            |

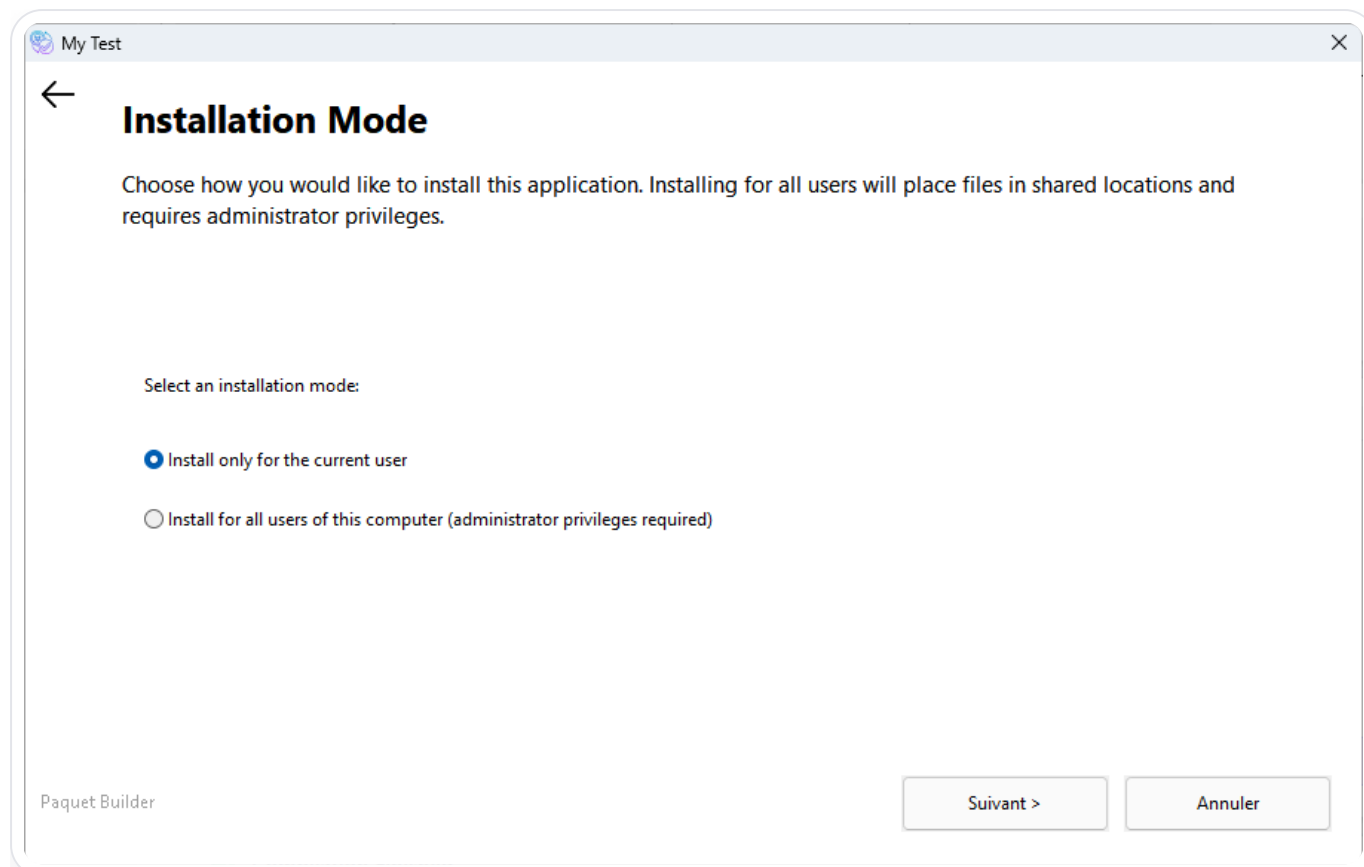
Best for **"no-install" style apps** that don't need system-level access.

Targets a shared location for every user on the machine. Requires administrator rights.

| Setting       | Value                   |
|---------------|-------------------------|
| Destination   | %PROGFILES_DIR%         |
| Shortcuts     | Common (shared) folders |
| Registry root | HKEY_LOCAL_MACHINE      |
| UAC elevation | Required                |

Mandatory for **services, drivers, or apps** that must be available to everyone.

Displays a dialog at runtime, letting the end user choose between Current User and All Users. Provides the best balance of flexibility and ease of use.



Best for **general-purpose applications** distributed online.

### Tip

This is the **recommended** mode for most packages.

Silently detects administrator privileges at runtime:

- **Elevated** — behaves as All Users (files to %PROFILES\_DIR%, shortcuts to common folders, registry to HKLM).
- **Not elevated** — falls back to Current User (files to %LOCALAPPDATA%\Programs, shortcuts to user folders, registry to HKCU).

No dialog is shown. Best for **silent deployments** and scripted enterprise installations.

No automatic scope management. All paths, shortcuts, and registry keys must be configured manually.

### Caution

Use only for backward compatibility with older projects. New projects should use one of the other modes.

## 11.2 Impact on Installer Behavior

When you select a scope mode, several core settings adapt automatically:



## Destination Folder

**Current User:** %LOCALAPPDATA%\Programs\YourApp **All Users:** %PROGRAMFILES%\YourApp



## Shortcuts

**Current User:** User-specific Start Menu and Desktop **All Users:** Common (shared) folders



## Registry Root

**Current User:** HKEY\_CURRENT\_USER **All Users:** HKEY\_LOCAL\_MACHINE



## UAC and Uninstaller

**Current User:** No elevation, user registry **All Users:** Elevation required, machine registry

### Note

Dependent pages (Package Output, Shortcuts, Uninstaller) update in real-time when the scope is changed in the project.

## 11.3 Scope Options

| Option                                  | Description   | Default  |
|---|---|----------|
| <b>Request UAC elevation</b>            | When the scope resolves to All Users, the installer requests UAC elevation automatically. Disable only if your installer does not need write access to protected locations.           | Enabled  |
| <b>Fail if admin rights unavailable</b> | The installation aborts with an error when All Users scope is required but administrator rights cannot be obtained. When disabled, the installer silently falls back to Current User. | Disabled |

## 11.4 User Choice — Under the Hood

### 11.4.1 UAC Self-Elevation Logic

If the end user selects **“All users”** but the installer is running with standard privileges, Paquet Builder handles the elevation transparently:

1

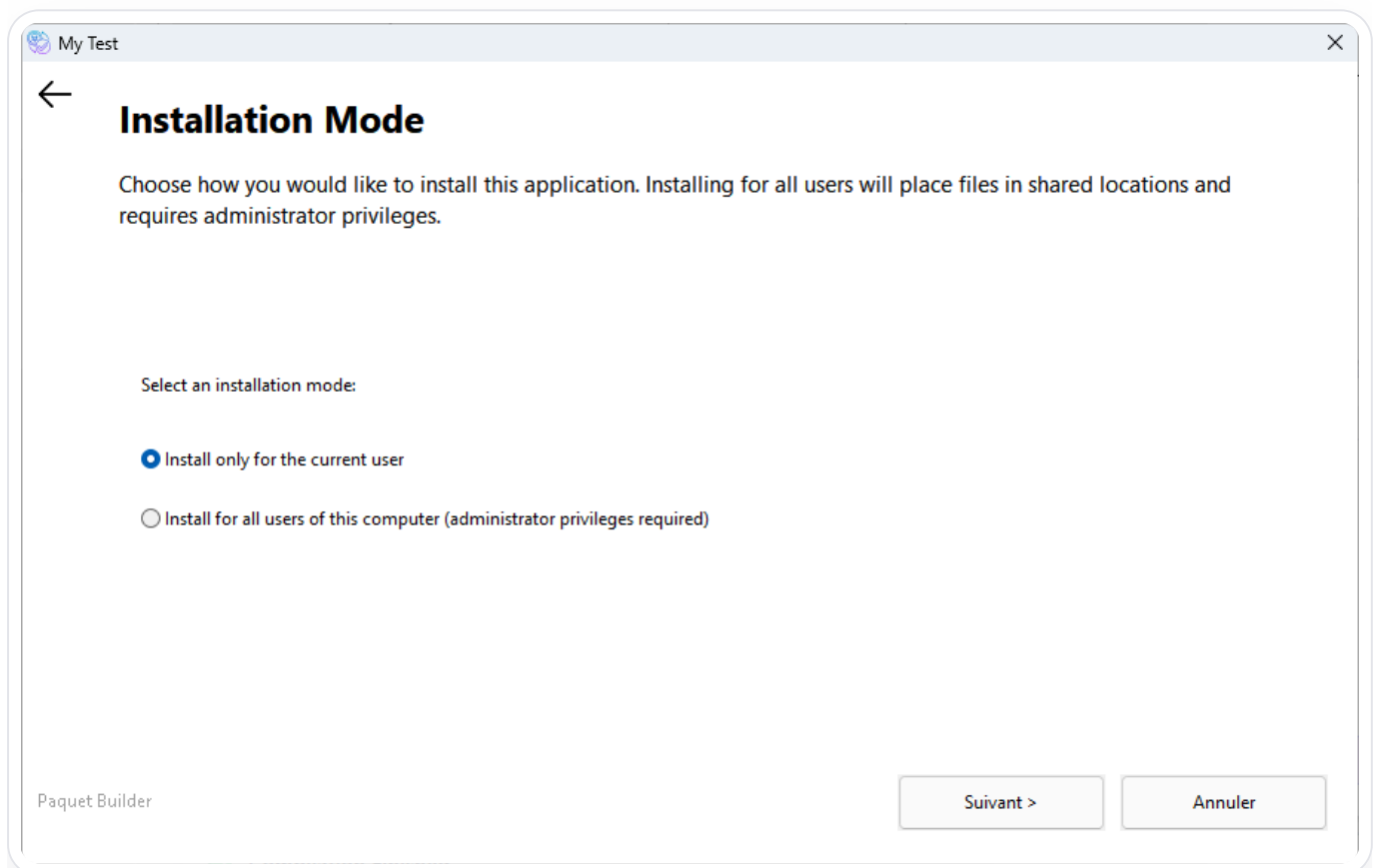
The installer re-launches itself with the `Run as administrator` verb.

- 2 All original command-line parameters are forwarded, along with `/PBSCOPE=1` which tells the elevated instance to use All Users scope directly.
- 3 The scope dialog is skipped in the elevated instance to prevent redundant prompts.
- 4 The installation proceeds using the All Users configuration.

### **Note**

`/PBSCOPE=1` is added internally during self-elevation. You do not need to pass it manually unless you want to force All Users scope from an external script.

## 11.4.2 Customizing the Dialog



The User Choice dialog uses five **Resource Strings** that can be translated or modified:

| Resource String                   | Purpose                          |
|-----------------------------------|----------------------------------|
| <code>#ScopeChoiceTitle</code>    | Dialog window title              |
| <code>#ScopeChoicePrompt</code>   | Main prompt text                 |
| <code>#ScopeChoiceAllUsers</code> | Label for the "All Users" option |

| Resource String         | Purpose                             |
|-------------------------|-------------------------------------|
| #ScopeChoiceCurrentUser | Label for the "Current User" option |
| #ScopeChoiceQuestion    | Additional explanatory text         |

## 11.5 Runtime Scope Variables

The following system [Variables](#) are resolved at runtime based on the active scope:

| Variable            | Description   |
|---------------------|---|
| %PBINSTALLSCOPE%    | Returns 0 for Current User or 1 for All Users             |
| %PBINSTALLSCOPEDIR% | Resolves to the correct base directory                    |
| %PBSHORTCUTROOT%    | Start Menu folder for the active scope                    |
| %PBDESKTOPROOT%     | Desktop folder for the active scope                       |
| %PBSCOPEELEVATED%   | Returns 1 if the installer was re-launched with elevation |

### Tip

Always use %PBINSTALLSCOPEDIR% in your destination paths instead of hardcoding folder locations. This ensures your installer remains scope-independent.

## 11.6 Directive Support

When using the [Console Command-Line Compiler](#) or [directives](#), the following parameters control the installation scope:

| Directive       | Values   | Description                                     |
|-----------------|--|---|
| InstallScope    | -1 None, 0 Current User, 1 All Users, 2 User Choice, 3 Automatic | Sets the scope mode                             |
| ScopeRequestUAC | 0 No, 1 Yes  | Request UAC elevation for All Users. Default: 1 |

| Directive        | Values      | Description                                     |
|------------------|-------------|---|
| ScopeFailOnAdmin | 0 No, 1 Yes | Fail if admin rights unavailable.<br>Default: 0 |

```
; Example: User Choice with UAC and silent fallback
[Settings]
InstallScope=2
ScopeRequestUAC=1
ScopeFailOnAdmin=0
```



### Tip

These parameters are essential for **build automation** scenarios where you control scope behavior without opening the GUI. See the full [Directives reference](#).

## 11.7 Best Practices



### User Choice

Best for general-purpose applications distributed online. Gives end users control without complexity.



### Automatic

Best for internal IT deployments or silent scripted installs where the calling process determines elevation.



### Current User

Ideal for lightweight apps, portable tools, or scenarios where admin rights are unavailable.



### All Users

Mandatory for services, drivers, or apps that must be visible to every account on the machine.

## 11.8 Related Pages

## Set Main Destination Folder [→](#)

Configure the default extraction path using scope-aware variables.

## Variables Reference [→](#)

Full list of runtime variables including scope-related ones.

## Resource Strings [→](#)

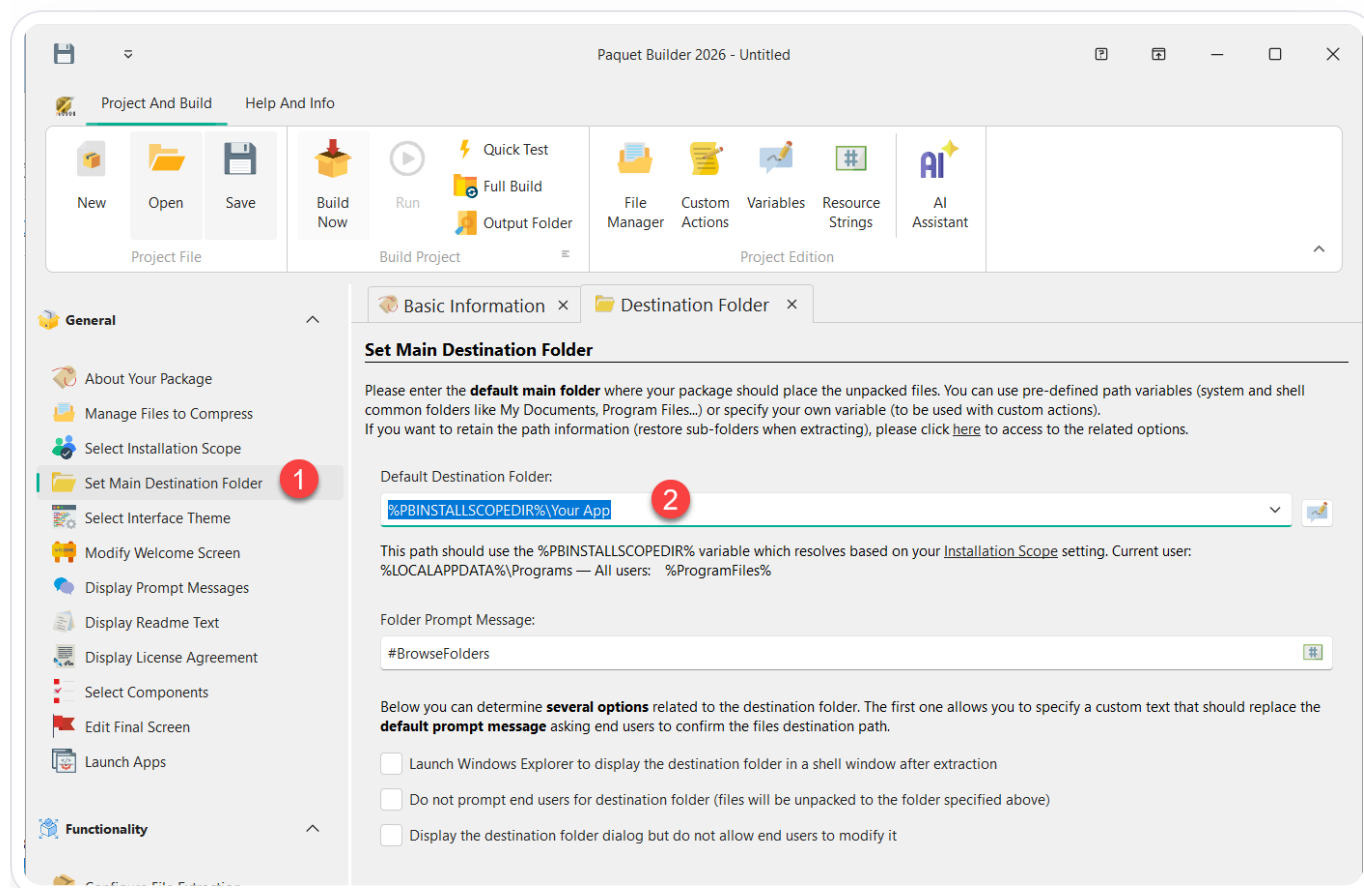
Translate or customize all dialog texts.

## Directives [→](#)

Automate scope configuration in CLI builds.

## 12. Set Main Destination Folder

This page defines **where** the package files are extracted on the end user's machine and **how much control** the user has over that location.



### 12.1 Default Destination Folder

Enter the default path where files should be extracted. If the user is prompted to choose a destination, this path appears pre-filled — most users will accept it as-is.

You can use **standard variables** to reference system and shell folders. For example:

```
%PBINSTALLSCOPEDIR%\My Application
```

#### Tip

Always use `%PBINSTALLSCOPEDIR%` instead of hardcoding paths — it automatically adapts to the **installation scope** (Current User or All Users).

## 12.1.1 Reading the Path from the Registry

You can also retrieve the destination from the Windows Registry using [custom actions](#):

- 1 Add a ["Read from Registry"](#) action to the **Initialization** event.
- 2 Set the destination variable to `%MYPATH%`.
- 3 Enter `%MYPATH%` in the "Default Destination Folder" field.

### Caution

Do not use the `%DESTPATH%` variable in this field — it is read-only. `%DESTPATH%` is set automatically to the final destination folder after extraction completes.

### Tip

To preserve sub-folder structure on the destination machine, enable "Save Path Information" in the [Compression Options](#) page or adjust the [component properties](#).

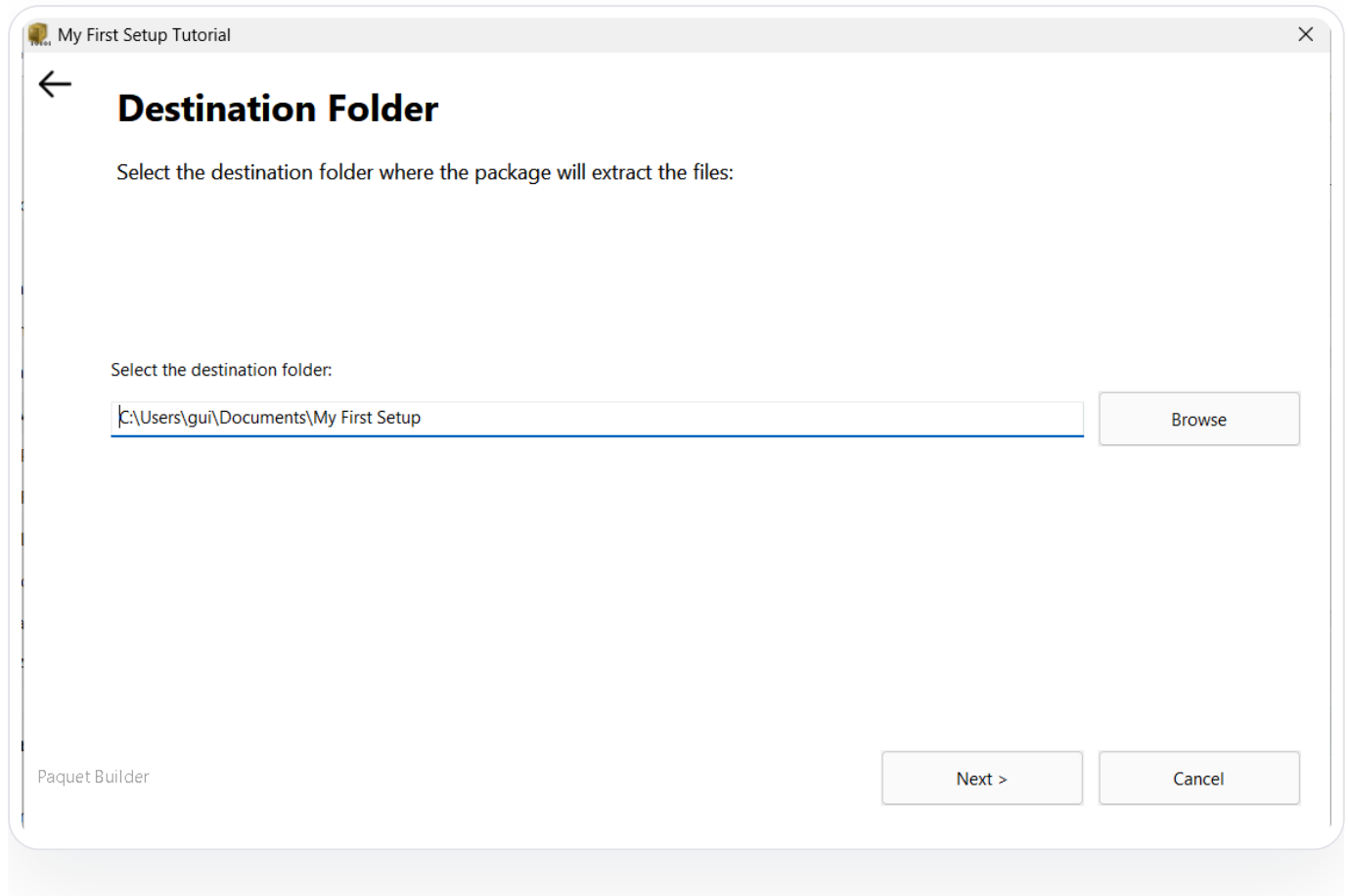
## 12.2 Folder Prompt Message

Enter a custom message displayed above the folder selection dialog. This field cannot be left blank.

Use a [resource string](#) for multilingual support — for example, `#BrowseFolders` is replaced at runtime with *"Please select the destination folder:"*.

## 12.3 Folder Dialog Options

The user sees the destination folder dialog and can change the path or browse to a different location. This is the default behavior.



The dialog is skipped entirely. Files are extracted to the default folder without asking the user.

Useful for **silent installations**, automated deployments, or packages where the destination must not be changed.

The dialog is displayed for informational purposes, but the path field is **grayed out** — the user can see the destination but cannot modify it.

Useful when you want to inform the user where files will go, without allowing changes.

## 12.4 Additional Option



### Launch Windows Explorer after extraction

Opens the destination folder in a Windows Explorer window after successful extraction, so the user can view and interact with the extracted files immediately.

## 12.5 Related Pages

### Installation Scope →

Control whether the package targets the current user or all users — affects destination paths.

### Variables →

Reference system folders and custom values in your destination path.

### File Destinations →

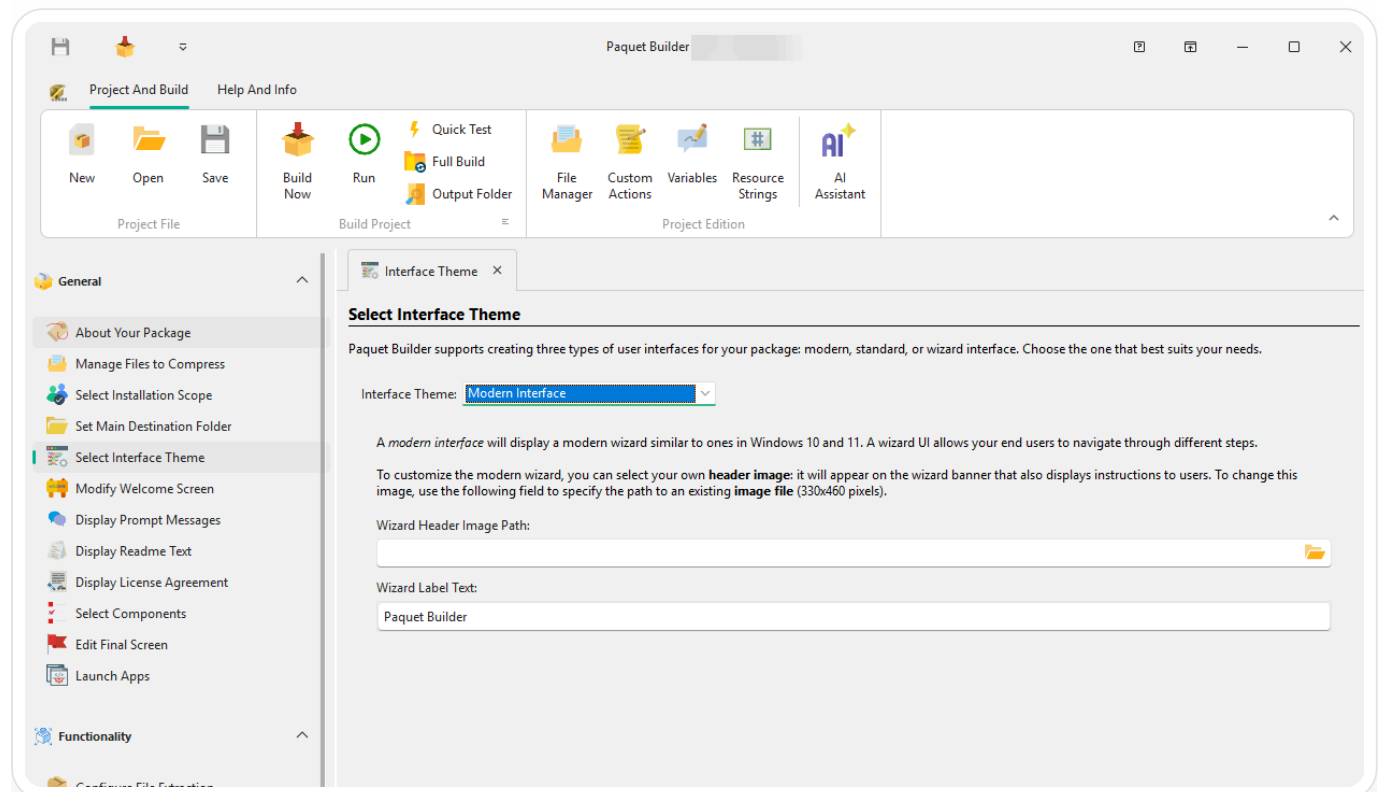
Assign individual destination paths to specific files within the package.

### Components →

Configure path preservation and folder linking per component.

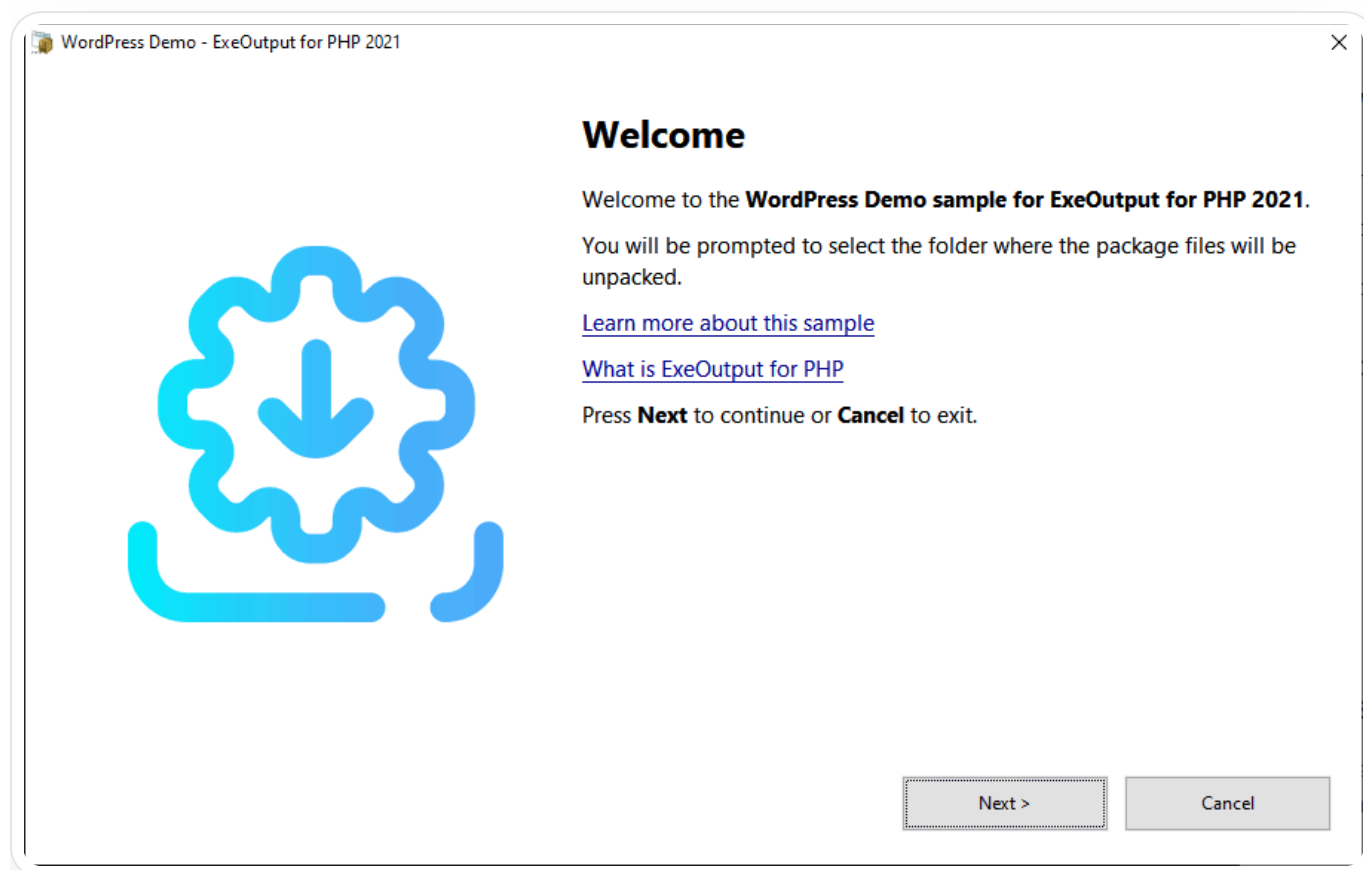
# 13. Select Interface Theme

Paquet Builder offers three interface themes that control the **look and feel of the installer dialogs** seen by end users. Each theme provides a different navigation style and level of visual customization.



## 13.1 Interface Themes

The **Modern** interface displays a wizard-style UI similar to Windows 10 and 11. It provides an intuitive, step-by-step navigation with a clean, contemporary look.



### Tip

This is the **recommended** theme for most packages.

## 13.1.1 Modern Options



### Wizard Header Image

Custom image displayed on the left side of pages such as the **Welcome Screen**. Leave blank for the default image.

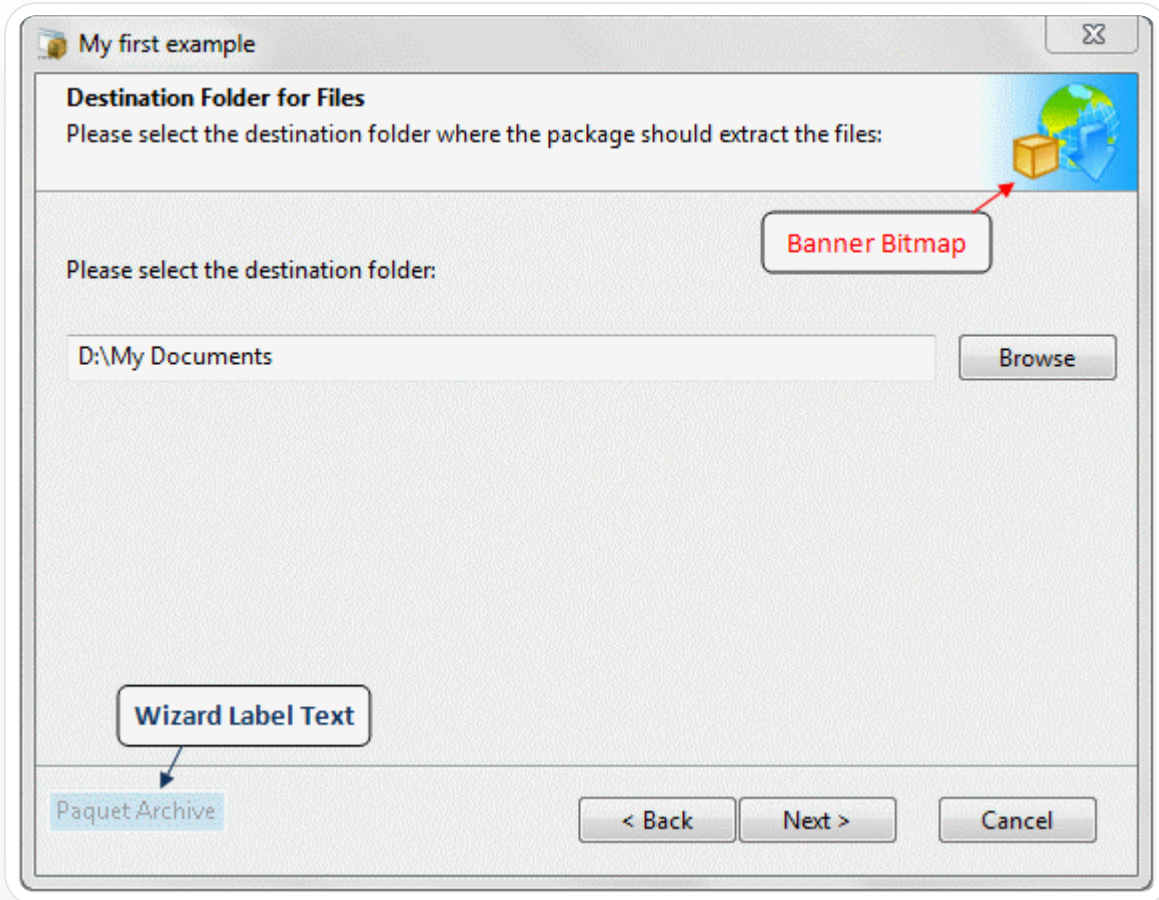
**Size:** 330 × 460 pixels — **Format:** PNG or BMP (PNG 8-bit recommended)



### Wizard Label Text

Text displayed near the navigation buttons — typically your company name or website domain.

The **Wizard** interface is a classic style with **Next / Back** navigation buttons, similar to wizards found in Windows 8, 7, and XP. Efficient and familiar for end users.



### 13.1.2 Wizard Options



#### Header Image

Shown in the white banner where user instructions appear. Leave blank for the default, or enable **“Generate a header image automatically from the package icon”** to extract the best icon size (48×48 or 32×32).

**Size:** 90 × 58 pixels — **Format:** PNG or BMP (PNG 8-bit recommended, white background)



#### Watermark Image

Custom image for the left side of the wizard, similar to classic Windows setup wizards.

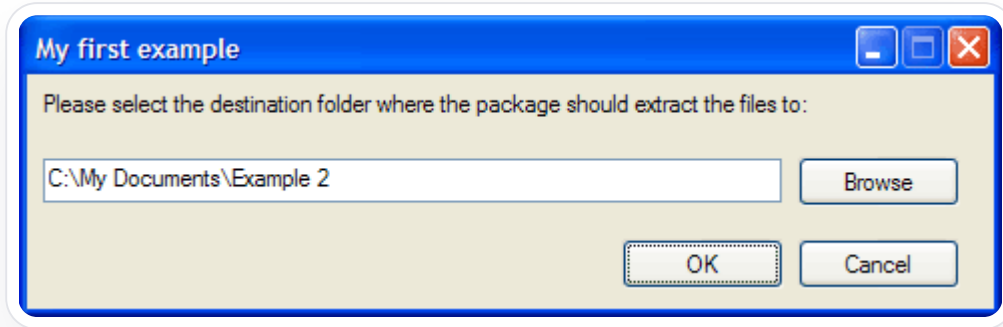
**Size:** 175 × 345 pixels — **Format:** PNG or BMP (PNG 8-bit recommended)



#### Wizard Label Text

Text near the navigation buttons — company name or website domain.

The **Standard** (traditional) interface presents a sequence of simple dialog boxes. Users move forward by clicking **OK** but cannot return to a previous dialog.



Less flexible than Modern or Wizard, but sufficient for simple packages where a full wizard is unnecessary.

### **Note**

Standard packages have no additional theme settings.

## 13.2 Theme Comparison

| Feature                    | Modern  | Wizard     | Standard     |
|----------------------------|---------|------------|--------------|
| Back / Next navigation     | Yes     | Yes        | No (OK only) |
| Custom header image        | Yes     | Yes        | —            |
| Watermark image            | —       | Yes        | —            |
| Label text                 | Yes     | Yes        | —            |
| Auto-generated icon header | —       | Yes        | —            |
| Windows look & feel        | 10 / 11 | XP / 7 / 8 | Classic      |

### **Caution**

Including custom images increases the size of the final package, although they are compressed by Paquet Builder.

## 13.3 Related Pages

## Welcome Screen →

Configure the first dialog seen by end users.

## Package Resources →

Set the package icon and version information embedded in the EXE.

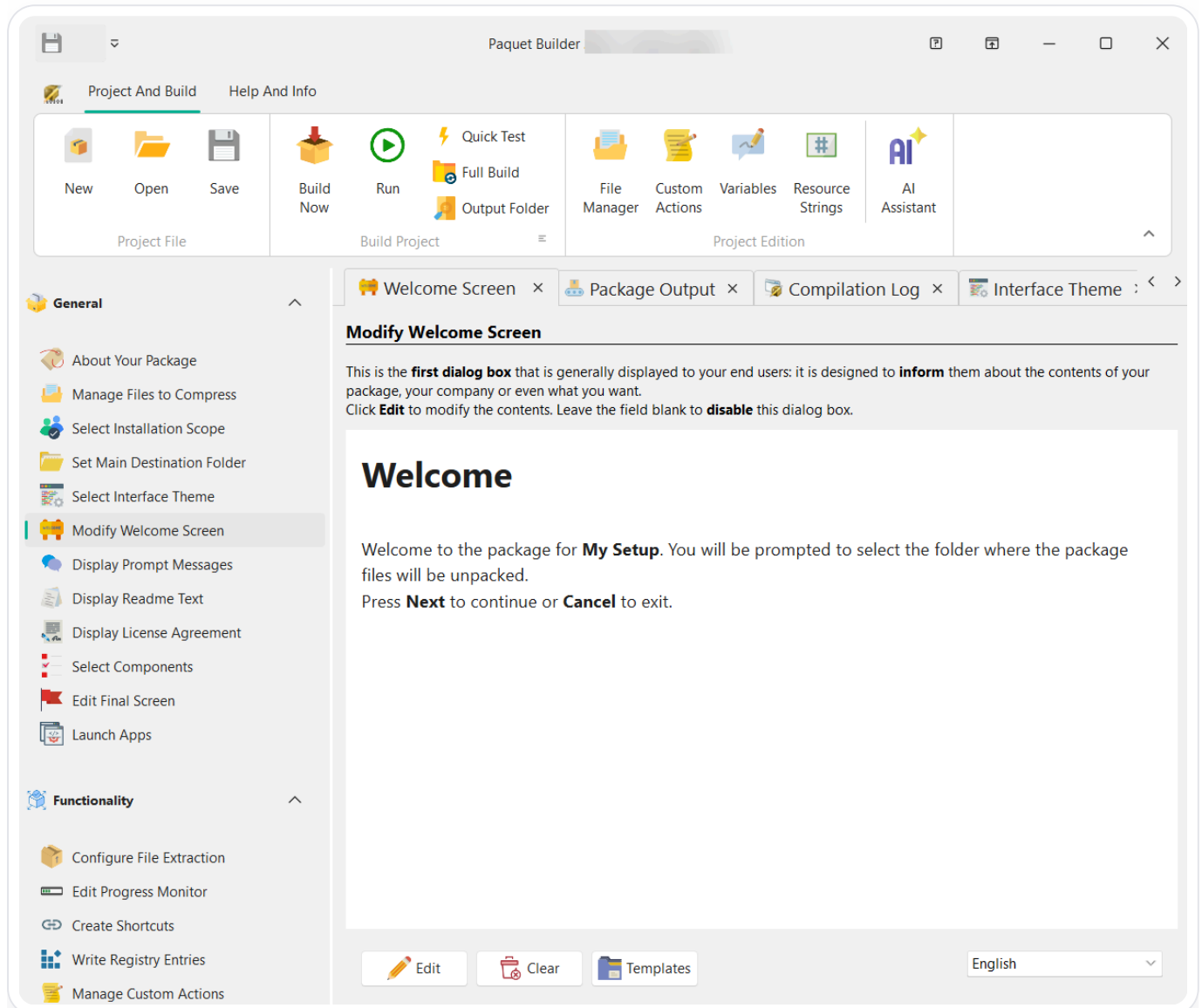
## Resource Strings →

Translate or customize all dialog texts and labels.

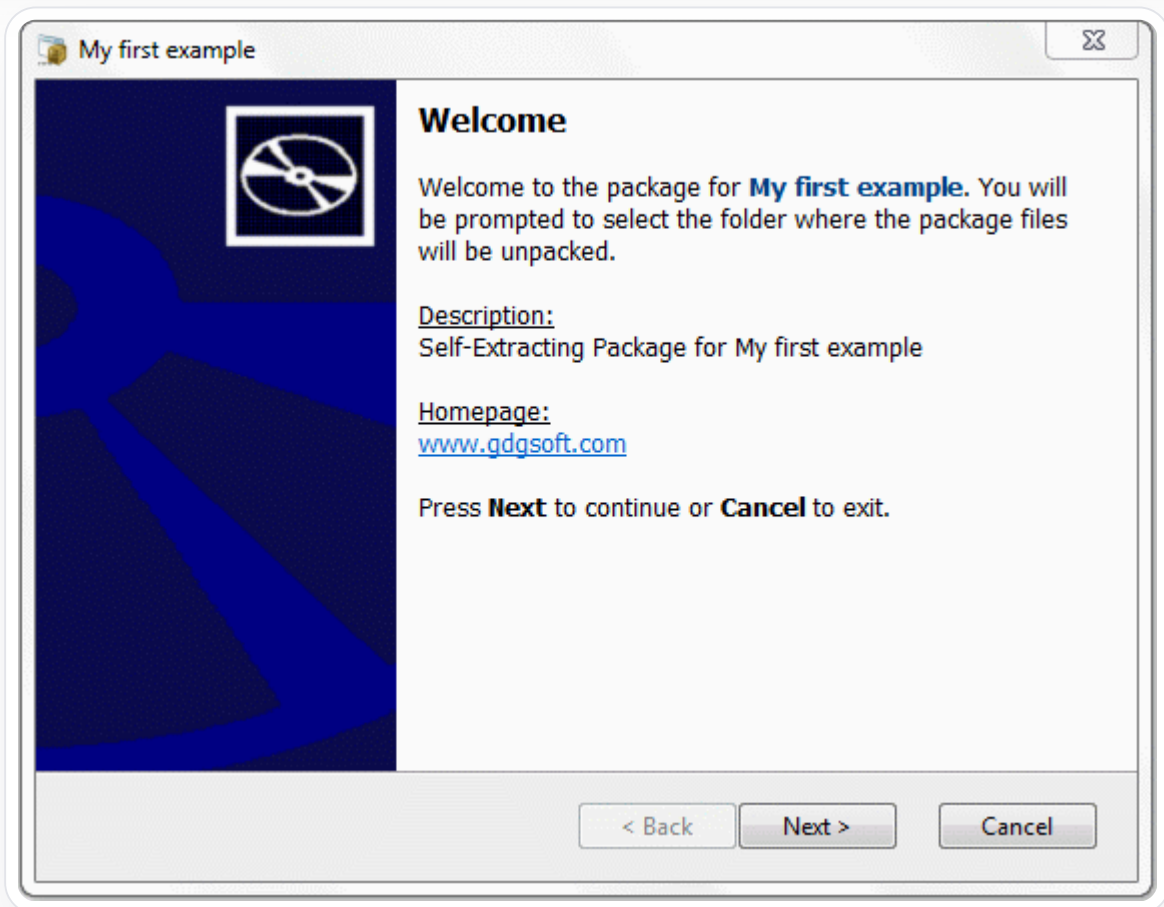
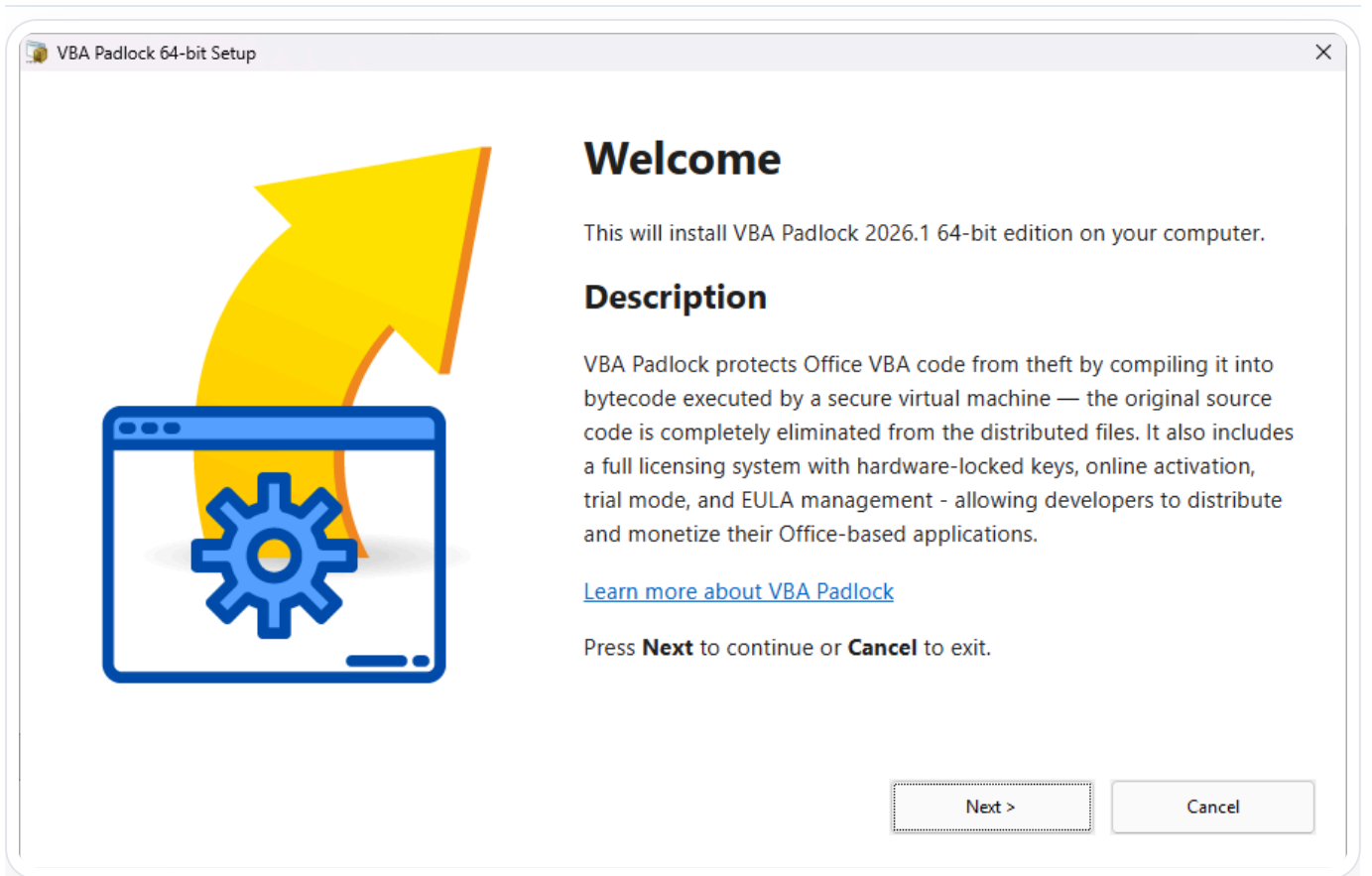
# 14. Modify Welcome Screen

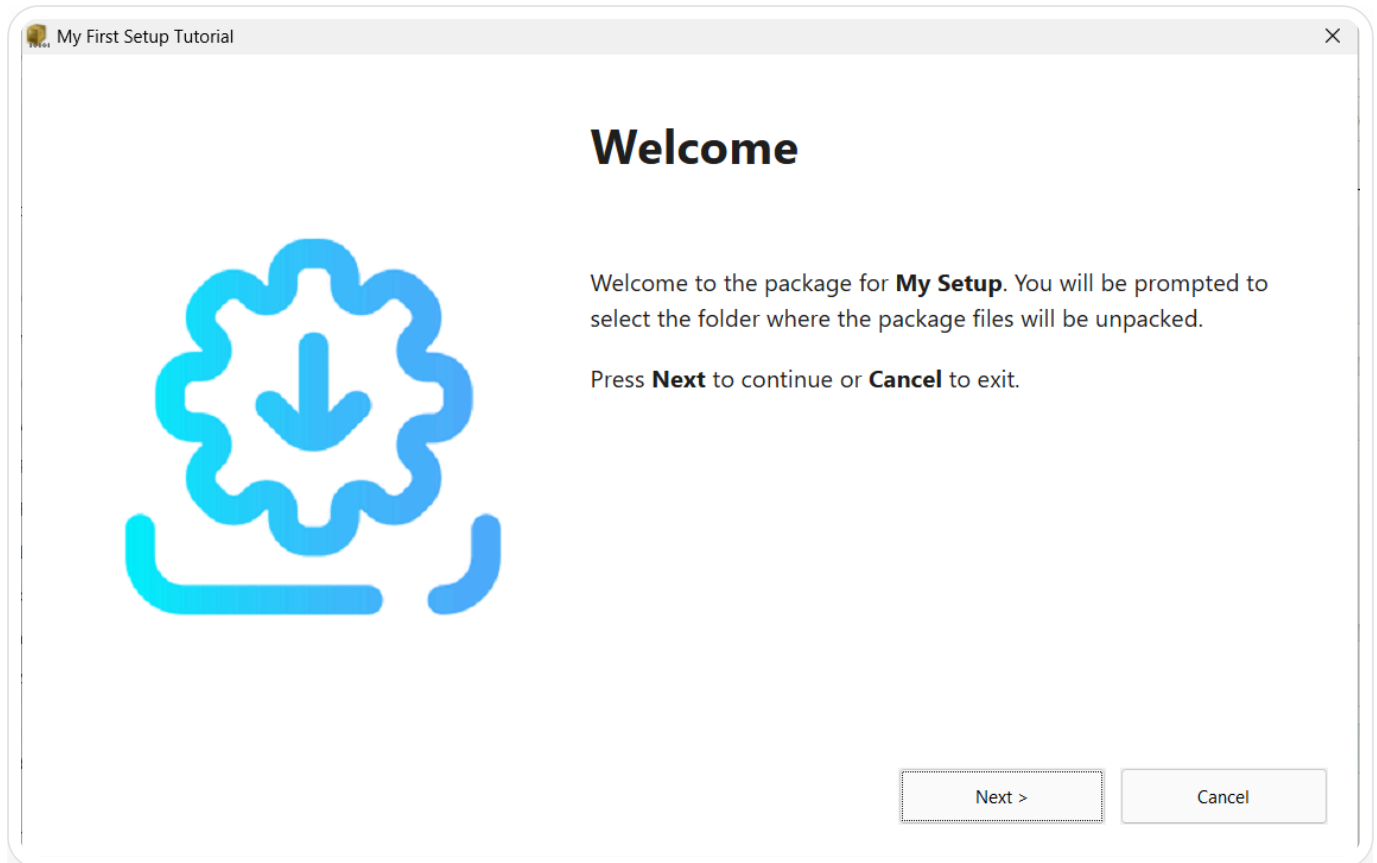
The Welcome screen is the **first dialog box** end users see when they run your package. It describes the package contents and provides instructions before extraction begins.

The appearance of the dialog depends on the **interface theme** you selected.



## 14.1 Welcome Screen Previews





## 14.2 Editing the Dialog Text

### Edit

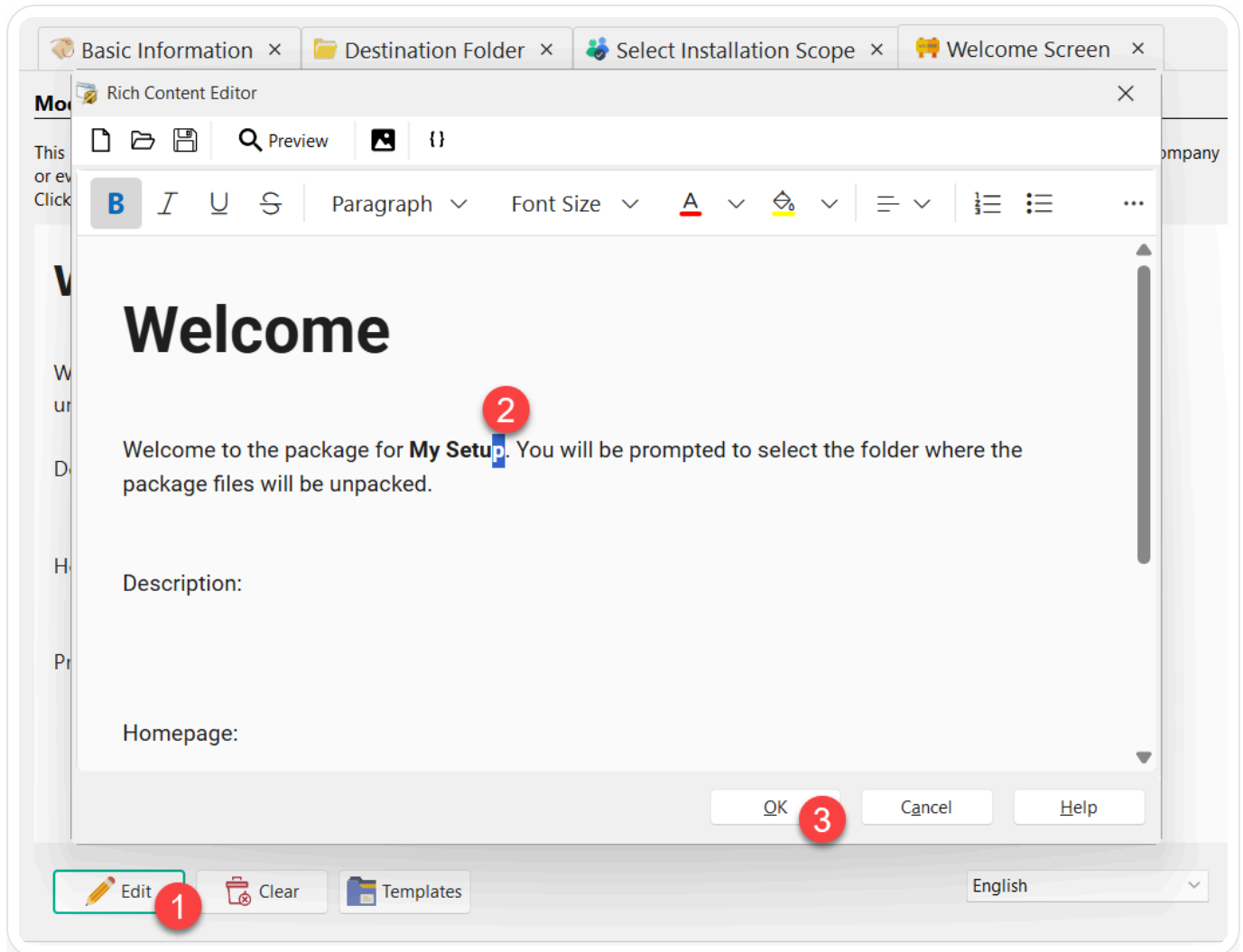
Opens the [Rich Content Editor](#) to write and format the welcome message with rich HTML content.

### Clear

Removes the current text entirely. If the welcome text is blank, the Welcome screen is disabled.

### Templates

Select a pre-made welcome template from the menu list. Templates provide a ready-to-use starting point.



### 14.3 Variables and Resource Strings

You can include **resource strings** and **variables** in the welcome text — they are automatically replaced at runtime. For example, `%APTITLE%` is replaced with the package title.

### 14.4 Templates

Templates are HTML files stored in the `Templates` subfolder and listed in `pbtempla.ini`. The following placeholders are supported and replaced with their corresponding values:

| Placeholder    | Replaced with         |
|----------------|-----------------------|
| [ARCHIVETITLE] | Package title         |
| [COMPANYNAME]  | Author / Company name |

| Placeholder    | Replaced with                  |
|----------------|--------------------------------|
| [COMPANYEMAIL] | Contact email address          |
| [HOMEPAGE]     | Company web homepage           |
| [COPYRIGHT]    | Legal copyright text           |
| [DESCRIPTION]  | Package description            |
| [PRODNUM]      | Product version number         |
| [FILENUM]      | Number of files in the package |

## 14.5 Multilanguage Packages

Paquet Builder supports **multilanguage packages**, allowing the Welcome dialog to display in the end user's language.

- 1 Open the language list



to select a language.

- 2 Edit the Welcome text for that language using the Rich Content Editor.
- 3 Repeat for each compiled language.

### **i** Fallback behavior

- If the Welcome text is missing for the user's language, the **default language** text is used.
- If the default language's Welcome text is blank, the Welcome screen is **disabled entirely** — even if texts for other languages exist.

### **Tip**

Always select the **default language** in the language list before writing Welcome texts.

## 14.6 Related Pages

## Rich Content Editor [→](#)

Learn how to format text with the built-in HTML editor.

## Interface Theme [→](#)

Choose the visual style of the installer dialogs.

## Language Manager [→](#)

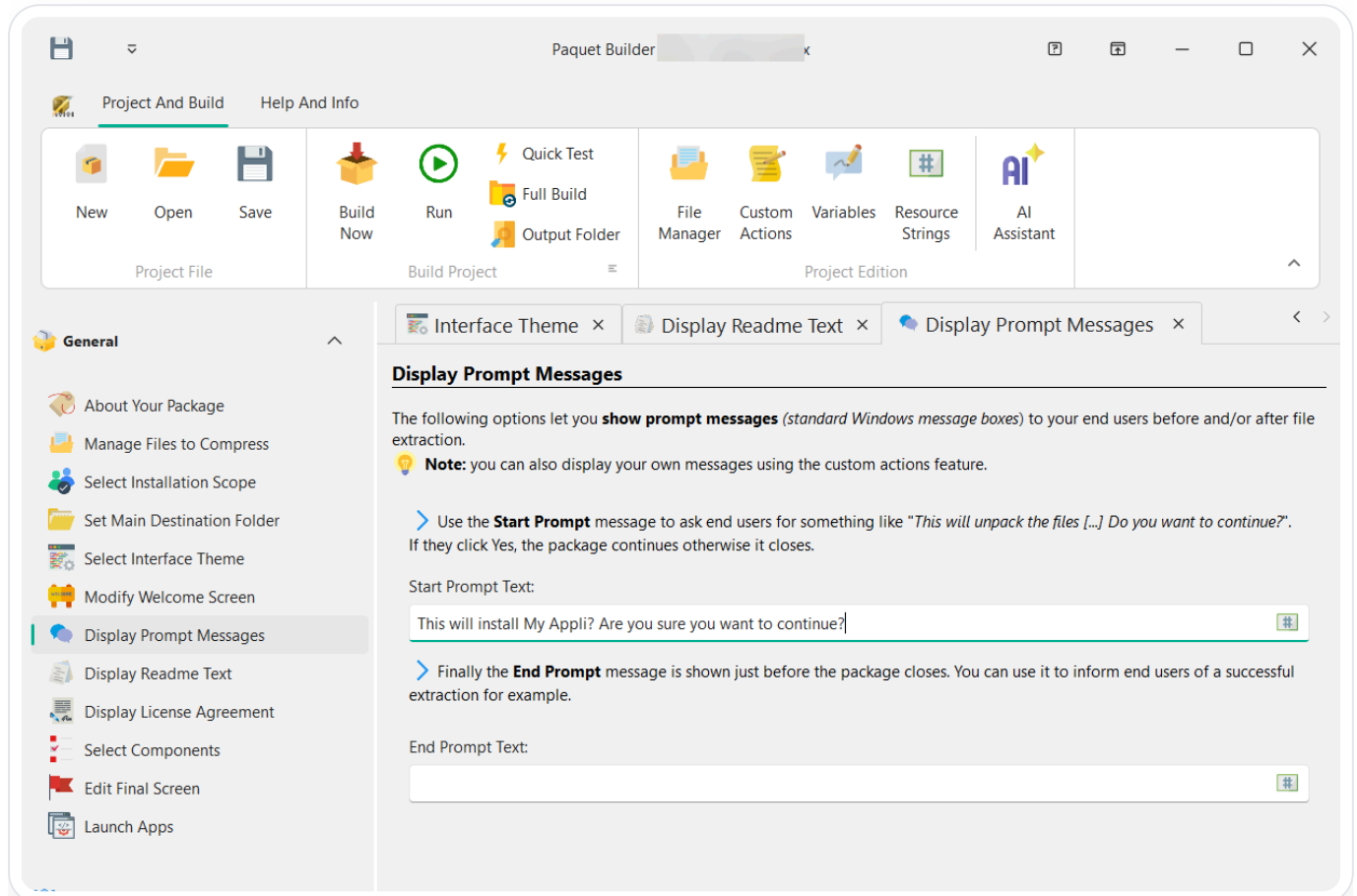
Add multilanguage support to your package.

## Resource Strings [→](#)

Customize and translate dialog labels and messages.

# 15. Display Prompt Messages

Paquet Builder can display standard Windows **message boxes** to your end users at two key moments: before extraction begins and after it completes. These are simple Yes/No or informational prompts — leave the fields blank to disable them.

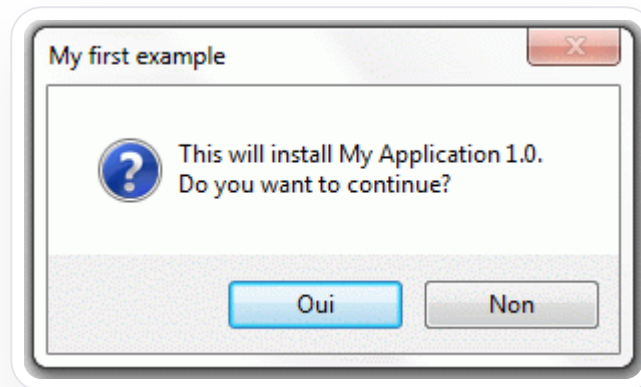


## 15.1 Prompt Messages

Shown **immediately** after the user launches your package, before any extraction takes place. The user must click **Yes** to proceed or **No** to cancel.

Typical example:

```
This will install My Application 1.0./$Do you want to continue?
```



This is useful to give users a clear confirmation step before files are extracted to their machine.

Shown **just before the package closes**, after extraction completes successfully. This is a simple informational message — the user clicks **OK** to dismiss.

Typical example:

```
Extraction successfully completed!
```

Use this to confirm that everything went well, or to provide brief follow-up instructions.

## 15.2 Formatting and Variables



### Line breaks

Use `/s` to insert a line break in the message text. For example: `Line 1/$Line 2`.



### Resource strings

Click the button next to each text box to insert a **resource string** for multilanguage support.



## Variables

Include **variables** like `%APPTITLE%` — they are replaced at runtime with their actual values.



## Custom messages

Need more than two prompts, or conditional logic? Use **custom actions** to display additional message boxes at any point during execution.

### Note

If you leave a prompt field **blank**, the corresponding message box is not displayed.

## 15.3 Related Pages

### Custom Actions

Display additional message boxes with conditions and custom logic.

### Resource Strings

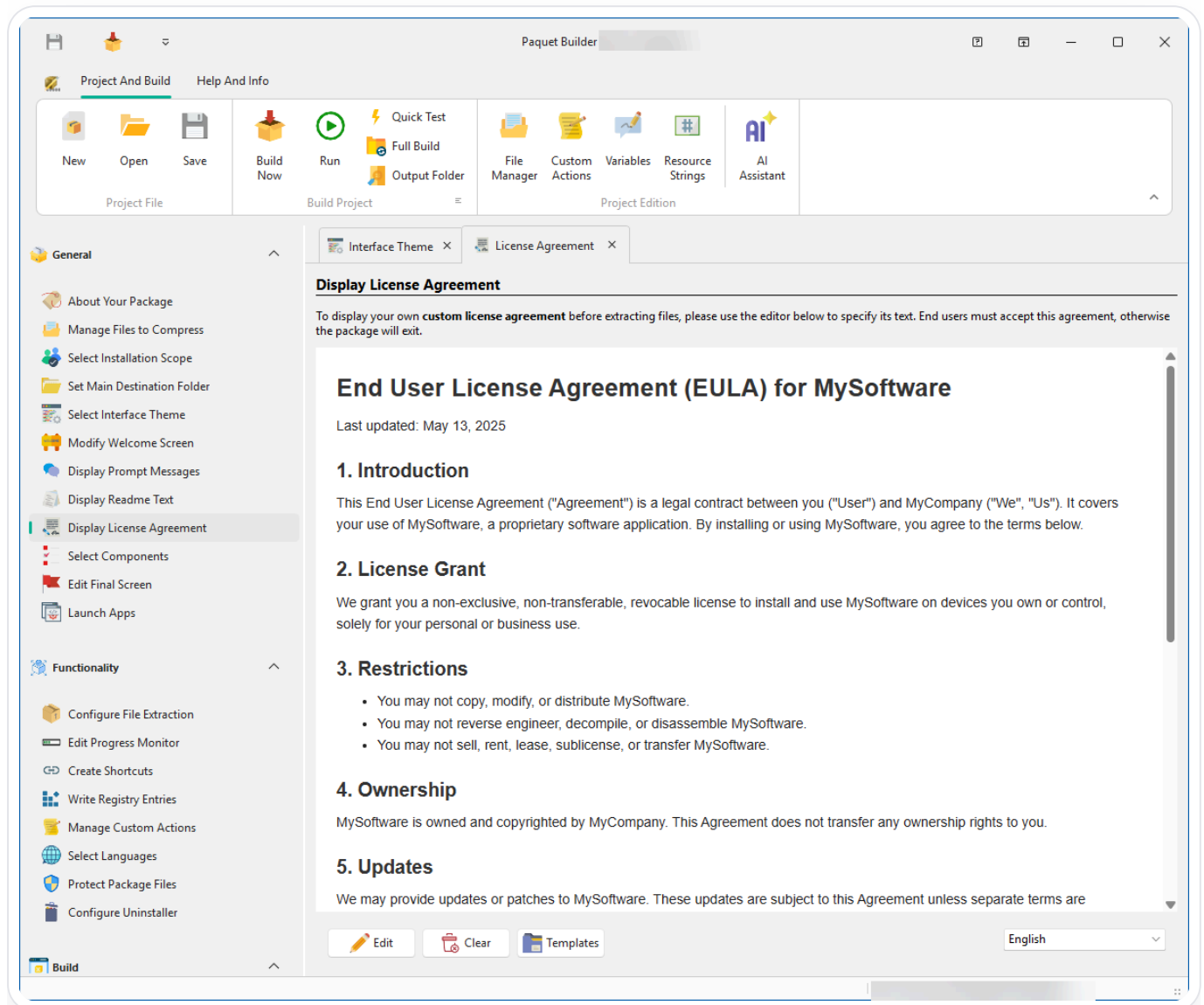
Translate prompt texts for multilanguage packages.

### Variables

Use runtime variables in your message texts.

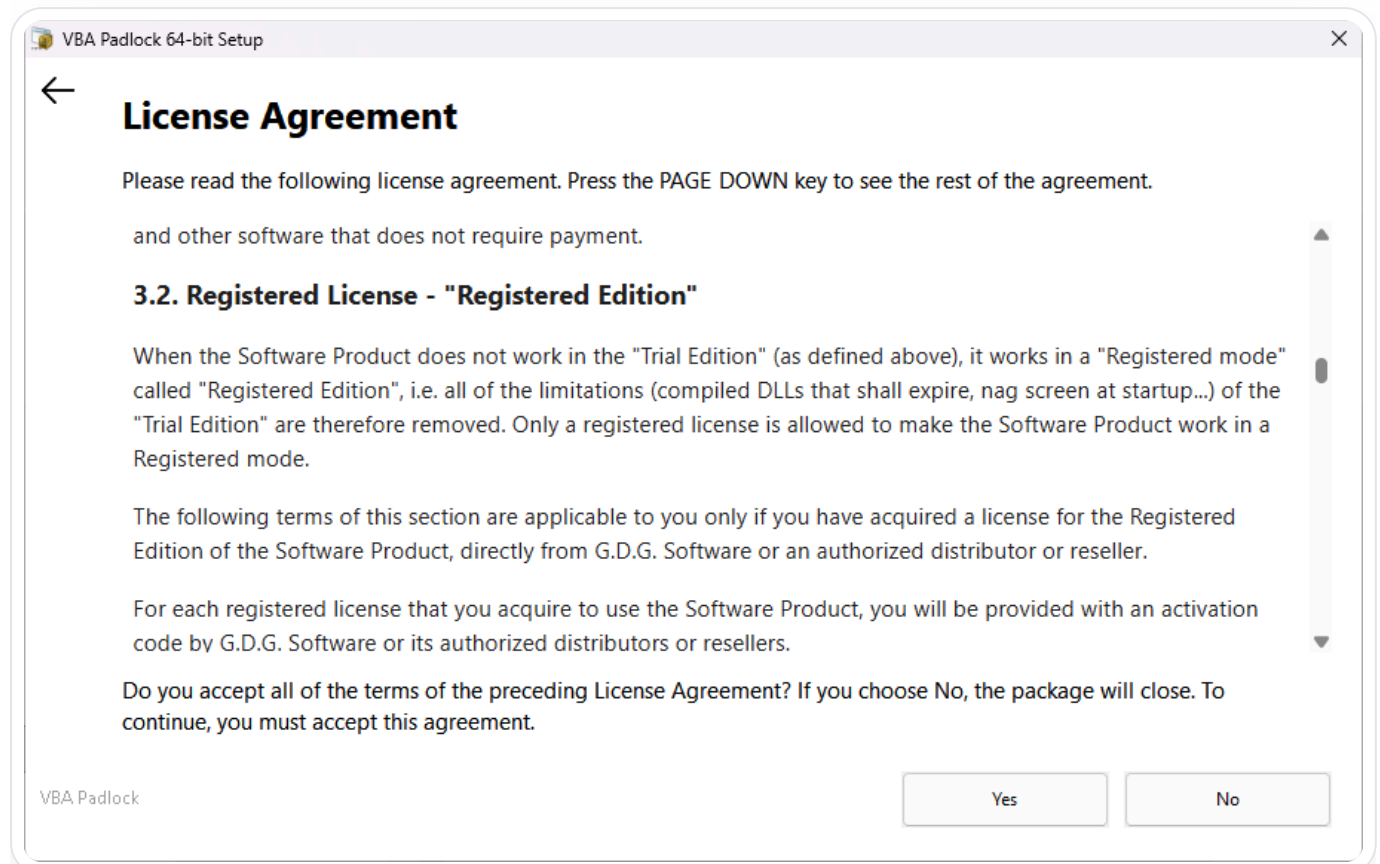
# 16. Readme and License Agreement

Paquet Builder lets you include two optional text screens in the installer flow: a **License Agreement** that end users must accept to proceed, and a **Readme** with important information or instructions.



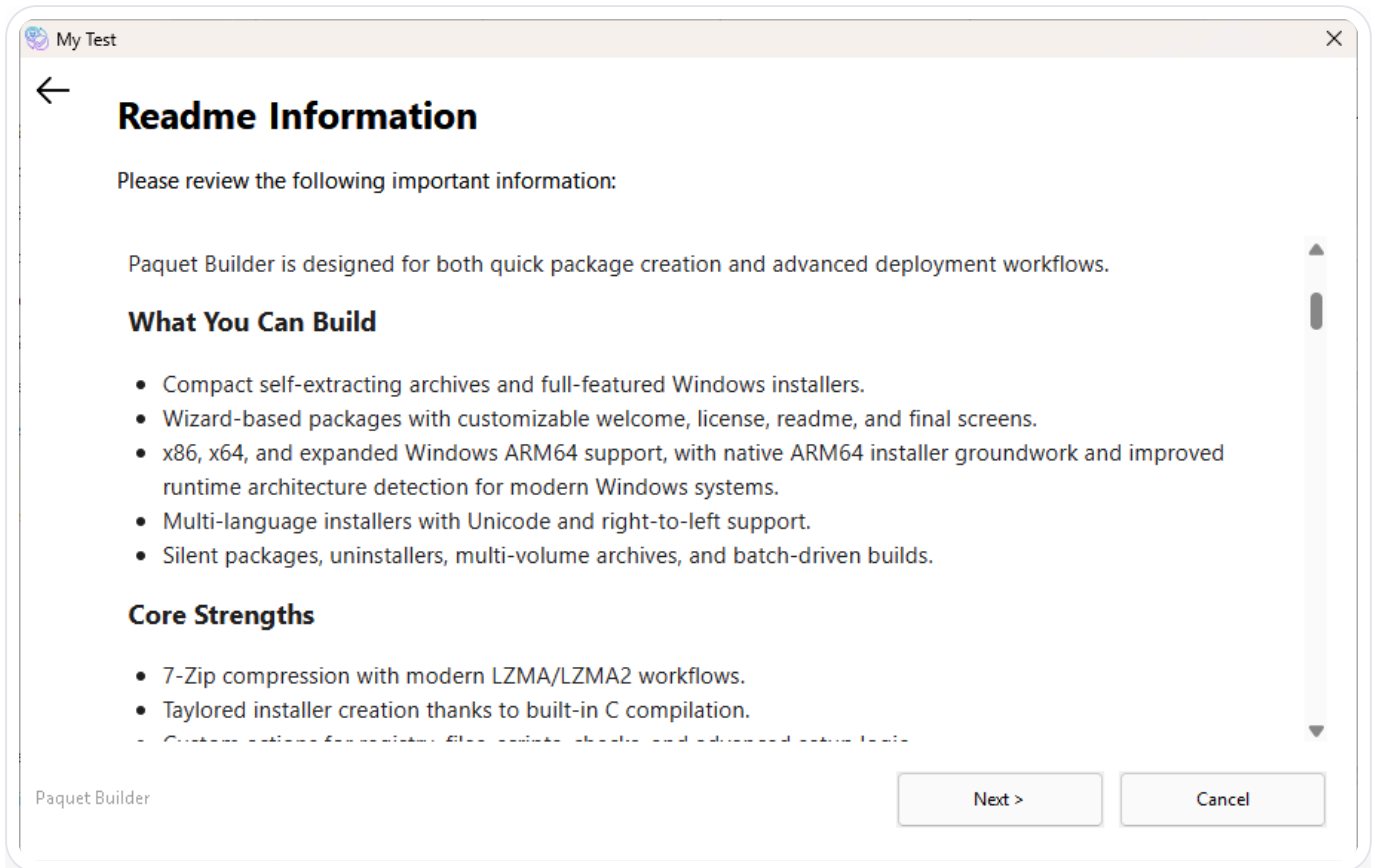
## 16.1 How Each Screen Works

Displayed **before extraction begins**. End users must click **Yes** (or **I Agree**) to continue — otherwise the package exits.



Use this for EULA, terms of service, or any legal text that requires explicit user consent.

Displayed as an **informational screen** — the user reads it and clicks **Next** to continue. No acceptance is required.



Use this for release notes, installation prerequisites, known issues, or post-install instructions.

## 16.2 Editing the Text

### Edit

Opens the [Rich Content Editor](#) to write and format the text with full HTML support.

### Clear

Removes the text entirely. If blank, the corresponding screen is **disabled** and not shown to end users.

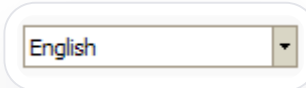
### Templates

Select a pre-made template from the menu to use as a starting point.

## 16.3 Multilanguage Support

For [multilanguage packages](#), you can write a localized version of each text for every compiled language.

1 Open the language list



to select a language.

2 Edit the Readme or License Agreement text for that language.

3 Repeat for each compiled language.

### Fallback behavior

- If a text is missing for the user's language, the **default language** text is used.
- If the default language text is blank, the screen is **disabled entirely** — even if other languages have content.

### Tip

Always select the **default language** in the language list before writing texts — it determines which screens are displayed.

## 16.4 Related Pages

### Rich Content Editor

Format text with the built-in HTML editor.

### Language Manager

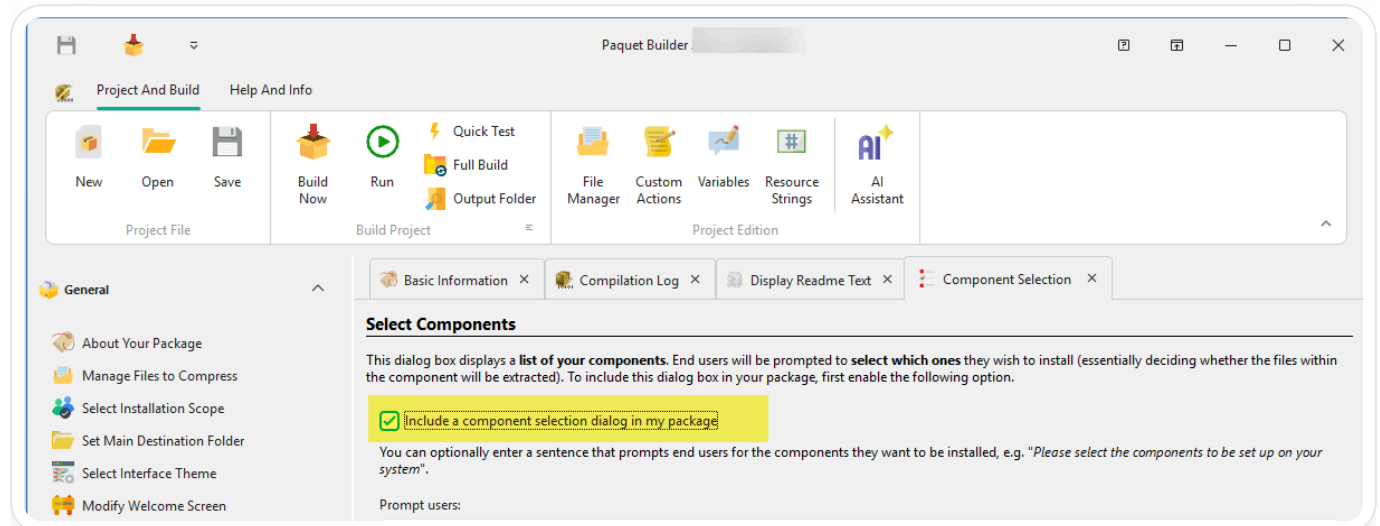
Add multilanguage support to your package.

### Resource Strings

Use translatable string variables in your texts.

# 17. Select Components

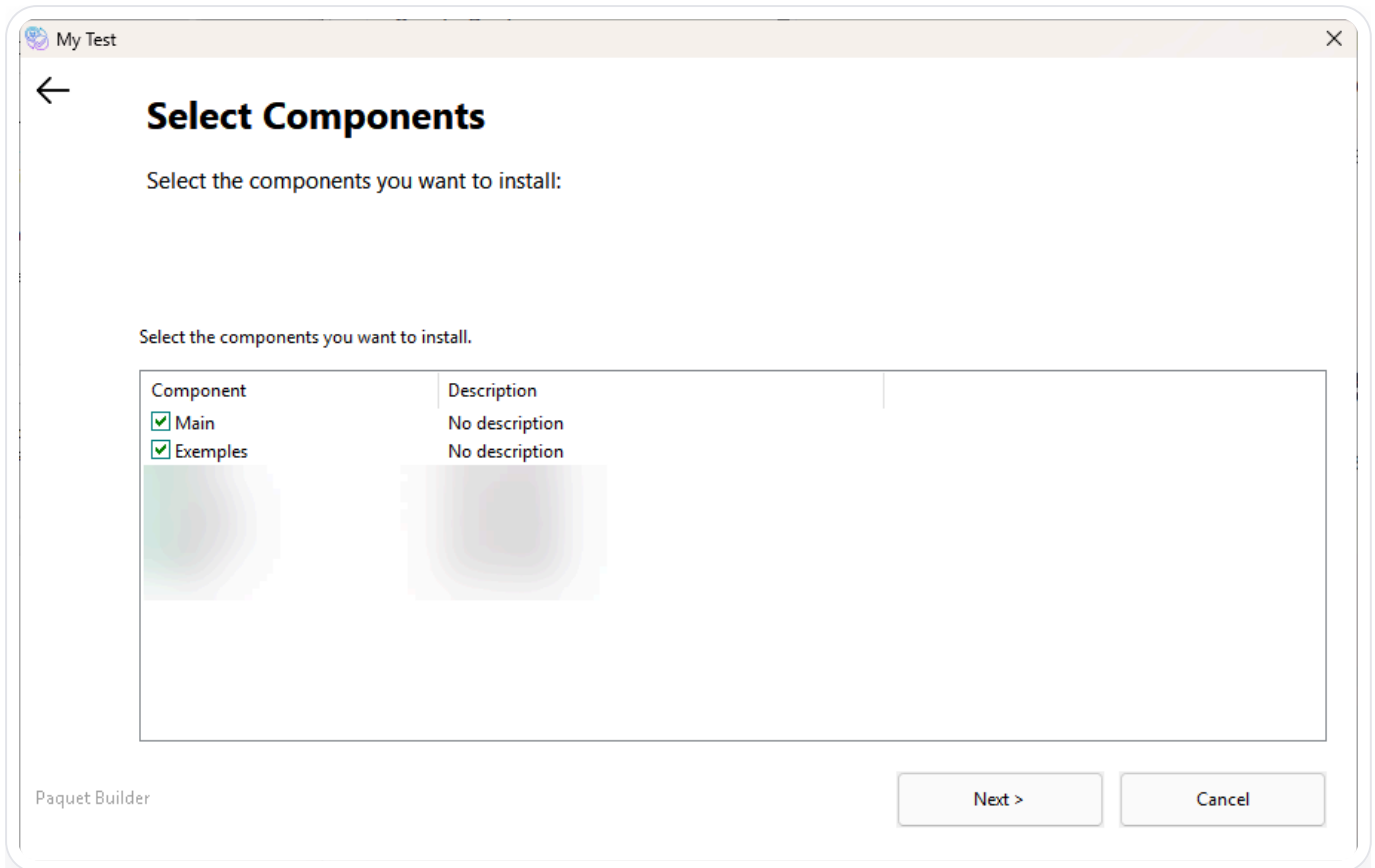
When your package contains multiple **components**, you can display a **component selection dialog** that lets end users choose which ones to install. Each component appears with a checkbox, a title, and an optional description.



To enable this dialog, activate the **“Include a component selection dialog in my package”** checkbox.

## 17.1 Runtime Preview

In the installer, end users see a list of components they can toggle on or off:



## 17.2 How It Works



### User selection

End users read each component's description and toggle its checkbox. Enabled components have their files extracted; disabled ones are skipped.



### Component variables

Each component has an **associated variable**. When enabled, its value is **1** (true); when disabled, **0** (false). You can also modify these variables programmatically with **custom actions**.



### Title and description

Customize the **Component Title** and **Component Description** in the **Component Properties** dialog — double-click a component in the list to open it.



### Prompt text

Customize the instructional text displayed above the component list using the **"Prompt users"** field. **Resource strings** are supported (e.g. `#DefSe1CompPrompt`).

## 17.3 Adjusting Component Order

By default, components appear in the order they were created, with **Main** always first. You can rearrange them with drag-and-drop:

- 1 Select a component and hold the left mouse button.
- 2 Drag it to the desired position and release.
- 3 Repeat as needed.

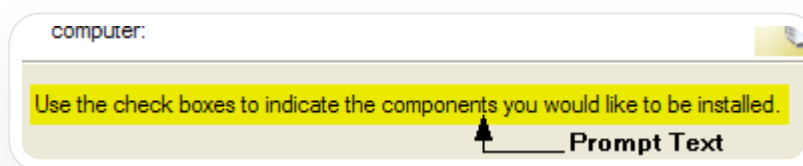
### Note

The display order in this dialog is independent of the order in the File Manager's component list.

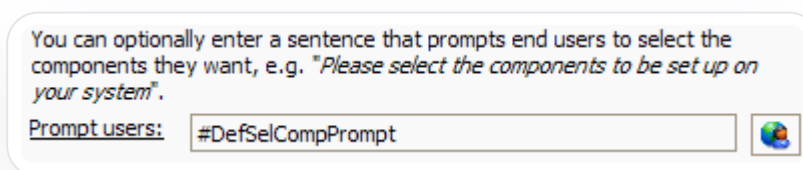
## 17.4 Customizing the Prompt Text

The "**Prompt users**" field controls the instructional text shown above the component list. You can use a [resource string](#) for multilanguage support.

In the package at runtime:



In Paquet Builder:



### Caution

This dialog only lets end users **select** components. To manage, rename, or edit components, use the [File Manager](#).

## 17.5 Related Pages

### Components

Learn how components work and configure their properties.

## Auto-Create Components →

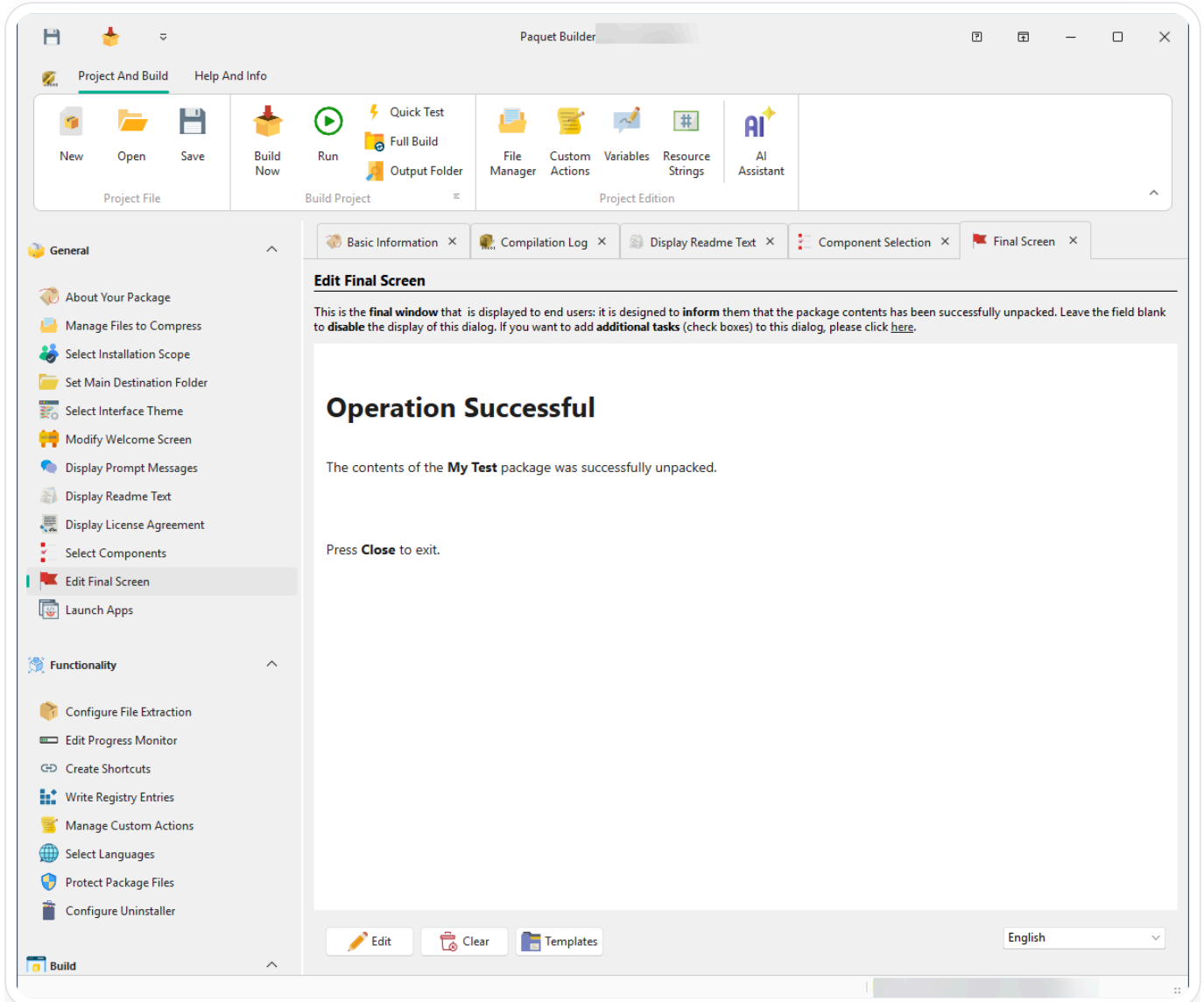
Generate components automatically from folder structures.

## File Manager →

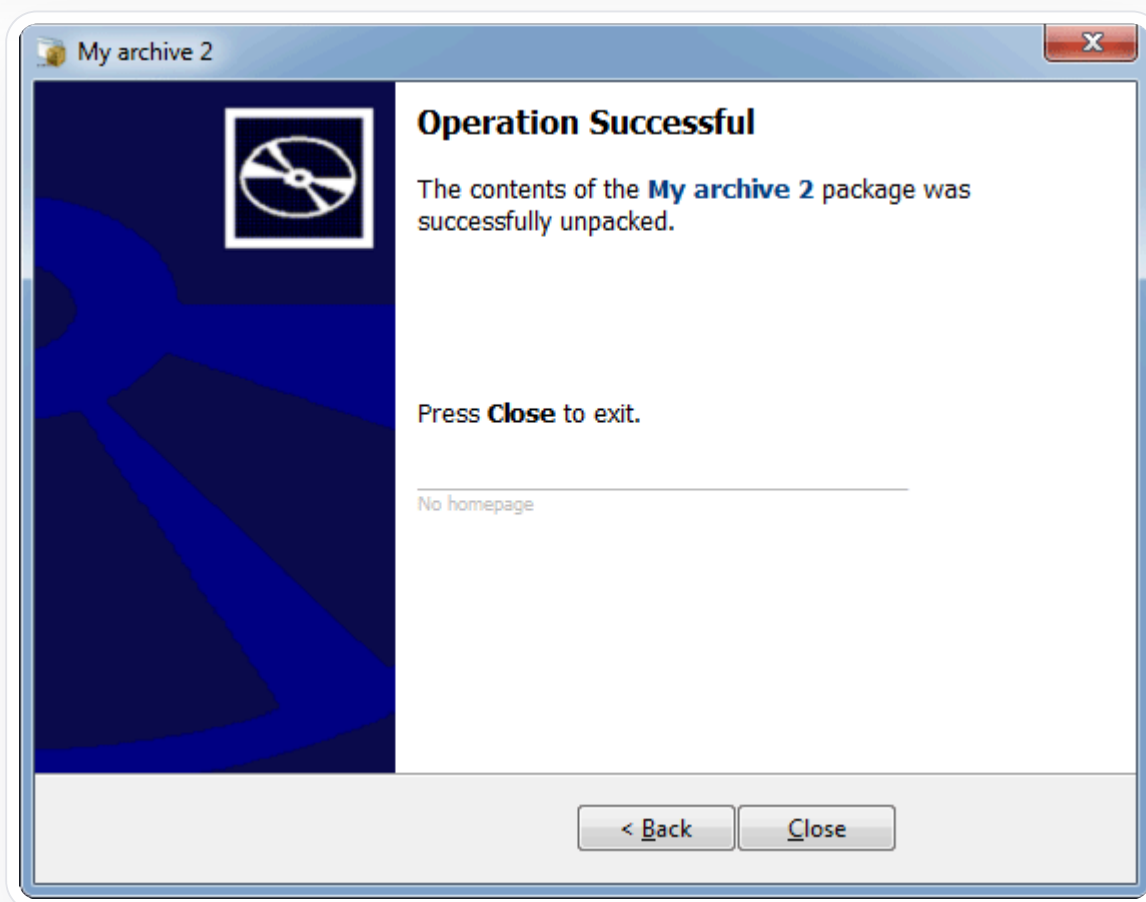
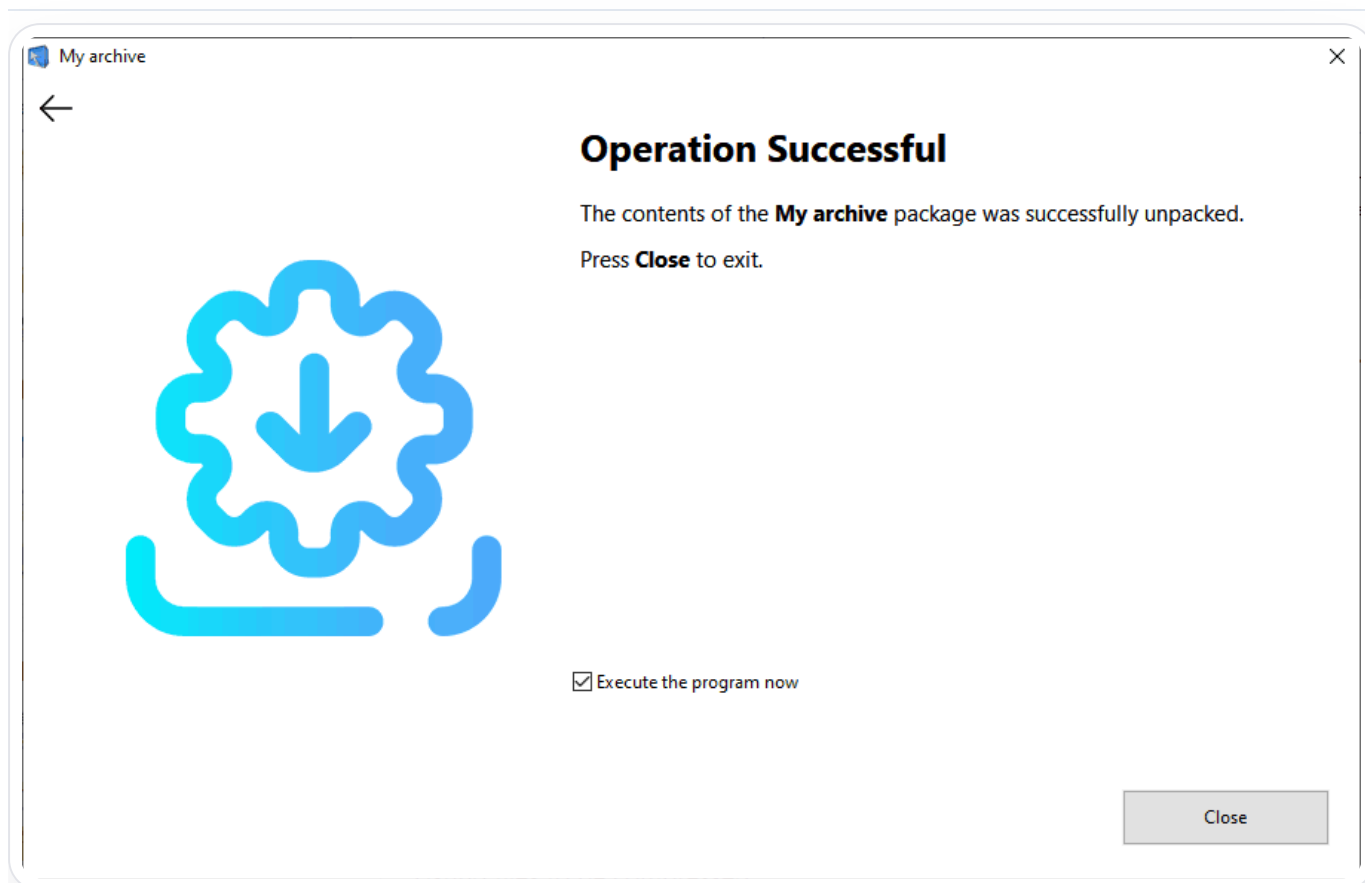
Manage files and components in your package.

# 18. Edit Final Screen

The Final Screen is the **last dialog** shown to end users after a successful extraction. It typically confirms that files were unpacked and may offer optional tasks such as launching the application or opening a readme file.



## 18.1 Runtime Previews



## Note

The Final Screen is optional. If the text is left blank, it is not displayed.

## 18.2 Editing the Screen Text

### Edit

Opens the [Rich Content Editor](#) to write and format the completion message.

### Clear

Removes the text. When blank, the Final Screen is disabled.

### Templates

Select a pre-made template from the menu as a starting point.

## 18.3 Checkboxes and Additional Tasks

You can add **checkboxes** to the Final Screen, allowing end users to choose optional post-extraction tasks — for example, *“Launch the application”* or *“Open the readme file.”*

Configure these tasks on the [Program, Setup Execution](#) page. For more complex logic, use [custom actions](#) to prompt users with choices.

End users can press **Back** to return to a previous screen or **Close** to exit.

## Detecting extraction errors

Use [custom actions](#) to evaluate the `%TOTALEXTFILE%` variable. If its value is `0`, an error occurred during extraction.

## 18.4 Multilanguage Support

For [multilanguage packages](#), write a localized Final Screen text for each compiled language.

- 1 Open the language list



to select a language.

- 2 Edit the Final Screen text for that language.
- 3 Repeat for each compiled language.

### Fallback behavior

- If the Final Screen text is missing for the user's language, the **default language** text is used.
- If the default language text is blank, the screen is **disabled entirely** — even if other languages have content.

### Tip

Always select the **default language** in the language list before writing Final Screen texts.

## 18.5 Related Pages

### Program, Setup Execution [→](#)

Configure post-extraction tasks and Final Screen checkboxes.

### Rich Content Editor [→](#)

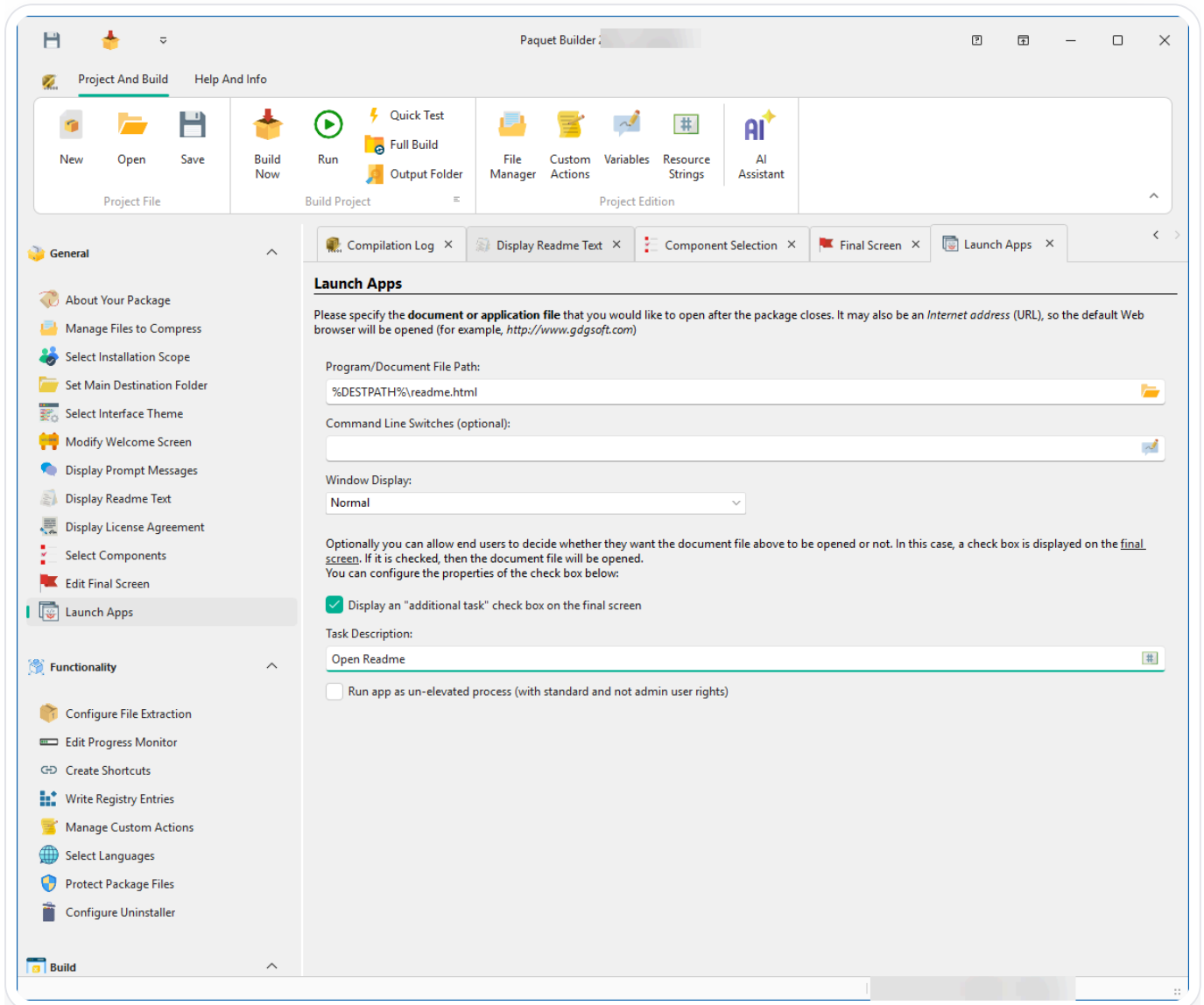
Format text with the built-in HTML editor.

### Language Manager [→](#)

Add multilanguage support to your package.

# 19. Program, Setup Execution

This page configures what happens **after files are extracted** — launch an application, open a document, or run an existing setup routine. The available options depend on the **package type**.



## Note

Some options on this page vary depending on the package type (Standard Package vs. Setup Routine).

## 19.1 Post-Extraction Action

Specify a **document or executable** to open after the package closes (optional).

Enter the full path in the **Program/Document Filename** field. Examples:

| Value                     | Result  |
|---------------------------|---|
| %DESTPATH%\Readme.txt     | Opens the extracted readme in the default text editor |
| %DESTPATH%\MyApp.exe      | Launches the extracted application                    |
| https://www.mycompany.com | Opens the URL in the default browser                  |

### Tip

Need to launch multiple programs? Use **custom actions** to execute additional files at any point during the installation flow.

Specify the **setup launcher** — the executable that runs your existing installation program.

The package extracts files to a temporary folder, launches the setup, waits for it to finish, then removes the temporary files.

### Caution

Always specify the **full path** (e.g. %DESTPATH%\MySetup.exe). Use the



button to select the file from the list.

To run a **Microsoft Windows Installer** package (MSI) or patch (MSP), enable the “Install a Windows Installer package” option. See [installing MSI packages](#).

## 19.2 Additional Options

### Final Screen checkbox

*Standard Package only.* Add a checkbox to the **Final Screen** letting the user decide whether to launch the specified file. Set the caption with the “Task Description” field. For more complex logic, use **custom actions**.



## Run as un-elevated

*Standard Package only.* If the package runs with **admin rights**, the launched program inherits those rights. Enable this option to run it with **standard user privileges** instead.



## Command-line switches

Pass parameters to the executed file. **Variables** referencing system or shell folders are supported.



## Fixed temporary folder

*Setup Routine only.* Some setups (e.g. MSI) require files to stay in the same location. Enter a unique name (e.g. `MyCompanySetupFiles`) to always use the same temporary folder.

### 19.2.1 Remove Setup Files at Windows Startup

*Setup Routine only.* Enable this if the setup routine requires a **reboot** (e.g. for replacing files in use):

- Files are extracted to a temporary folder and the setup is launched.
- Temporary files remain until the next Windows restart.
- An uninstaller silently removes the temporary files and folders at startup — this does not affect installed application files.



## Caution

Refer to **Setup programs** for specific guidance on distributing **InstallShield** setups, INF files, or **MSI packages**.

### 19.3 Related Pages

#### Edit Final Screen →

Configure the completion dialog and its checkboxes.

#### Custom Actions →

Execute additional programs, scripts, or logic during installation.

#### UAC and Manifest →

Control elevation and admin privilege behavior.

## MSI Packages



Learn how to wrap MSI and MSP packages.

## 20. Functionality

The **Functionality** section controls runtime behavior: extraction rules, shortcuts, registry integration, language support, and package execution details.

### Use this section when

You already know what files you want to ship, but you now need to define how the package behaves when end users run it.

#### Extraction options →

Control overwrite rules, extraction flow, and how files are delivered.

#### Create shortcuts →

Create Start Menu, desktop, or custom shortcuts during installation.

#### Registry entries →

Write or remove registry data as part of the package lifecycle.

#### Protect files →

Apply password protection and content integrity behavior.

#### Language manager →

Build multilingual packages with localized strings and UI.

#### Progress monitor →

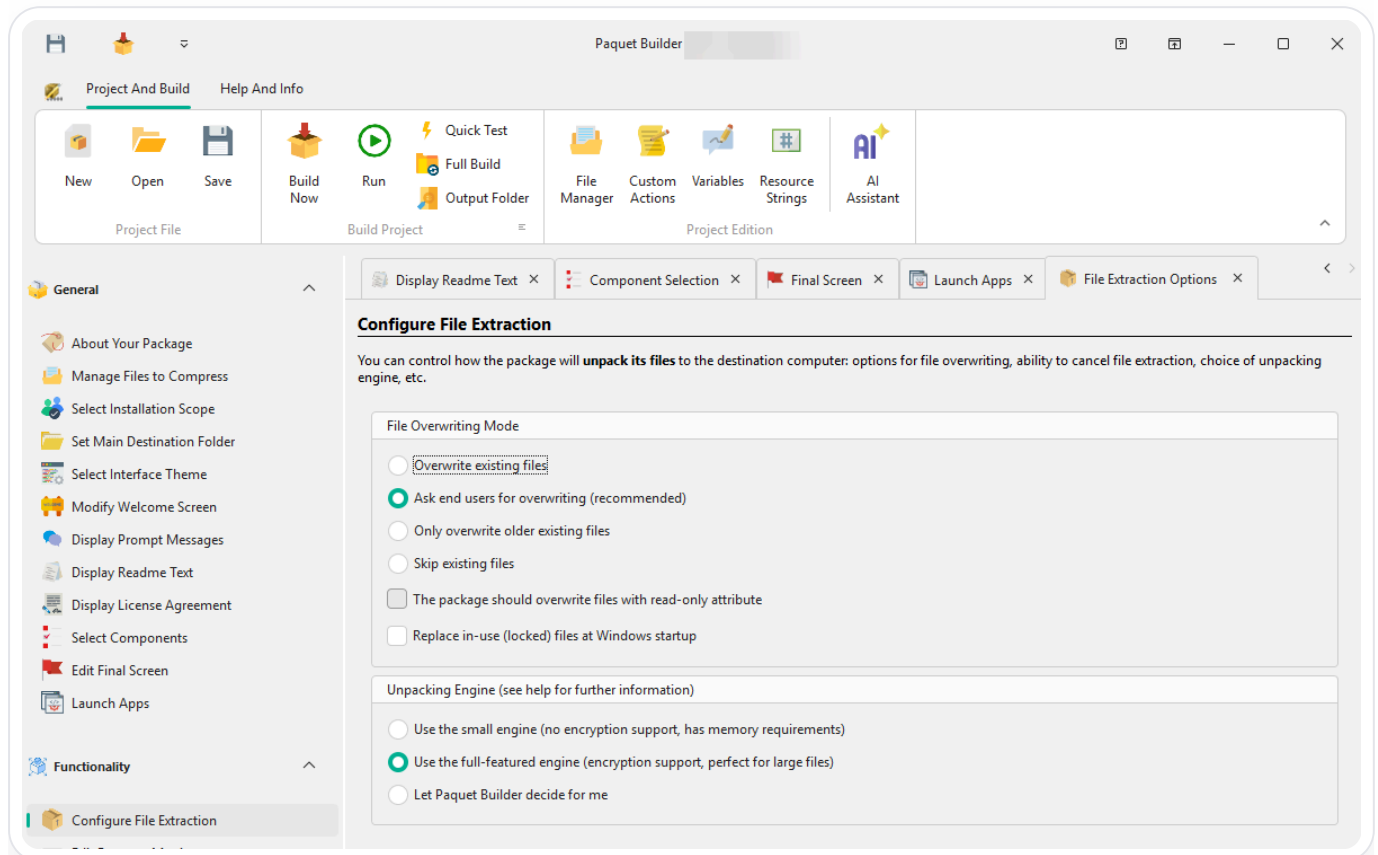
Control what users see while extraction or setup is running.

#### Package uninstaller properties →

Configure how uninstall support is created and presented.

# 21. File Extraction Settings

This page controls how the package **unpacks files** on the destination machine: what happens when a file already exists, how locked files are handled, and which decompression engine is used.



## 21.1 Overwrite Modes

Displays a prompt asking the user whether to overwrite, skip, or apply the choice to all remaining files.

This is the **recommended** setting for **standard packages**.

Replaces existing files automatically without prompting.

### **Note**

The package cannot overwrite files currently locked by Windows (e.g. active DLLs or EXEs). See the locked files option below.

Compares modification dates and only replaces the destination file if the packaged version is **newer**.

Never overwrites existing files, regardless of version or date.

## Tip

For complex version-checking logic, use [Custom Actions](#) and [File Properties](#).

## 21.2 Advanced Options

### Overwrite read-only files

Clears the “Read-Only” attribute of an existing file before attempting to overwrite it.

### Replace locked files at startup

If a file cannot be replaced because it is in use, the package queues the replacement for the next **Windows reboot**. The `%NEEDREBOOT%` variable is set to 1 if a reboot is required. Requires [Administrative Privileges](#).

## 21.3 Decompression Engines

Paquet Builder includes two high-performance engines based on 7-Zip technology:

| Engine              | Characteristics  |
|---------------------|--|
| <b>Small Engine</b> | Optimized for size. Supports LZMA/LZMA2 and BCJ2. No encryption. Requires enough RAM to hold the largest single file (avoid for files > 256 MB). |
| <b>Full Engine</b>  | Supports AES-256 encryption and multi-threaded decompression. Writes directly to disk — ideal for very large files.                              |
| <b>Automatic</b>    | <b>(Recommended)</b> Paquet Builder selects the best engine based on your project settings. The choice is logged in the compilation log.         |

## 21.4 Related Pages

### File Destinations

Configure per-file extraction paths and overwrite behavior.

### Progress Monitor

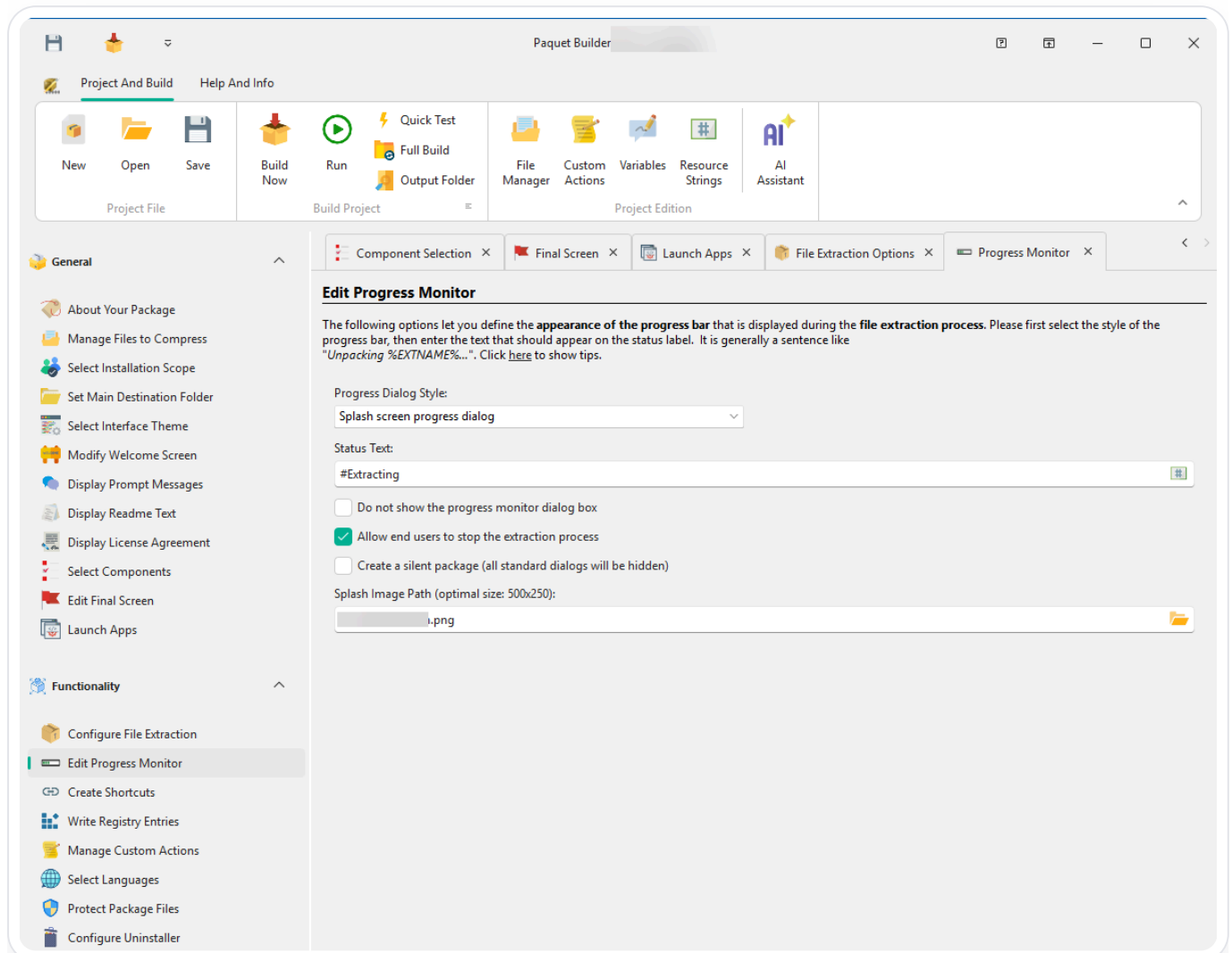
Control what users see during extraction.

## UAC and Manifest →

Administrative privileges for locked file replacement.

## 22. Edit Progress Monitor

The Progress Monitor is the dialog shown to end users **during file extraction**. You can choose its visual style, customize the status text, and optionally hide it entirely for silent packages.



### 22.1 Progress Dialog Style

Displays a large **progress bar** and a custom text message. Works with all **interface themes**.

A wizard page with an integrated progress bar, matching the look of other wizard screens. Only available when the interface theme is set to **Wizard** or **Modern**.

### Tip

This option is recommended for wizard-style packages and produces smaller package sizes.

Displays a **custom image** alongside the progress bar. Specify the image path (PNG or BMP) in the **Splash Image Path** field.

**Recommended size:** 500 × 250 pixels (images are automatically resized if needed).

## 22.2 Status Text

Enter the text displayed on the progress monitor during extraction. Use the `%EXTNAME%` variable to show the name of the file currently being unpacked.

Extracting %EXTNAME%

This displays, for example: *"Extracting HESetup.cab"* while that file is being extracted.

## 22.3 Options

### Hide progress monitor

The extraction runs silently in the background with no visible progress indicator.

### Allow users to stop

Adds a **Stop** button to the progress dialog. If clicked, the user confirms cancellation and the package closes **without removing extracted files** (no rollback).

### Silent package

Hides **all standard dialog boxes** and extracts files directly to the specified folder. **Prompt messages** and **custom action** dialogs are still shown. Use the `%SILENT%` variable to toggle at runtime.

### Caution

To define a **silent mode command-line switch**, use the **custom action template Enable Silent Mode from Package Command Line** and import it into the **"Initialization"** event.

## 22.4 Related Pages

### Interface Theme [→](#)

Choose the visual style of installer dialogs.

### Custom Actions [→](#)

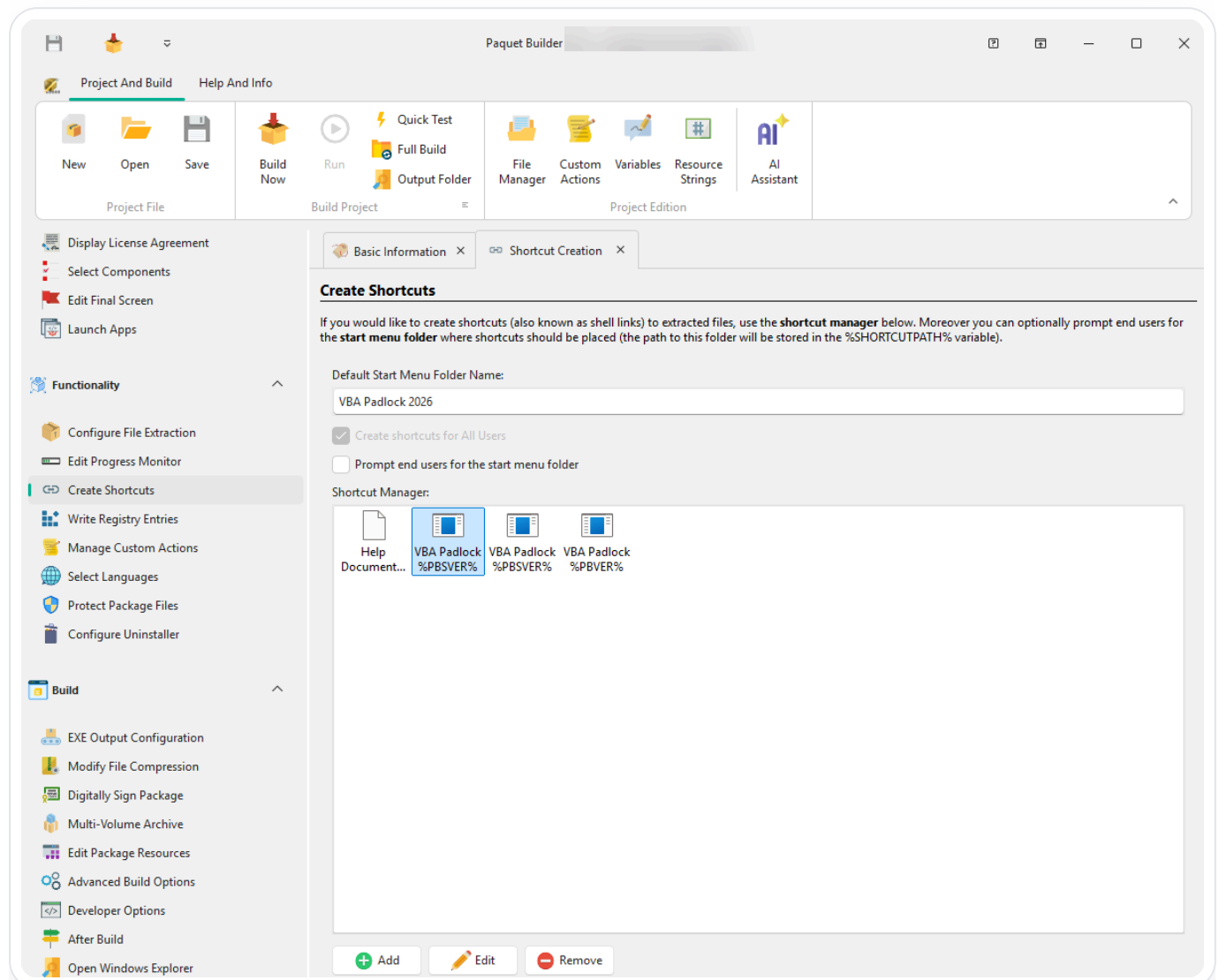
Add logic, prompts, and scripted behavior to the installation flow.

### Action Templates [→](#)

Import ready-made action templates like silent mode switches.

## 23. Shortcut Management

The Shortcut Manager lets you create shortcuts (shell links) to your extracted files and place them on the Desktop, in the Start Menu, or within custom program groups.



### 23.1 Using the Shortcut Manager

#### Add

Click **Add** to select a file from your project and open the Shortcut Editor.

#### Edit

Double-click any shortcut to modify its properties.

 **Remove**

Select a shortcut and press **Del**.

 **Copy / Paste**

Right-click a shortcut to duplicate it — useful for creating multiple targets for the same file.

 **Tip**

You can also create shortcuts directly from the **File Manager** using the right-click context menu.

## 23.2 Start Menu Folder

Group multiple shortcuts into a dedicated Start Menu folder (Program Group):

- 1 Enter a **Default Name** for the folder (e.g. My App 2.0).
- 2 Optionally enable **Prompt User** to let end users rename the folder during installation.
- 3 Choose whether to create shortcuts for **All Users** (common Start Menu, requires admin rights) or **Current User** only.


The selected folder path is stored in `%SHORTCUTPATH%`, which you can use in **custom actions**.

 **Tip**

When using **Installation Scope**, the variables `%PBSHORTCUTROOT%` and `%PBDESKTOPROOT%` automatically resolve to the correct Start Menu and Desktop folder. Use these instead of hardcoded paths.

## 23.3 Shortcut Properties

Shortcut Properties
✕



**Shortcut Editor**  
Use the fields below to modify the properties of the selected shortcut.

First enter the filename of your shortcut (including the **.lnk** extension): for example, `%STARTMENU%\Start My Program.lnk` will create a link in the Start Menu called "Start My Program".

`%COMDESKTOPDIR%\Adminer MySQL admin.lnk`

Then specify the document or program file that should be opened when end users double-click or select the shortcut. Must be a full path like `%DESPATH%\MyProgram.exe`.

Target File:

`%DESPATH%\adminer-mysql.exe`

Command Parameters\*:

Shortcut Description\*:

Icon Resource File\*:

'Start In' Folder Path\*:

You can optionally select the component this shortcut belongs to. The shortcut will be created if the component is installed.

Component:

Main

\* = optional

| Property           | Description  |
|--------------------|--|
| <b>Filename</b>    | Full path to the <code>.lnk</code> file (e.g. <code>%DESKTOPDIR%\App.lnk</code> )  |
| <b>Target</b>      | File or URL to execute (e.g. <code>%DESPATH%\app.exe</code> )  |
| <b>Parameters</b>  | Optional command-line arguments passed to the target   |
| <b>Description</b> | Tooltip text shown when hovering over the shortcut   |
| <b>Icon File</b>   | Path to an <code>.ico</code> , <code>.exe</code> , or <code>.d11</code> file. Use index syntax for multi-icon files: <code>file.d11 2</code> |
| <b>Start-In</b>    | Working directory for the application when launched (commonly <code>%DESPATH%</code> )   |

| Property         | Description  |
|------------------|--|
| <b>Component</b> | Link to a <b>component</b> — if skipped, the shortcut is not created |

### Caution

If you create shortcuts for **All Users**, make sure the icon resource file is accessible to all users — place it in a shared location such as %DESKTOP%, not in a user-specific folder.

## 23.4 Useful Variables

| Variable           | Description  |
|--------------------|--|
| %PROGMENU DIR%     | Current user's Programs folder (Start Menu)            |
| %COMPROGMENU DIR%  | Common (All Users) Programs folder (Start Menu)        |
| %PB SHORTCUT ROOT% | Start Menu folder adapted to <b>installation scope</b> |
| %PB DESKTOP ROOT%  | Desktop folder adapted to <b>installation scope</b>    |
| %SHORTCUT PATH%    | Start Menu folder selected by the user (when prompted) |
| %DESKTOP DIR%      | Current user's Desktop folder                          |

## 23.5 Internet Shortcuts

To create Internet shortcuts (.url files that open a web page), use the **Create an Internet Shortcut** custom action instead.

### Note

Paquet Builder automatically removes all shortcuts created by the package when the **Uninstaller** is run.

## 23.6 Related Pages

## Create Shortcut (Custom Action) →

Advanced shortcut creation with custom actions.

## Installation Scope →

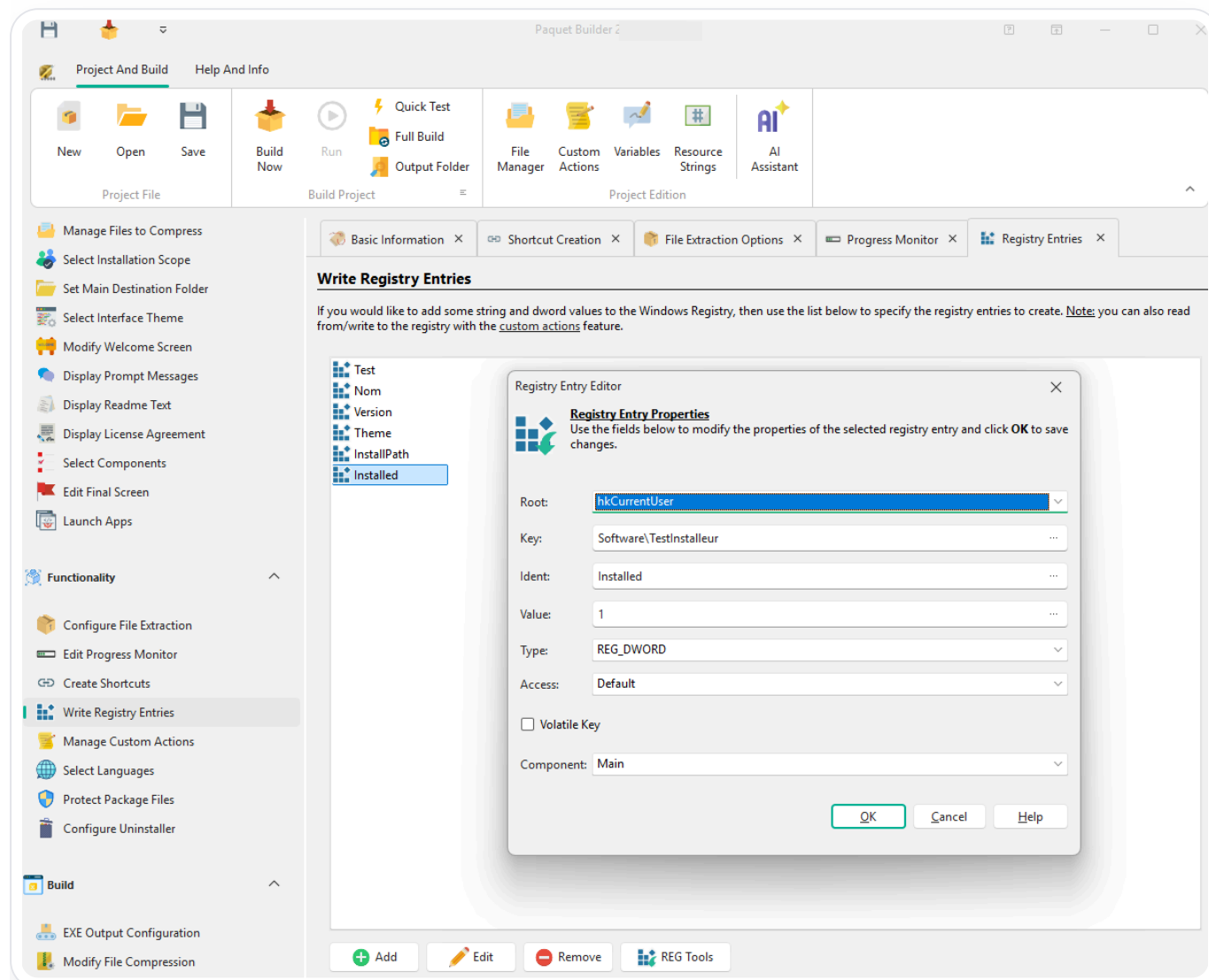
Control per-user vs. all-users shortcut placement.

## Variable List →

Full list of available runtime variables.

## 24. Write Registry Entries

The Registry Entry Manager lets you configure registry entries that your package will create or modify on the destination computer. It provides a visual interface that simplifies what can also be done with [custom actions](#).



### 24.1 Managing Entries

- 1 **Add an entry** — Click the **Add** button to open the entry editor dialog.
- 2 **Edit an entry** — Select an entry and click **Edit**, or double-click the entry directly.
- 3 **Remove an entry** — Select an entry and click **Remove**, or press **DEL**.
- 4 **Import/Export** — Use the **REG tools** button to import or export `.reg` files.

## Tip

Right-click any registry item to **copy**, **cut**, or **paste** entries. This is useful for duplicating entries with similar properties.

## 24.2 Import / Export .reg Files

You can import registry entries from .reg files created with **RegEdit** (RegEdit.exe).

| Format                    | Supported |
|---------------------------|-----------|
| REGEDIT 5.0 (Unicode)     | Yes       |
| REGEDIT 4.0 (Non-Unicode) | Yes       |
| Other formats             | No        |

Only **string**, **dword**, and **qword** values are imported — other entry types are ignored.

Exported .reg files are fully compatible with RegEdit and can be edited with any text editor.

When exporting from RegEdit, select the **REGEDIT5** format for best compatibility.

## 24.3 Entry Editor

The registry entry editor allows you to create or modify individual settings. Refer to the [Write a registry entry](#) custom action topic for detailed field explanations.

### 24.3.1 Supported Entry Types

| Type            | Registry Type |
|-----------------|---------------|
| String          | REG_SZ        |
| Expanded string | REG_EXPAND_SZ |
| DWORD (32-bit)  | REG_DWORD     |

| Type           | Registry Type |
|----------------|---------------|
| QWORD (64-bit) | REG_QWORD     |

### Note

To modify the **default value** of a registry key, leave the **Ident** field blank.

## 24.3.2 Component Assignment

Each registry entry can be assigned to a **component**. The entry will only be created if the associated component is selected for installation. Select **None** to always create the entry.

## 24.4 Accessing an Alternate Registry View

By default, a 32-bit package accesses the 32-bit registry view on WOW64, while a 64-bit package accesses the 64-bit view. ARM64 packages access the native 64-bit registry view, the same as 64-bit packages.

Custom actions for registry modifications have an `Access` property to specify the desired view:

| Value                        | Behavior  |
|------------------------------|---|
| <code>regaDefault</code>     | Default behavior based on the package type                          |
| <code>regaWOW64_64KEY</code> | Access a 64-bit registry key from either a 32-bit or 64-bit package |
| <code>regaWOW64_32KEY</code> | Access a 32-bit registry key from either a 32-bit or 64-bit package |

### Caution

On 32-bit systems, `regaWOW64_32KEY` and `regaWOW64_64KEY` are ignored without errors.

Refer to [Microsoft Windows documentation](#) for more details.

## 24.5 Important Considerations

### Caution

Editing the Windows Registry is recommended only for users familiar with its functionality.

### Note

Modifying registry entries under `HKEY_LOCAL_MACHINE` requires administrative privileges. Use the `%ISADMIN%` [variable](#) to check user permissions.

### Tip

When using [Installation Scope](#), the registry root key (`HKEY_CURRENT_USER` or `HKEY_LOCAL_MACHINE`) is automatically determined by the selected scope. This simplifies creating installers that support both per-user and all-users installations.

Registry entries created through the manager are automatically removed by the [uninstaller](#) by default.

## 24.6 Related Topics

### Write a registry entry [→](#)

Custom action reference for registry write operations.

### Read from the registry [→](#)

Read registry values into variables at runtime.

### Remove a registry key [→](#)

Delete registry keys or entries during installation.

### Check if a registry key exists [→](#)

Conditional logic based on registry key presence.

# 25. Language and Localization

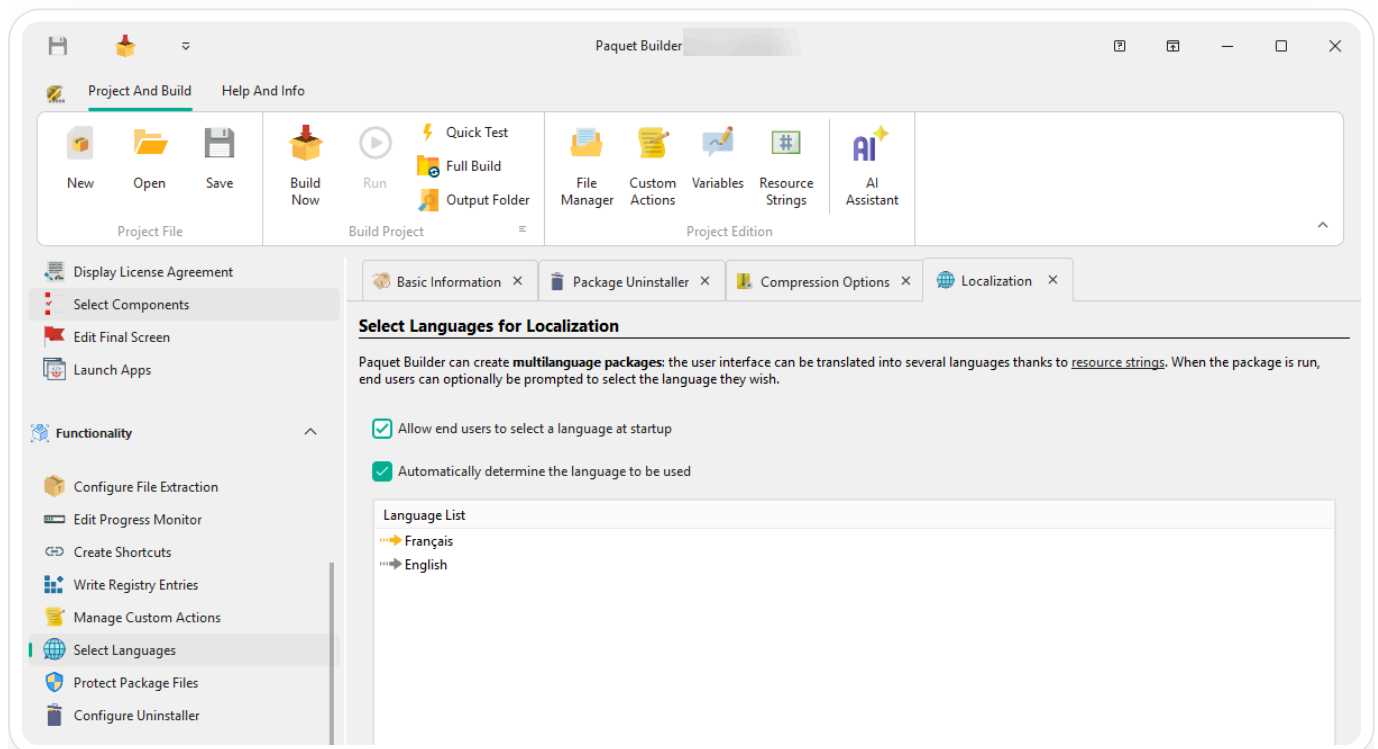
A **language** in Paquet Builder is a collection of **resource strings** translated into a specific language (e.g. English, French). Each language is stored in a `.pbr` file. By including several language files in your project, you create a multilingual package that adapts its text to the end user.

## Tip

For details on editing and translating text values, see [Working with Resource Strings](#).

## 25.1 Managing Languages

The **Language Manager** handles the collection of language files compiled into your package.



## Add

Include a new language file (`.pbr`) in the project.

## Edit

Double-click a language to open the **Resource String Editor**.

## ☆ Default

Mark a language as the fallback (



icon). When no system match is found, the package uses this language.

## ✕ Remove

Exclude a language from the compilation.

## ⚠ Caution

Every package must include at least **one default language**.

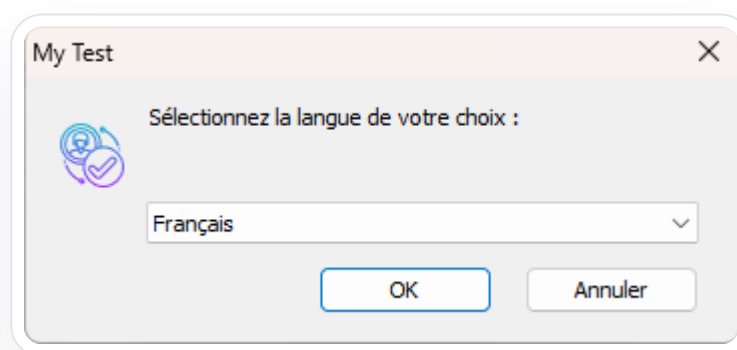
## 25.2 Language Detection

### Select Languages for Localization

Paquet Builder can create **multilanguage packages**: the user interface can be translated into several languages thanks to [resource strings](#). When the package is run, end users can optionally be prompted to select the language they wish.

- Allow end users to select a language at startup
- Automatically determine the language to be used

A dialog asks the user to choose from the available languages immediately upon launch.



The package checks the operating system's **Locale ID** (LCID) and matches it against the included languages. If a match is found, that language is used silently. The detected LCID is stored in `%PBLANGID%` and the language name in `%SELLANG%`.

The package first detects the system language automatically and **pre-selects** it in the language dialog. The user can still change the selection before continuing.

## 25.2.1 RTL Support

Enable **Right-to-Left** layout for languages such as Arabic or Hebrew. This option is set per language in the [Resource String Editor](#).

## 25.3 Setting the Language Programmatically



### Via custom action

Change the active language at any point by assigning a new LCID to %PBLANGID% using a **Modify Variable** action. For example, 1036 for French.



### Via command line

Pass `/LANG 1036` as a command-line argument and process it with a **custom action** to force a specific language.

## 25.4 Role of the Default Language

The default language acts as a **fallback**. If a resource string has no translation in the user's selected language, the package uses the default language text instead.

### Caution

If the default language text for a screen is blank, that screen will **not** be shown at runtime — even if translations exist in other languages. Always fill in the default language values first.

### Tip

If you don't need a multilingual package, include only one language and set it as default. The language selection dialog will not appear.

### Tip

Use the **AI-powered translation** feature within the Resource String Editor to quickly generate translations for your entire project.

## 25.5 Related Pages

## Resource Strings →

Edit and translate all dialog texts and labels.

## Welcome Screen →

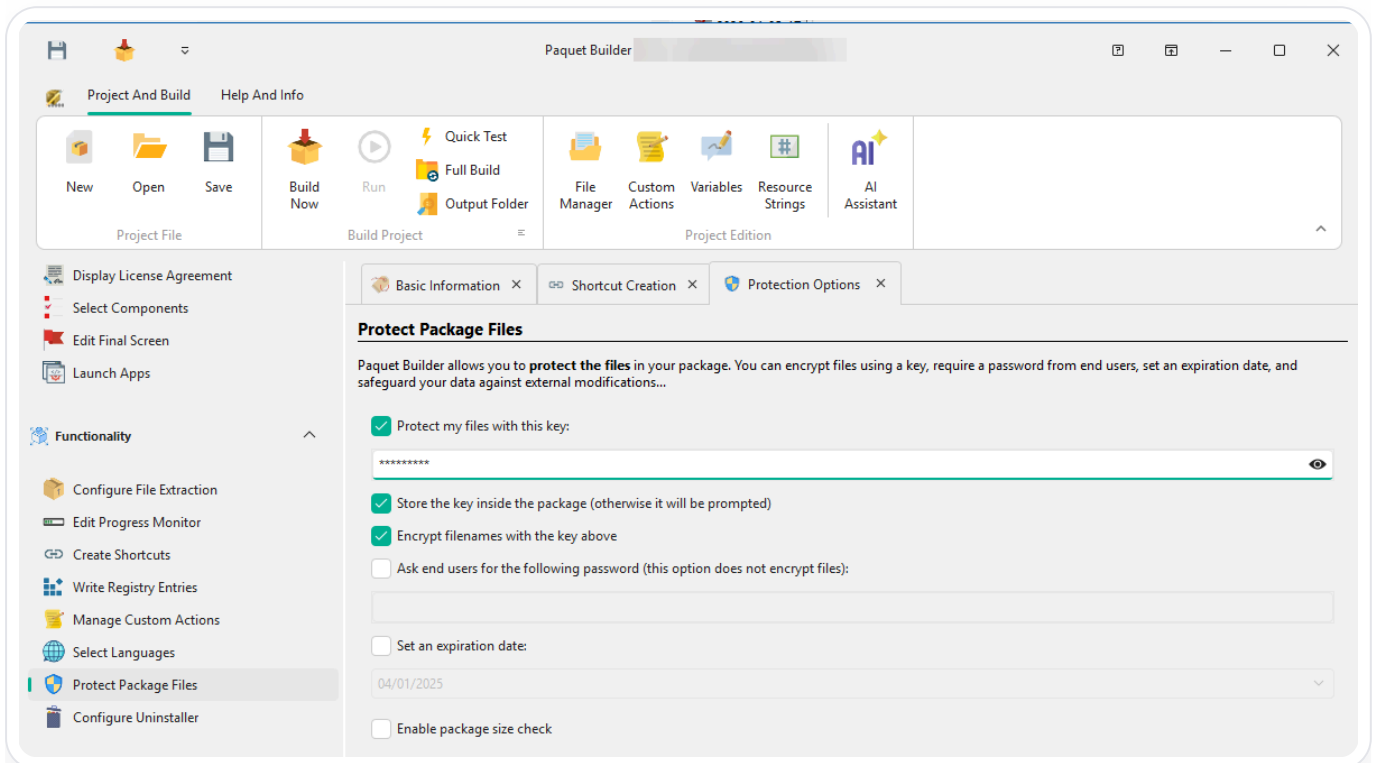
Write localized welcome messages for each language.

## Custom Action Templates →

Import ready-made templates including silent mode switches.

## 26. Protect Package Files

Paquet Builder provides several security features to protect your packages and their contents against tampering, unauthorized access, or incomplete downloads.



### 26.1 File Encryption and Passwords

Enter a key to **encrypt your files** using 7-Zip AES encryption. This prevents extraction with external tools without running your package.

By default, end users are **prompted for the key** before extraction. If the wrong key is entered, the package closes.

### Caution

If you use an **external .7z archive**, files inside the archive will not be encrypted. Encryption only applies during file compression.

Enable **“Store the key inside the package”** to avoid prompting users. The package stores the key internally and uses it automatically during extraction.

Combine this with **“Encrypt filenames”** to also prevent unauthorized users from viewing the file list with hex editors or 7-Zip.

Enter a password that users must provide to proceed. This option **does not encrypt** files but adds a layer of protection by requiring the correct password before extraction begins.

## 26.2 Additional Protection

### Expiration date

Set a **trial expiration date**. After this date, the package displays an error (*“Package Expired!”*) and exits. Primarily intended as a reminder to download a newer version.

### Package size check

Verifies the package file size at startup. If it differs (e.g. due to incomplete download or truncation), the package exits with an error. Incompatible with digital signing.

### Tip

For production-grade integrity verification, use **digital signing** instead of the size check. Code signing certificates ensure the package has not been tampered with.

## 26.3 Related Pages

## Digital Signature



Sign your package with Authenticode or Azure Artifact Signing.

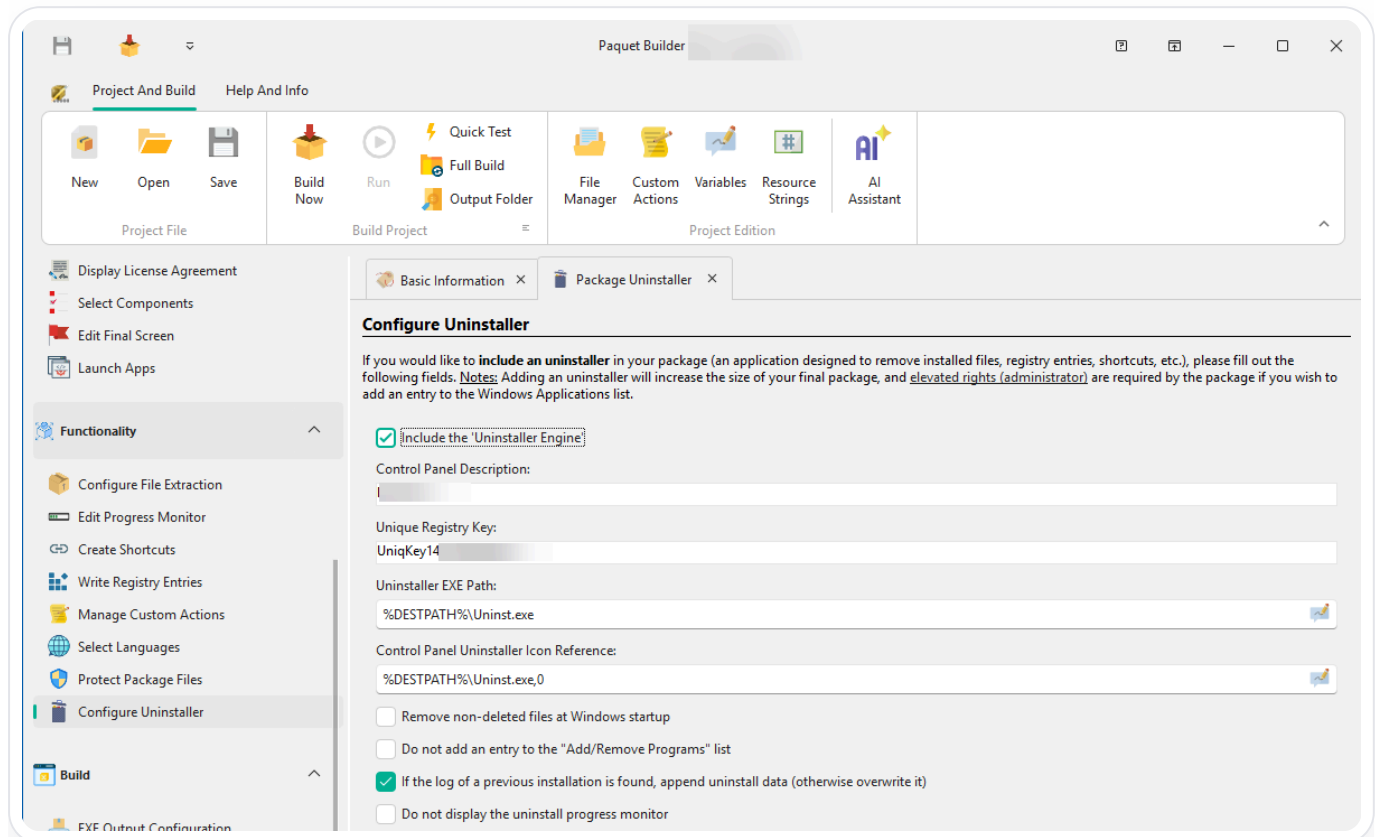
## File Manager



Manage files and external archives in your package.

## 27. Configure Uninstaller

The uninstaller is an application generated by your package that removes installed files, registry entries, shortcuts, and other changes. This page configures its properties.



### 27.1 Core Settings

| Field                            | Description  |
|----------------------------------|--|
| <b>Control Panel Description</b> | Text displayed in <i>Add/Remove Programs</i> (e.g. My Application 1.7)   |
| <b>Unique Registry Key</b>       | A unique subkey name for the registry (e.g. CompanyName_App_1.0 or a GUID).<br>Retrievable at runtime via %PBUNINSTREGKEY% |
| <b>Uninstaller EXE Path</b>      | Full path for the generated executable (e.g. %DESPATH%\Uninstall.exe). A .log file is created alongside it                 |
| <b>Icon Reference</b>            | Icon for the <i>Add/Remove Programs</i> entry — .ico file or resource in .exe/.dll (e.g. %DESPATH%\Uninst.exe,0)           |

## Caution

Adding an entry to *Add/Remove Programs* requires administrative privileges. See [UAC elevation](#).

## Note

The uninstaller and log files are created between the “After File Extraction” and “Finalization” events. To include custom actions for the uninstaller, add them to the “After File Extraction” event or use specific uninstaller events.

## 27.2 Additional Options

### Remove locked files at startup

The uninstaller removes in-use files (e.g. shared DLLs) during the next **Windows reboot**.

### Hide from Add/Remove Programs

Does not create a Control Panel entry. Ensure users have another way to run the uninstaller.

### Append to existing logs

If a previous `.log` file exists, new changes are appended instead of overwriting — useful for packages that install different components over time.

### Hide progress monitor

Runs the uninstaller silently without showing a progress bar.

### Skip admin check

Allows non-administrators to run the uninstaller. Changes the requested execution level from *Require Administrator* to *Highest Available*.

## 27.3 Silent Uninstaller

- 1 Add a [Modify Variable](#) custom action in the “**Uninstall Initialization**” event.
- 2 Set `%SILENT%` to `1` — this prevents all prompts (except unexpected errors) during uninstallation.

 **Tip**

To enable silent mode via command line, import the [custom action template Enable Silent Mode from Package Command Line](#) into the “**Uninstall Initialization**” event.

## 27.4 Related Pages

### Package Uninstaller [→](#)

Understand how the uninstaller works and what it removes.

### UAC and Manifest [→](#)

Control elevation and admin privilege behavior.

### Custom Actions [→](#)

Add logic to the uninstall flow with custom events.

## 28. Build Settings

The **Build** section is where Paquet Builder turns your project into a distributable artifact. These topics cover output paths, compression, trust, resources, and specialist build options.

### Think of this section as final assembly

If the setting affects the generated EXE or archive rather than the end-user workflow itself, it usually belongs here.

#### Package output →

Choose output folder, filename, bitness, and build trigger options.

#### Compression options →

Tune archive size, compression method, and build performance.

#### Digital signature →

Sign the output with Authenticode and improve trust on client systems.

#### SignTool code signing →

Configure Microsoft SignTool based signing workflows.

#### Azure Artifact Signing →

Use Microsoft's cloud signing infrastructure for package signing.

#### Package resources →

Set version info, icons, and embedded resources for the final EXE.

#### Advanced build options →

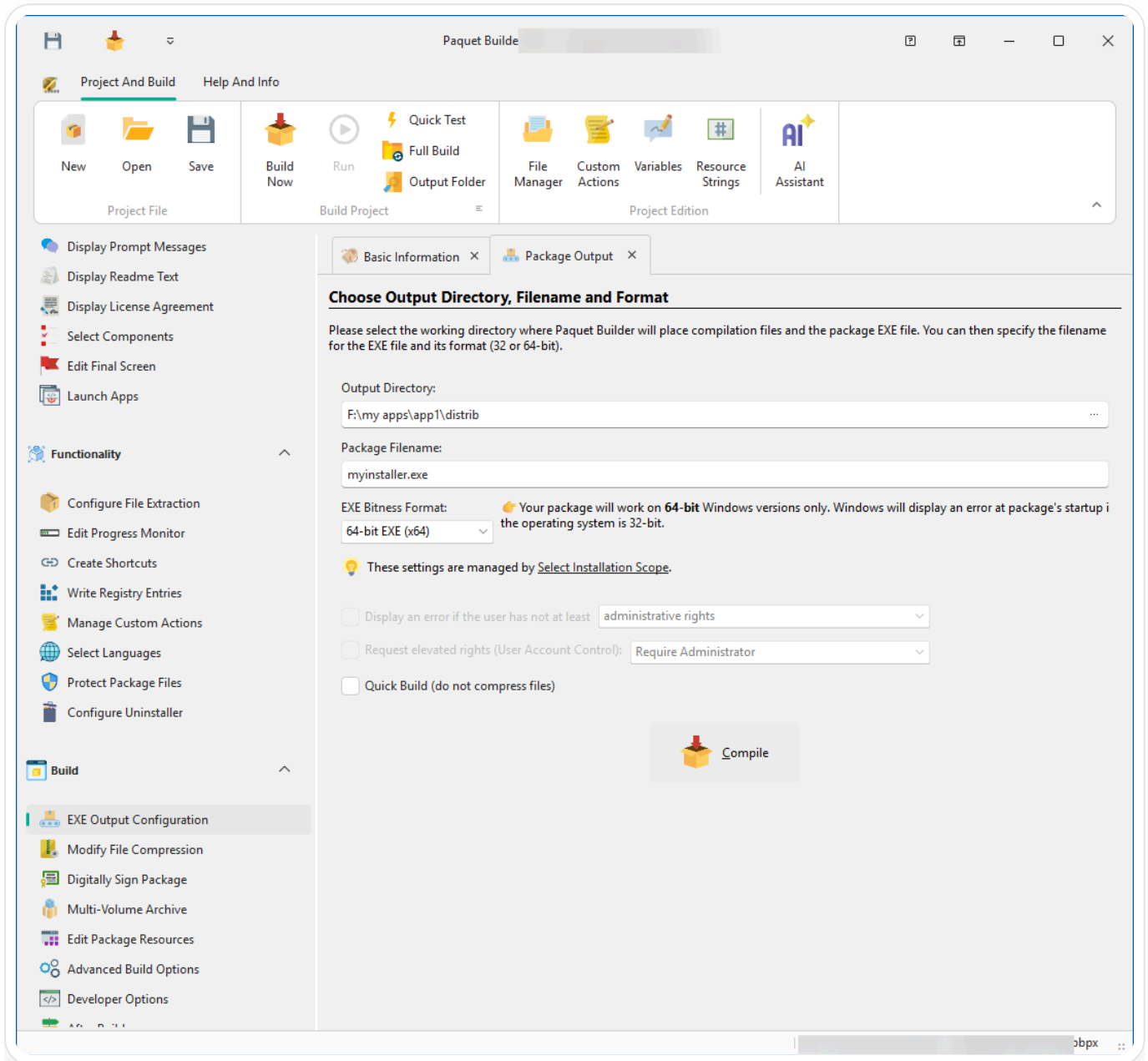
Adjust temporary folders, executable packing, and low-level build details.

#### After build →

Run tasks after compilation completes.

## 29. Set output EXE file path

This page controls where the final package is written, which binary format is generated, and how compilation behaves.



### 29.1 Building your package

#### Output directory

Choose the folder where Paquet Builder writes the final package and the temporary build files it needs during compilation.



## Package filename

Set the output file name. Use `.exe` for executable packages or `.7z` for archive output when the project type supports it.



## Build trigger

Launch a build immediately from this page with **Build Now**, keyboard shortcuts, or command-line automation.

### 29.1.1 Output Directory

Specify the full path to the folder where Paquet Builder should place the package file and its temporary `bin` folder. If the target folder does not exist yet, Paquet Builder creates it automatically.

### 29.1.2 Package Filename

Define the file name without the path. Use an `.exe` extension for executable packages or `.7z` for archive output, depending on the project type you created.

### 29.1.3 EXE Bitness Format

Paquet Builder can create x86, x64, and ARM64 packages. Choose the target that matches the software you want to install and the registry view you need to access.

---

**32-bit packages** can run on both 32-bit and 64-bit Windows systems. They are the right choice for 32-bit application deployments or when you intentionally want 32-bit registry redirection.

---

**64-bit packages** run only on 64-bit Windows. They are typically used when your package installs 64-bit software or must access the native 64-bit registry view directly.

---

**ARM64 packages** target Windows on ARM (e.g. Snapdragon) and install native ARM64 software. They also use the native ARM64 registry view. ARM64 packages **do not** run on x86 or x64 systems.

On x64 Windows, x64 packages write directly to the native 64-bit registry, while x86 packages are redirected under `Wow6432Node`. See [Registry Entries](#) for details.

## Notes

- 1 Save the project before compiling.
- 2 Existing package files with the same name are overwritten.

- 3 Make sure all source files and resources are available.
- 4 Ensure there is enough free disk space for both the output and temporary build files.
- 5 Paquet Builder creates a temporary `bin` folder during compilation and removes it afterwards unless you keep it through [Advanced Build Options](#).
- 6 If needed, you can also [create multi-volume archives](#).

### 29.1.4 Shortcuts for Launching Compilation

- Click **Build Now** to start compilation.
- Press **CTRL+F9** or **F10** for a full build.
- Use [command line parameters](#).

## 29.2 About Archive Caching

When building a package, Paquet Builder compresses files into an archive before converting them into a self-extracting executable. Compression is the most time-consuming process.

To save time, Paquet Builder **caches the archive** in a temporary folder. If the source files and compression settings remain unchanged, the cached archive is reused for subsequent builds.

### Why caching matters

Archive caching is especially useful while you are iterating on screens, custom actions, or other package logic that does not require recompressing all source files every time.

To force a full archive rebuild, press **F10** or choose **Build Full Package** from the ribbon split button. The cache is cleared automatically when you close Paquet Builder or switch to another project. It is stored in the working folder defined in [Environment Options](#).

## 29.3 Build Options

### 29.3.5 Request Elevated Rights (User Account Control)

Configure how the package requests elevated rights on Windows 7 and later systems.

### Caution

Review the article on [UAC and Paquet Builder compatibility](#).

### 29.3.6 Display an Error if the User Lacks Administrative Rights

Enable this option if the package must run only under administrative or power-user rights. This is important for tasks such as modifying protected system areas or writing to privileged registry locations.

### Note

The %ISADMIN% variable can also be checked using [custom actions](#).

## 29.3.7 Interaction with Installation Scope

### When Installation Scope is active

When an [Installation Scope](#) mode is configured (Current User, All Users, User Choice, or Automatic), the UAC and administrative rights options above are **disabled** on this page. A banner indicates that these settings are now managed by the scope system.

Scope mode automatically determines whether the installer requests elevated rights and how it handles missing admin privileges — there is no need to configure them manually here. To regain manual control, set the scope to **None (Legacy)** on the Installation Scope page.

## 29.3.8 Quick Build

For testing purposes, files are not compressed and the package is not digitally signed, even if signing is configured. This is equivalent to clicking **Quick Test** in the ribbon.

## 29.3.9 7Z Archive Projects

When the project type is **7Z archive** (not a self-extracting EXE), the following options are hidden or disabled on this page because they only apply to executable packages:

- EXE Bitness Format
- Request Elevated Rights
- Display an Error if the User Lacks Administrative Rights
- Quick Build

The page header adapts to show “Choose Output Directory and Filename” instead of the full EXE-oriented heading.

## 29.4 Compilation Log

The compilation log shows a detailed, real-time view of the build process:

- **Build phase stepper** at the top of the window
- **Collapsible sections** for each build phase

- **Status badges** for warnings, notes, informational messages, and success states
- **Copy path button** for the final package location
- **Light and dark theme support** to match the application appearance

### Tip

Enable “Display the compilation log immediately when building packages” in [Environment Options](#) to watch the log being filled during compilation.

## 29.5 Cancel

During compilation, you can cancel the process with the **Cancel** button in the progress monitor. Cancellation is effective only between compression operations, so it may be unavailable near the end of the build.

## 29.6 Related Pages

### Installation Scope →

Control per-user or all-users installation and how it affects UAC settings on this page.

### Advanced Build Options →

Keep temporary build files, configure linker flags, and other advanced compilation settings.

### Multi-Volume Archive →

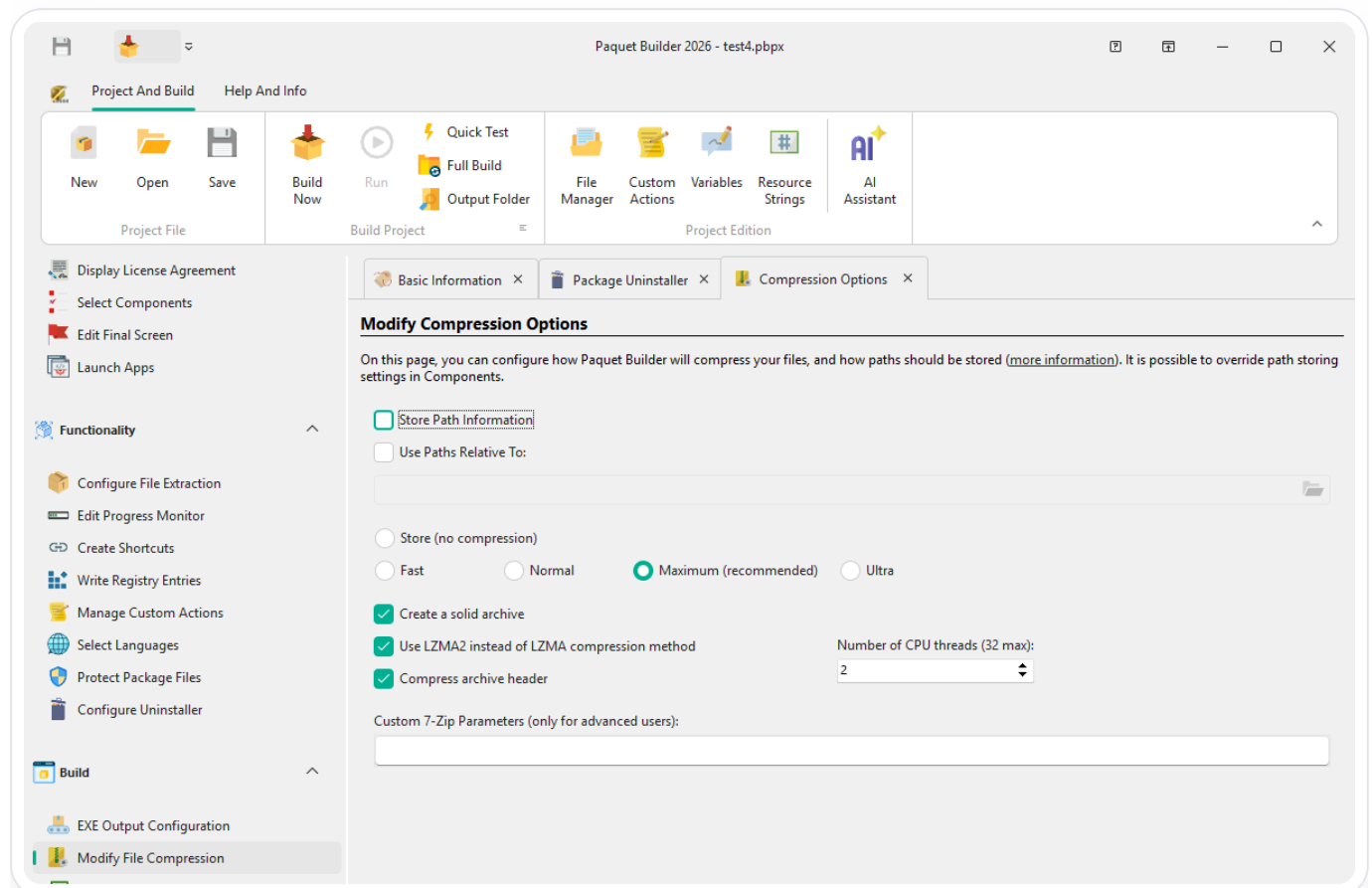
Split the output package into multiple volumes.

### UAC and Paquet Builder →

Understand User Account Control compatibility and manifest settings.

## 30. Modify File Compression

Paquet Builder uses **7-Zip** to compress source files into the package. This page controls path storage, compression level, and advanced archive parameters.



### 30.1 Storing Path Information

When source files come from multiple folders, you may need to preserve the folder structure in the archive.

Enable **“Store Path Information”** to include the full path (excluding drive letter) for each file.

**Example:**

- Source: C:\My Documents\My Results\Report.pdf
- Stored path: My Documents\My Results
- Extracted to: %DESTPATH%\My Documents\My Results\

Enable **“Use Paths Relative To”** and specify a root folder to store only a portion of the path.

**Example** (root: C:\My Documents):

- Source: C:\My Documents\My Results\Report.pdf
- Stored path: My Results
- Extracted to: %DESTPATH%\My Results\

Files outside the root folder still store their full paths.

**Note**

Path storage is important when multiple files share the same name but reside in different folders.

## 30.2 Compression Level

| Level          | Description  |
|----------------|--|
| <b>Store</b>   | No compression — files are stored as-is            |
| <b>Fast</b>    | Fastest compression, minimal ratio                 |
| <b>Normal</b>  | Balance between speed and ratio                    |
| <b>Maximum</b> | Excellent ratio — recommended for most cases       |
| <b>Ultra</b>   | Best ratio, but requires significant RAM (350 MB+) |

## 30.3 Compression Options



## Solid archive

Compresses all files as one continuous data stream for better compression ratios. Recommended for Self-Extracting packages.



## LZMA2 multithreading

LZMA2 supports **multithreaded compression and decompression**, improving speed on multi-core systems. You can specify the number of CPU threads.



## Compress archive header

Compresses file metadata and archive headers for a smaller overall package size.



## Custom 7-Zip parameters

Override defaults with custom 7-Zip commands. Use | to separate parameters (e.g. `0|BCJ2|1|LZMA2:d24`).



## Tip

Not sure which settings to use? Ask the [AI Assistant](#): *"What compression settings do you recommend for a large installer?"*

## 30.4 Related Pages

### About the 7-Zip Format →

Technical details about the archive format.

### Multi-Volume Archive →

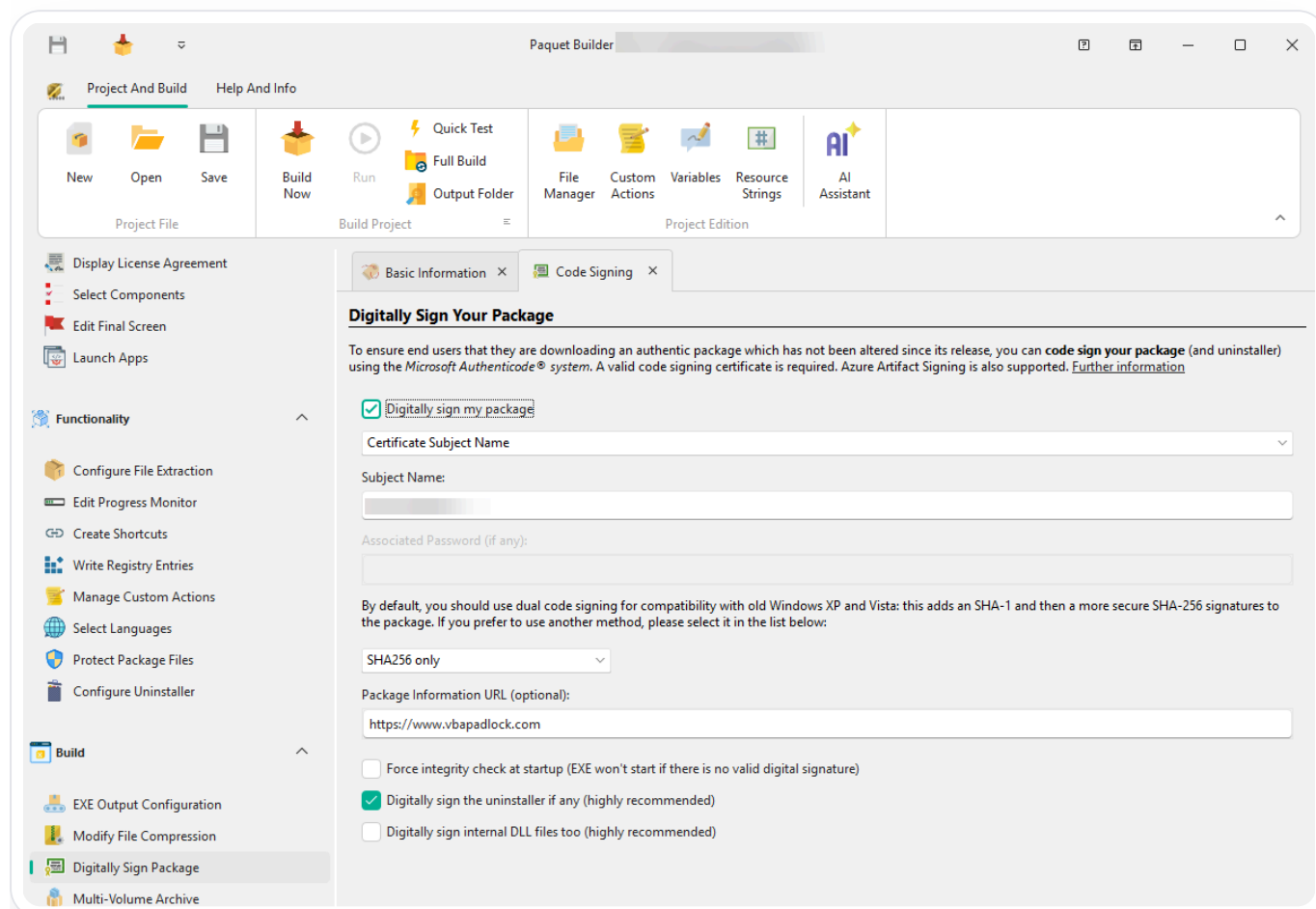
Split archives into multiple volumes for large packages.

### Components →

Override path storage settings per component.

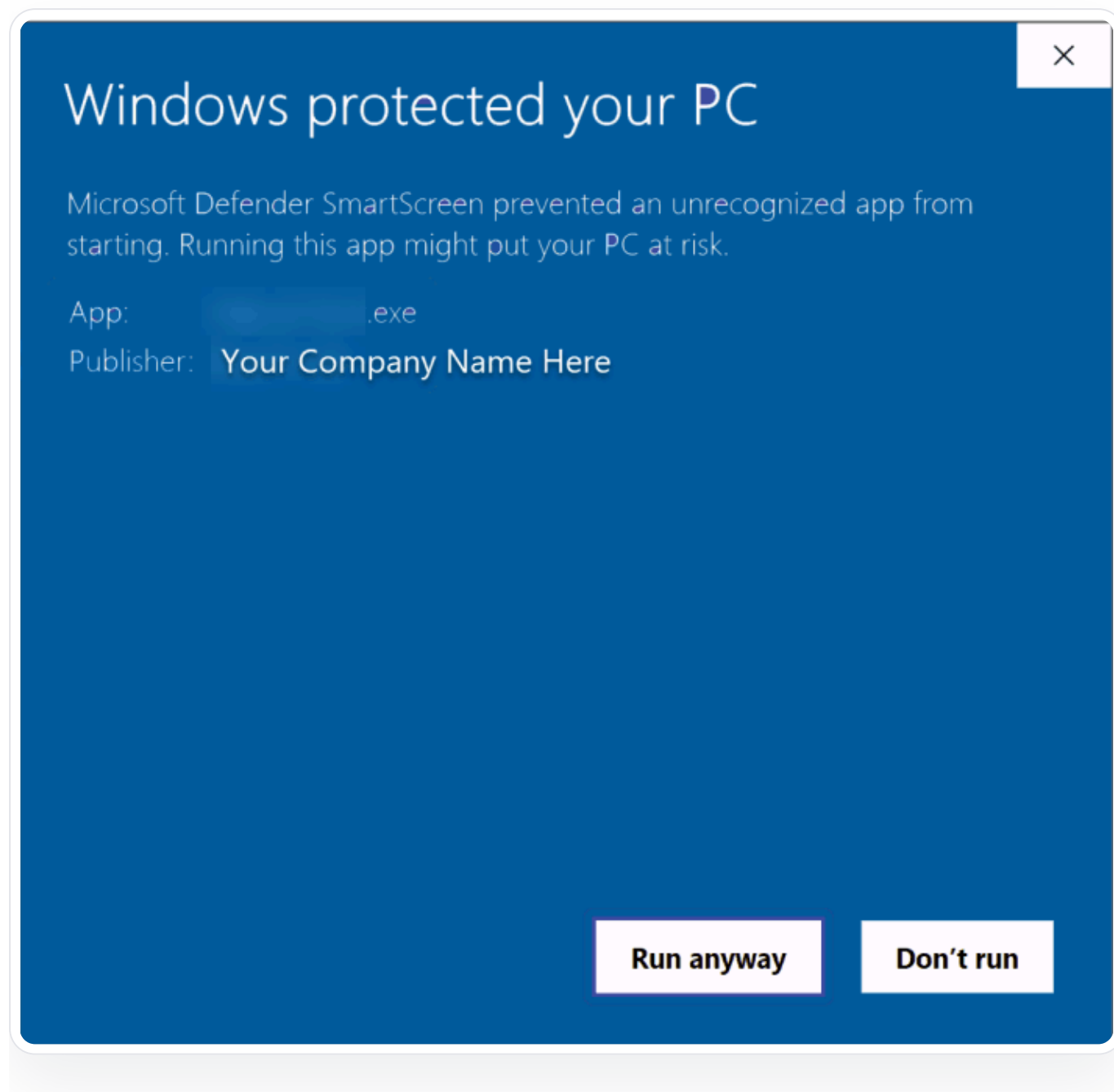
# 31. Digitally Sign Packages — Code Signing

Digitally signing your package ensures end users that the code has not been tampered with. Based on **Microsoft Authenticode**, digital signing verifies the source and integrity of your installer.



## 31.1 Signed vs. Unsigned

End users see a certificate indicating the software's origin and authenticity, reducing security warnings:



Windows displays a warning that may discourage users from proceeding:

## Windows protected your PC

Windows SmartScreen prevented an unrecognized app from starting. Running this app might put your PC at risk.

App: XXXXXXXXXX  
Publisher: Unknown Publisher

Run anyway

Don't run

### 31.2 Choose Your Signing Method



#### PFX / Certificate Store

Use a .pfx file or a certificate installed in the Windows Certificate Store. Paquet Builder signs with the built-in **GSignCode** utility.



#### Hardware token

Connect your HSM or USB token containing the private key. GSignCode adapts automatically, including ECC support.



#### SignTool Commands

Use **Microsoft SignTool** for advanced signing scenarios and full control over parameters.

[Read the guide](#) →



#### Azure Artifact Signing

Use **Microsoft's cloud signing service** — lower cost, no USB tokens, cloud-managed keys.

[Read the guide](#) →



#### JSign (open-source)

Free tool supporting **Azure Key Vault**, **AWS KMS**, **Google Cloud KMS**, hardware tokens, and more. [Read the guide](#) →

### 31.3 Obtaining a Code Signing Certificate

You need a valid code signing certificate from a trusted Certificate Authority (CA) such as Sectigo or DigiCert. Alternatively, **Azure Artifact Signing** offers a much cheaper pay-as-you-go model — see the [dedicated topic](#).

### Note

As of June 2023, certificate keys must be stored on a FIPS 140-2 Level 2 HSM or token. The traditional PFX format is being phased out.

## 31.4 Configuring the Certificate

Specify the path to your .pfx file and its password. To create a PFX from separate key files:

```
pvk2pfx.exe -pvk MyPrivateKey.pvk -spc MyPublicKey.cer -pfx MyPFX.pfx -po  
your_password
```

Provide the **subject name** or **thumbprint** of your certificate installed in the Current User or Local Computer store.

Connect your token and ensure it is accessible during the build. GSignCode adapts to the token's capabilities automatically.

### Tip

If your token uses the SafeNet client, enable "Single Logon" to enter your password only once per session.

## 31.5 Digest Algorithms

| Algorithm           | Description   |
|---------------------|---|
| <b>SHA-256</b>      | Default and recommended standard  |
| <b>Dual signing</b> | Combines SHA-256 and SHA-1 for backward compatibility with Windows XP/Vista |

### Caution

Dual code signing is supported on Windows 8+. On Windows 7, only SHA-256 is used.

## 31.6 Additional Options

### Integrity check at startup

Forces a mandatory signature validation by the OS. Unsigned files will fail to execute.

### Timestamp

Ensures the signature does not expire. Configure Authenticode and RFC-3161 timestamp servers in [Environment Options](#).

### Sign the uninstaller

Signs the [uninstaller](#) EXE file. **Highly recommended.**

### Sign internal DLLs

Signs internal runtime libraries used temporarily by your package.

### Caution

Windows cannot sign EXE files larger than 2 GB. Consider [multi-volume archives](#) if needed.

## 31.7 Related Topics

### SignTool Commands

Custom signing with Microsoft SignTool.

### Azure Artifact Signing

Microsoft's cloud-based signing service.

### Signing with JSign

Free, open-source signing with cloud KMS support.

## Environment Options →

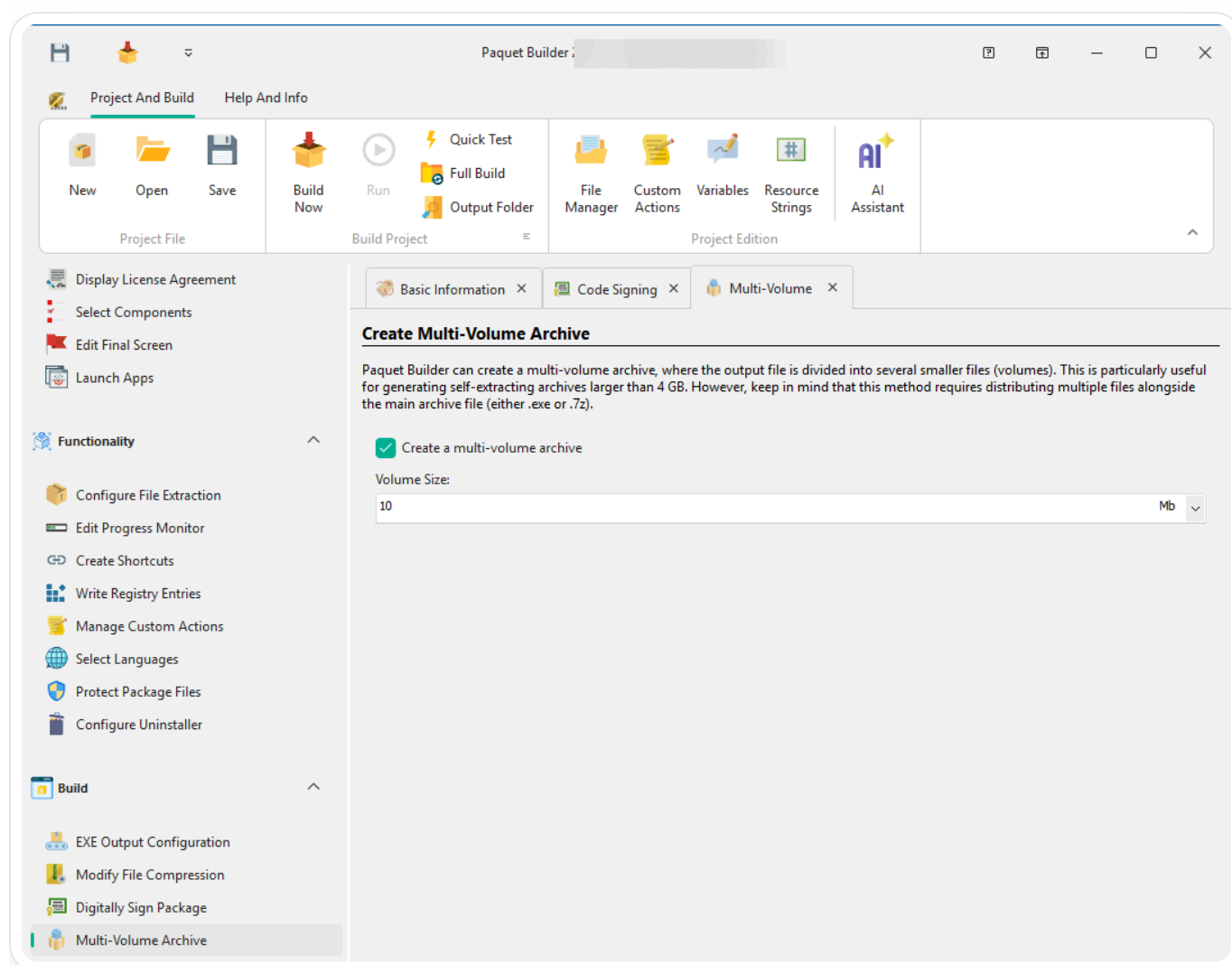
Configure timestamp servers and signing tool paths.

## Directives →

Automate signing in build scripts.

## 32. Multi-volume Archive

Paquet Builder allows you to create multi-volume archives, where the output file is **split into smaller, more manageable files** (volumes).



### 32.1 Why Use Multi-volume Archives?

Large single archive files can be difficult to share online or over networks, as they are prone to corruption and data loss. Multi-volume archives split the file into smaller chunks, making them easier to distribute and more resilient to issues during transfer.

Additionally, this feature enables the creation of Self-Extracting archives **larger than 4 GB**, bypassing the 4 GB limit for Windows EXE files.

#### 32.1.1 Drawback

The main disadvantage is the need to distribute multiple files alongside the primary Self-Extracting archive (EXE).

### Caution

The Multi-volume Archive feature is incompatible with the **“Using an existing archive”** option.

## 32.2 Volume Size

Specify the desired size for each volume in bytes. You can also use units like Gigabytes, Megabytes, or Kilobytes for convenience.

## 32.3 Output Filenames

Paquet Builder uses 7-Zip’s naming convention for multi-volume archives:

If the **Package Filename** is `myarchive.exe`, the generated volumes will be named:

- `myarchive.exe`
- `myarchive.exe.001`
- `myarchive.exe.002`
- `myarchive.exe.003`

If the package type is set to **Archive File**, the generated 7z multi-volume archive will have filenames like:

- `myarchive.7z.001`
- `myarchive.7z.002`
- `myarchive.7z.003`

## 32.4 Extracting a Multi-volume Archive

### Important

To successfully extract a multi-volume archive, all volume files must be located in the same folder as the main EXE file. If a volume is missing, the extraction will fail, and the package will exit with the following error message:

*“An unexpected error occurred while unpacking file(s). Error code: -5”*

This error occurs because the package cannot reconstruct the original archive from incomplete volume files.

## 32.5 GSplit Advanced File Archiver

For more advanced file splitting options, consider using **GSplit**, our free file splitter software. GSplit provides versatile features for managing split files.

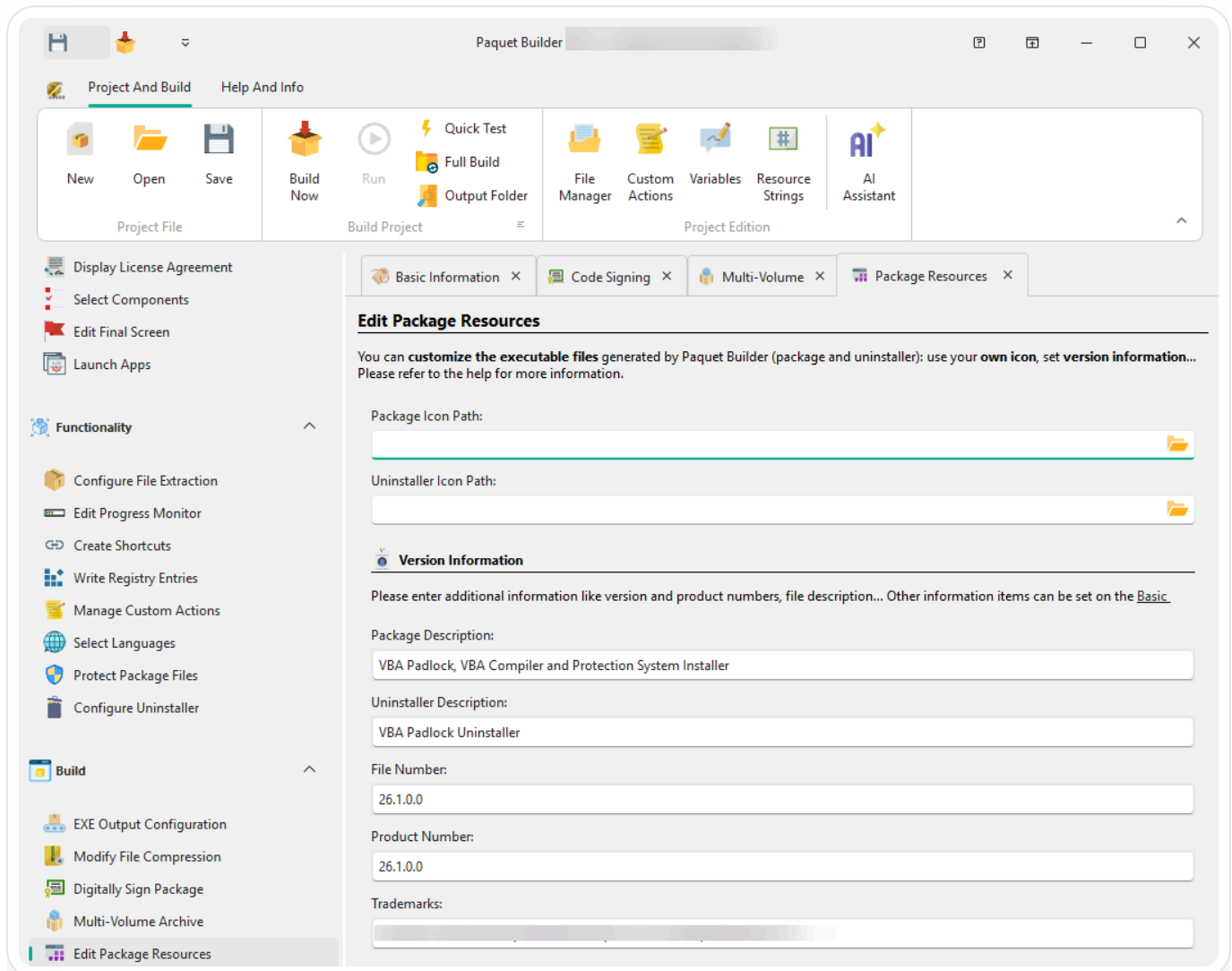
### Learn more about GSplit



Free file splitter software with advanced splitting options

## 33. Edit Package Resources

Paquet Builder generates a single executable file (PE format) with a **resource section** that stores metadata such as version information, icons, and string tables. This page lets you customize these resources.



### 33.1 Icons



#### Package Icon

Replace the default icon of the package EXE with your own `.ico` file. Supports icons of various sizes and color depths (32×32, 48×48, 256×256, etc.).



#### Uninstaller Icon

Replace the default icon of the **uninstaller** EXE file.

 **Tip**

If you need to create or extract icons, consider using **GConvert** to easily convert images into `.ico` format.

## 33.2 Version Information

Version information is embedded in the compiled file and visible when users right-click the EXE and select **Properties** (or press **Alt+Enter** in Explorer).

| Field                                   | Description   |
|---|---|
| <b>Package Description</b>              | Brief description of the package contents                 |
| <b>Uninstaller Description</b>          | Description for the uninstaller executable                |
| <b>File Number</b>                      | Release number in X.X.X.X format (e.g. 1.20.34.45)        |
| <b>Product Number</b>                   | Product version in X.X.X.X format (can match File Number) |
| <b>Trademarks</b>                       | Company trademarks (leave blank to exclude)               |
| <b>Copyright, Company, Product Name</b> | Populated from <a href="#">About Your Package</a>         |

## 33.3 Related Pages

### About Your Package →

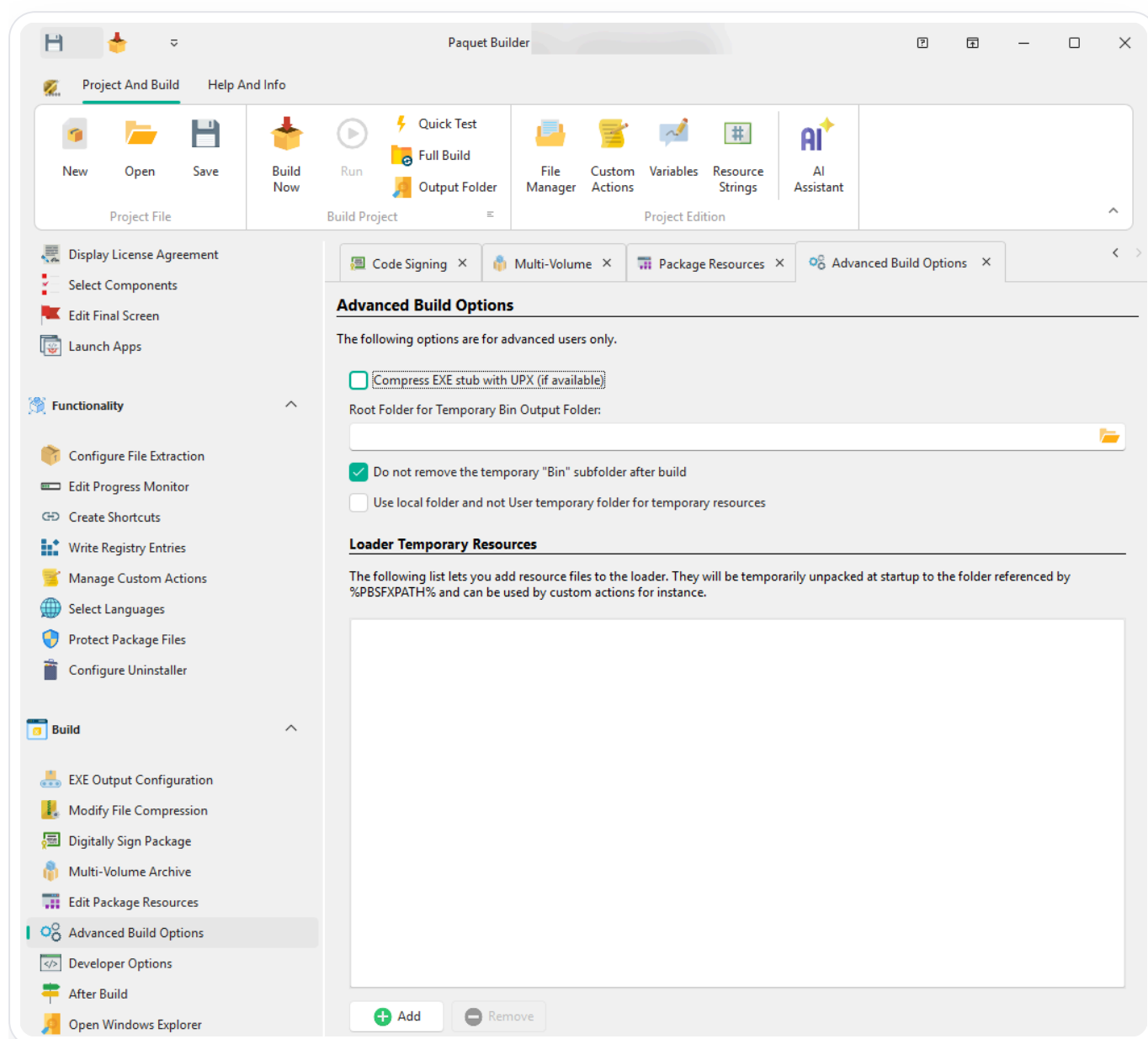
Set the package title and company metadata.

### Digital Signature →

Sign the EXE for authenticity and trust.

## 34. Advanced Build Options

This page covers advanced settings that control how Paquet Builder compiles packages and how the package behaves at startup on the end user's machine.



### 34.1 Compress EXE Stub

Paquet Builder generates EXE files with **small overhead** (under 256 KB). You can further reduce it with an executable packer:

**UPX** is a free executable packer that supports x86, x64, and ARM64 targets. Place the extracted files in the `compiler\UPX` subfolder of Paquet Builder.

### Note

UPX is not shipped with Paquet Builder for legal reasons.

### Caution

Some antivirus programs may flag compressed EXE files. [Code signing](#) often resolves this. Test with [VirusTotal](#).

## 34.2 Build Folder Options



### Temporary Bin folder

During compilation, Paquet Builder stores intermediate files in a “Bin” folder. If the output directory is a network share, specify a **local folder** here to prevent build failures.



### Keep Bin folder

By default, the “Bin” folder is deleted after compilation. Enable this to retain it for debugging.



### Local temporary resources

When enabled, runtime temporary resources stay in the same directory as the package (e.g. USB stick, network share) instead of the user’s temp folder. The path is stored in `%PBSFXPATH%`.

### Caution

When using local temporary resources, ensure the directory is writable — otherwise the package will fail.

## 34.3 Loader Temporary Resources

Include additional files in the package’s loader. These are compressed into the EXE overhead and extracted at startup to the temporary folder (accessible via `%PBSFXPATH%`).

Use this for custom DLL files or resources required by [custom actions](#).

## 34.4 Related Pages

## Package Output →

Set the output directory and filename.

## Digital Signature →

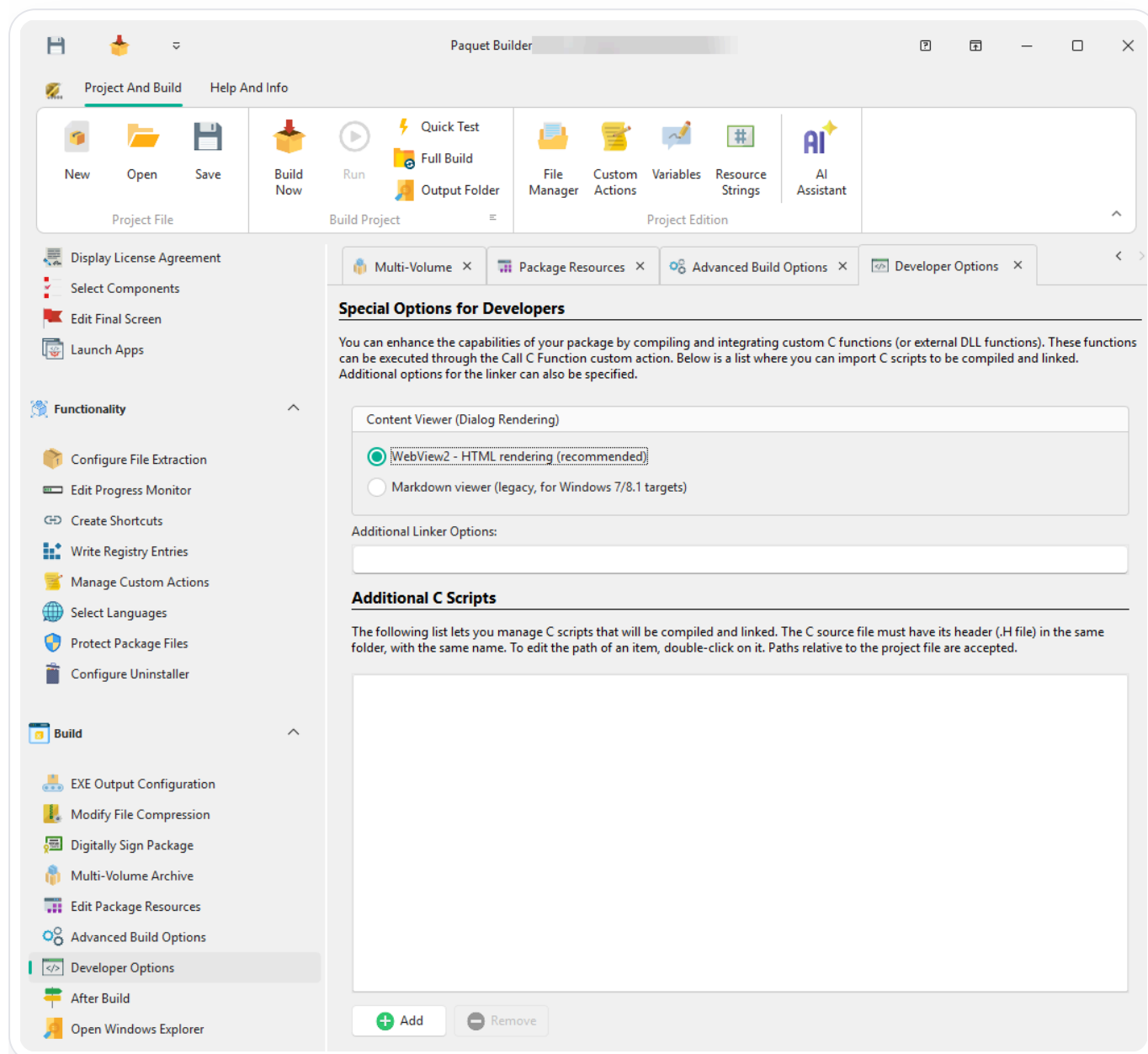
Sign the EXE to avoid antivirus false positives.

## Variable List →

Reference %PBSFXPATH% and other runtime variables.

## 35. Developer Options

Paquet Builder includes a **C compiler** to generate its EXE stubs, enabling developers to extend package functionality with custom C code and plugins.



### 35.1 Content Viewer (Dialog Rendering)

**HTML rendering** via Microsoft WebView2. Recommended for modern systems — provides full HTML/CSS support for dialog content.

Legacy rendering for Windows 7/8.1 targets. Use only when backward compatibility is required.

## 35.2 Additional C Scripts

Add one or more C source files to your project. They are compiled and linked into the final EXE during package compilation, and their functions can be invoked via **Call C Function** custom actions.



### Compiled twice

C source files are compiled once for the installer and once for the uninstaller.



### Header required

Each `.c` file must have a corresponding `.h` header file in the same folder.



### Linker options

Specify additional linker options to import third-party DLLs. Place `.lib` files in the `Compiler\Lib` subfolder.

### Note

Sample C scripts are available in the `customccode` subfolder of Paquet Builder.

### 35.2.1 Core C Functions

Include `pbcore.h` to access built-in functions:

```
// Set the value of a variable
SetVar(const wchar_t *name, const wchar_t *val);

// Get the value of a variable
int EvalVarBuf(const wchar_t *str, wchar_t *outBuffer, int outBufferSize);
```

## 35.3 Tutorial: Call a C Function

- 1 Navigate to **Build > Developer Options**, click **Add**, and select a C source file (e.g. `simplehello.c` from the `customccode` subfolder).
- 2 Open the **Custom Action Editor** (click **Custom Actions** in the toolbar), select an event (e.g. "After Welcome Screen"), and click **Add Action**.
- 3 Choose **Call a C function**, then select the imported C file.

- 4 In the “Code to invoke” field, enter: `ShowMyMessage()`;
- 5 Compile and test — the function displays a custom message box.

## 35.4 Related Pages

### Call C Function →

Custom action reference for invoking C functions.

### Custom Actions →

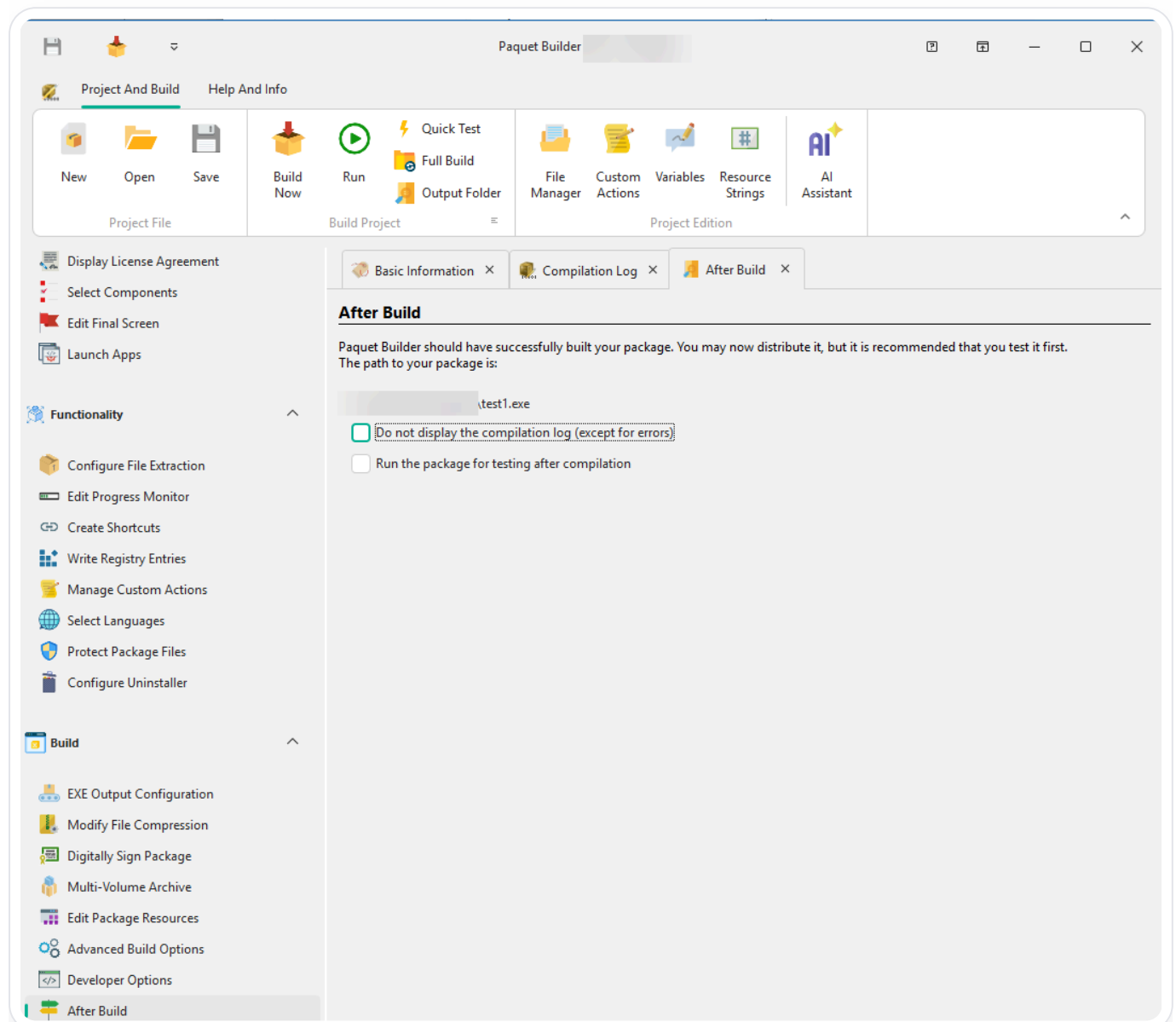
Add logic and scripted behavior to the installation flow.

### Advanced Build Options →

Configure temporary folders and loader resources.

## 36. After Build

This page provides options to **test and manage** your compiled package immediately after a successful build.



### 36.1 Post-Build Options

#### Package path

Displays the full path to your compiled package. Copy it or navigate to the output folder directly.

### Hide compilation log

Enable **“Do not display the compilation log”** to skip the log view after build (errors are always shown).

### Auto-run after build

Enable **“Run the package for testing after compilation”** to launch the package immediately for quick testing.

### Tip

Testing your package before distribution is strongly recommended. Use **Quick Test** in the toolbar for fast iterations without full compression.

## 36.2 Splitting Large Packages

For large packages that need to be split into smaller files for distribution, use **GSplit**, a free file splitter tool.

- 1 **Compile** your package with Paquet Builder.
- 2 **Split** the package into smaller pieces with GSplit.
- 3 **Create** a Self-Uniting executable that automatically restores and runs the package.
- 4 **Distribute** the smaller files via email, USB, or web.

## 36.3 Related Pages

### Package Output

Set the output directory, filename, and EXE bitness.

### Multi-Volume Archive

Built-in volume splitting for archives larger than 4 GB.

### Compression Options

Tune compression level and archive settings.

## 37. How to Sign Installers with SignTool and Paquet Builder

---

To ensure users trust the authenticity of your package, you can **digitally sign it** using **Microsoft Authenticode** thanks to Paquet Builder. This guarantees that the package has not been modified since its release. Paquet Builder can handle signing your installer for you, either with its built-in utility **GSignCode** or, for more advanced use cases aimed at experienced users, **Microsoft SignTool**.

Below, we describe the procedure to use **SignTool**. If you do not require advanced features, please refer to the topic [Adding a Digital Signature](#).

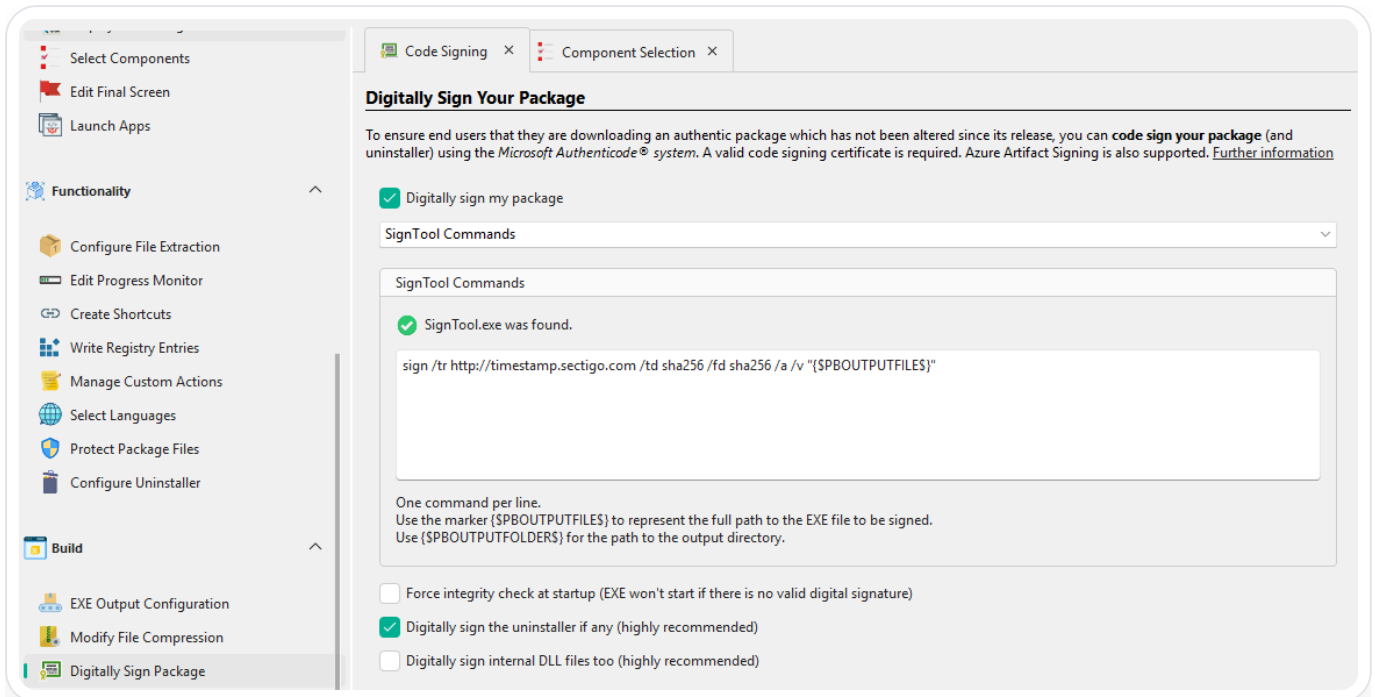
### 37.1 1. Prerequisites

Before using **SignTool**, ensure that:

- You have a **valid code signing certificate** installed in your system.
- You have installed the **Windows SDK**, which includes **SignTool.exe**.
  - Download the latest Windows SDK from the official Microsoft website: [Windows SDK Download](#)

### 37.2 2. Enabling Digital Signing in Paquet Builder

- 1 Open **Paquet Builder** and navigate to the **Digital sign package** page.
- 2 Check the box **"Digitally sign my package"**.
- 3 Configure the **SignTool Commands** by specifying the appropriate parameters.



## 37.3 3. Understanding SignTool Parameters in Paquet Builder

Paquet Builder allows you to dynamically reference the installer file and output directory using **placeholders**. This ensures flexibility and reduces the need for manual path configuration.

### 37.3.1 Example SignTool Command in Paquet Builder

```
sign /a /fd SHA256 /tr "http://timestamp.digicert.com" /td SHA256 "{SPBOUTPUTFILE$}"
```

#### Caution

Do not add signtool.exe nor its path: Paquet Builder automatically deals with it.

### 37.3.2 Breakdown of Parameters and Placeholders

| Parameter | Description   |
|-----------|---|
| /a        | Automatically selects the best certificate available in the system's certificate store. |

| Parameter                              | Description  |
|--|--|
| /fd SHA256                             | Specifies the file digest algorithm ( <b>SHA-256</b> is recommended).      |
| /tr<br>"http://timestamp.digicert.com" | Defines the timestamp server for RFC 3161 timestamping.                    |
| /td SHA256                             | Specifies the digest algorithm for the timestamp.                          |
| {%PBOUTPUTFILE%}                       | Automatically inserts the full path to the installer file(s) to be signed. |

### 37.3.3 Available Placeholders in Paquet Builder

| Placeholder        | Description   |
|--------------------|---|
| {%PBOUTPUTFILE%}   | Represents the full path to the executable file(s) to be signed.                  |
| {%PBOUTPUTFOLDER%} | Represents the path to the output directory where the signed file will be stored. |

These placeholders allow **Paquet Builder** to dynamically insert the correct file paths during the signing process, eliminating the need to hardcode file locations.

## 37.4 4. Signing Your Installer Package

Once the **SignTool Commands** are configured, click **Build** in **Paquet Builder**. The tool will automatically invoke **SignTool** to sign the generated executable.

### Tip

If you encounter signing errors, check that:

- Your **certificate** is installed in the system's certificate store.
- The **timestamp server** is reachable.

## 37.5 5. Alternative: Signing with a Specific Certificate

If you need to sign with a specific certificate (instead of letting SignTool choose automatically), you can use:

```
sign /fd SHA256 /sha1 ABCD1234... /tr "http://timestamp.digicert.com" /td  
SHA256 {$PBOUTPUTFILE$}
```

- `/sha1 ABCD1234...` — Selects a certificate using its SHA-1 fingerprint.
- `/n "My Certificate Name"` — Selects a certificate by its common name.

For more details, refer to the official [SignTool Documentation](#).

## 37.6 Multiple Digital Signatures in Paquet Builder

In **Paquet Builder**, it is possible to execute multiple **SignTool** commands sequentially by entering them **line by line** in the designated field. Each command will be executed in order, allowing you to apply multiple signatures, such as adding both **SHA-1 and SHA-256 signatures** to your installer.

### 37.6.4 Example: Dual SHA-1 and SHA-256 Signatures

To sign an installer first with **SHA-1** and then with **SHA-256**, enter the following commands **on separate lines** in the SignTool Commands field:

```
sign /a /fd SHA1 /tr "http://timestamp.digicert.com" /td SHA1 "  
{PBOUTPUTFILE$}"  
sign /a /fd SHA256 /tr "http://timestamp.digicert.com" /td SHA256 "  
{PBOUTPUTFILE$}"
```

### 37.6.5 Explanation

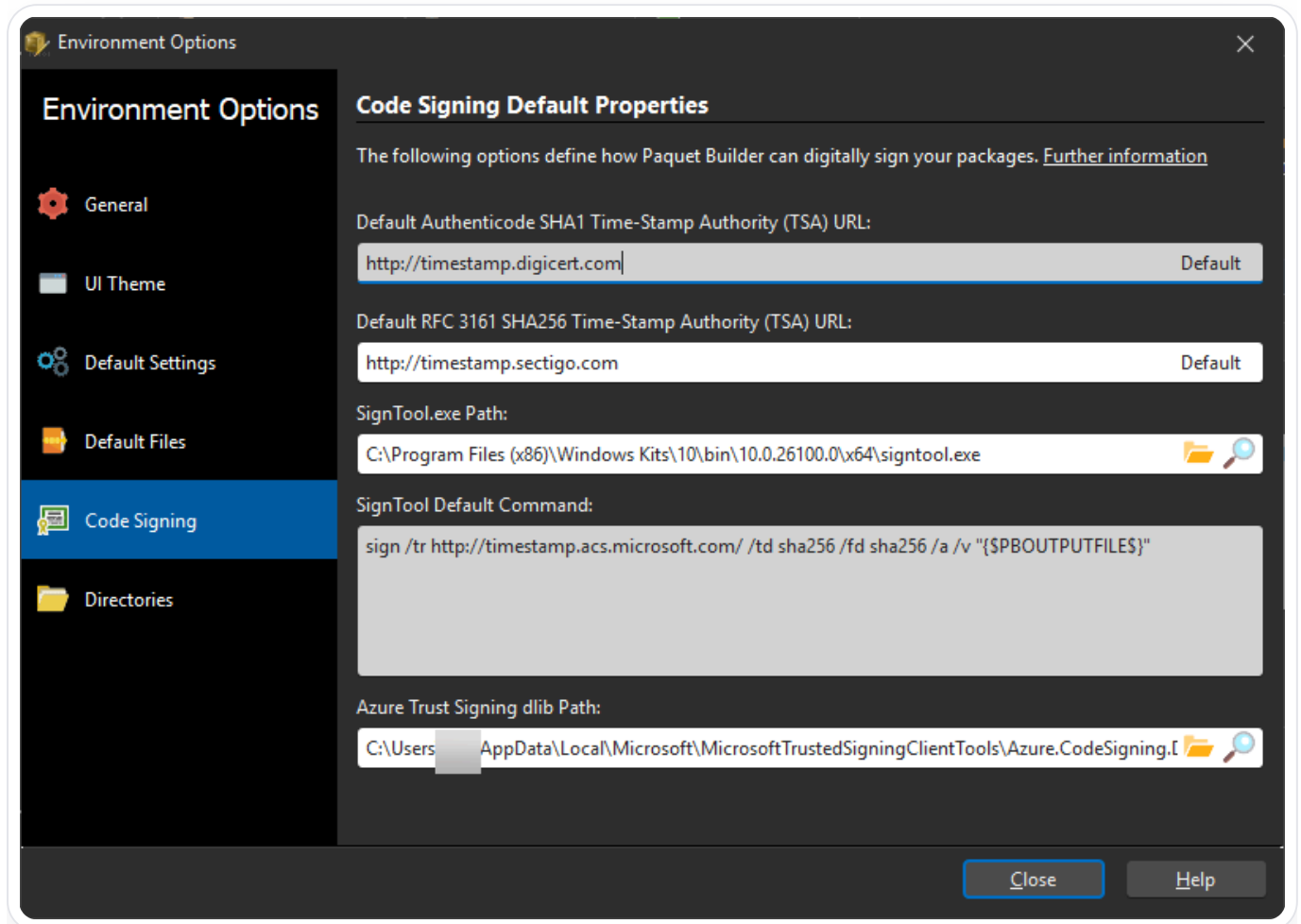
1. The first command signs the installer using **SHA-1** (useful for backward compatibility with older systems).
2. The second command **adds an additional SHA-256 signature**, ensuring compliance with modern security standards.

By entering each command on a new line, Paquet Builder will execute them one after the other automatically.

## 37.7 Defining the Path to SignTool in Paquet Builder

Before using **SignTool**, you must ensure that its path is correctly set in **Paquet Builder**.

In **Paquet Builder**, go to: **Menu** → **Environment Options** → **Code Signing**



Here, you will find the field where you need to specify the path to **SignTool.exe**.

### Automatically Detecting SignTool

- Click on the **magnifying glass** button.
- **Paquet Builder** will attempt to locate **SignTool.exe** automatically by checking the Windows registry.

### Manually Setting the Path

If **Paquet Builder** cannot find **SignTool**, you will need to manually enter the full path to **SignTool.exe**.

- Typically, SignTool is located in:

```
C:\Program Files (x86)\Windows Kits\10\bin\X.X.X.X\x64\signtool.exe
```

 **Tip**

If you have multiple versions of the Windows SDK installed, ensure you select the correct one that contains **SignTool.exe**.

## 37.8 Related Topics

### Digital Signature Overview [→](#)

General code signing guide

### Signing with JSign [→](#)

Free, open-source alternative with cloud KMS support (Azure Key Vault, AWS KMS, Google Cloud KMS)

### Azure Artifact Signing [→](#)

Microsoft's cloud-based signing service

### Environment Options [→](#)

Configure SignTool path and timestamp servers

### About Directive Files [→](#)

Automate signing with [SignToolCmd] section

# 38. How to Sign Installers with Azure Artifact Signing in Paquet Builder

## 38.1 1. Introduction

**Azure Artifact Signing** is a cloud-based code signing solution by Microsoft that eliminates the need for **local code signing certificates**. Instead, it securely signs executables through Microsoft's artifact signing service, providing enhanced security and ease of use.

Using **Azure Artifact Signing** in **Paquet Builder** ensures a streamlined and automated code signing process directly within your installer build workflow.







## 38.2 2. Why Choose Azure Artifact Signing?

Azure Artifact Signing offers several key benefits over traditional certificate-based signing:

- **Lower Cost:** Traditional certificates can be expensive (hundreds of USD), while Azure Artifact Signing provides a **pay-as-you-go model**, making it more affordable.

### Choose pricing tier

Browse the available plans and their features

|   |   |
|---|---|
| <b>Basic</b>  | <b>Premium</b>  |
|  1 certificate profile of each type    |  10 certificate profiles of each type  |
|  5k signatures/month quota             |  100k signatures/month quota           |
|  0.005 USD/signature above quota limit |  0.005 USD/signature above quota limit |
| <b>9.99</b><br>USD/Month  | <b>99.99</b><br>USD/Month   |

- **No Need for USB Tokens:** Standard certificates require a **hardware security module (HSM) or USB token** for private key storage, which can be cumbersome. Azure Artifact Signing eliminates this requirement.
- **Cloud-Based Security:** Signing keys are managed **entirely in the cloud**, preventing theft or loss and reducing security risks.

- **Microsoft Trust and Compliance:** As a **Microsoft-backed solution**, Azure Artifact Signing meets industry security standards while ensuring broad compatibility.

### 38.3 3. Prerequisites

Before signing with **Azure Artifact Signing**, you need:

- An **Artifact Signing account** with identity validation and a certificate profile.
- A **role assignment:** The **Artifact Signing Certificate Profile Signer** role must be assigned to you or your team.
- A **compatible Windows version:** Windows 11, Server 2016, or **10 (1809/October 2018 Update or newer)**

Official Microsoft Documentation on Prerequisites: [Azure Artifact Signing Integration Guide](#)

#### Note

See our tutorial about [how to Create an Azure Artifact Signing Account - Step-by-Step Guide](#)

### 38.4 4. Required Installations

To use Azure Artifact Signing with Paquet Builder, install the following tools:

#### 38.4.1 Artifact Signing Client Tools Installer

This tool enables the integration of Azure Artifact Signing with SignTool.

Download the MSI Installer: [Artifact Signing Client Tools](#)

#### 38.4.2 Azure CLI for Windows

Azure CLI is required to authenticate your account with Azure Artifact Signing.

Installation Guide: [Install Azure CLI for Windows](#)

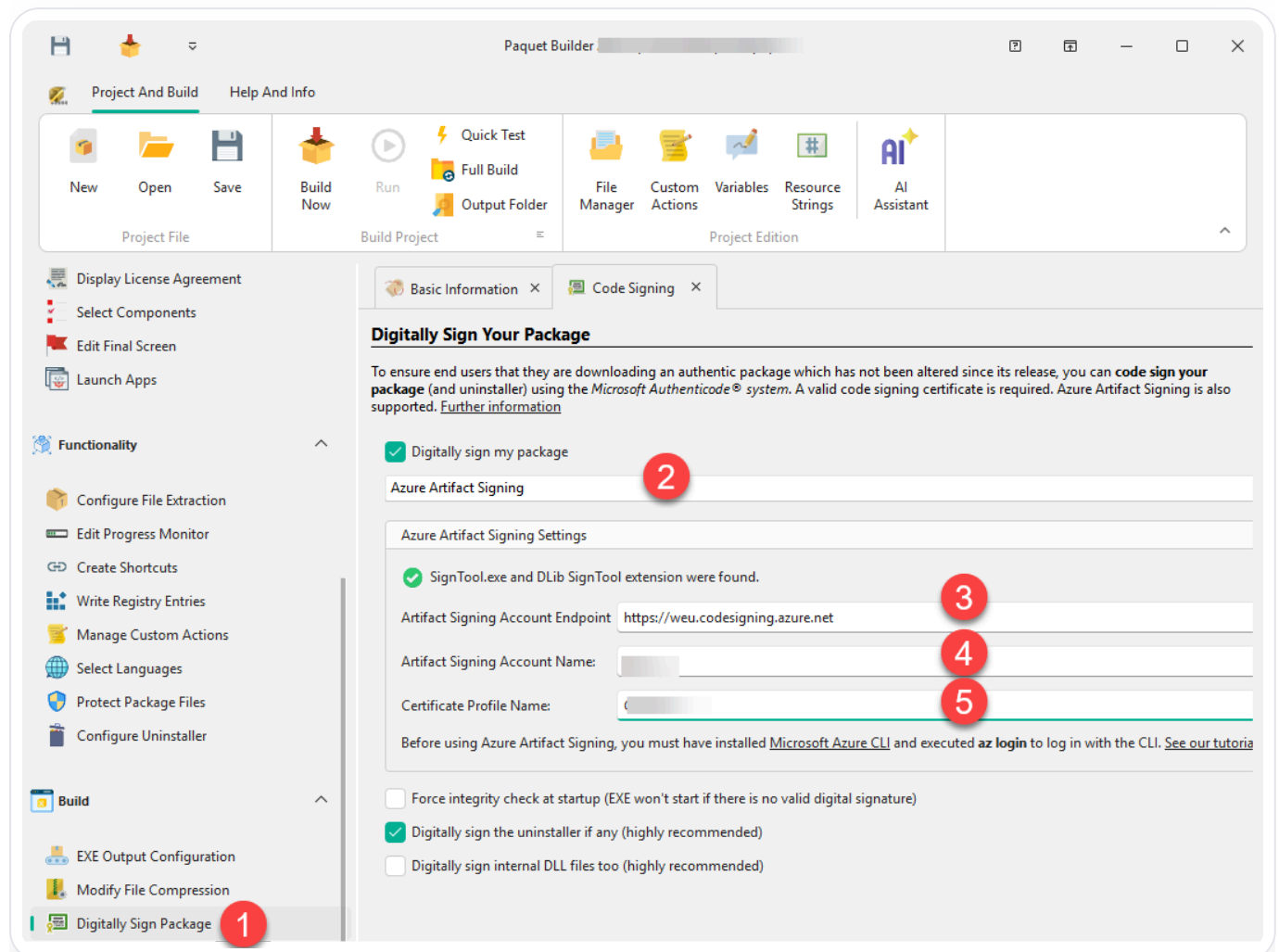
Direct MSI (64-bit) Download: [Azure CLI MSI Installer](#)

##### Why Install Azure CLI?

- Used for **logging into Azure** before signing packages.
- Required for **managing Azure credentials** securely.
- Allows **automated authentication** when using Azure services with Paquet Builder for code signing.

## 38.5 5. Configuring Azure Artifact Signing in Paquet Builder

Once everything is installed, you can configure **Paquet Builder** for **Azure Artifact Signing**:



- 1 **Open Paquet Builder**
- 2 **Go to the Digitally Sign Package page**
- 3 **Select Azure Artifact Signing from the dropdown**

## 4 Enter the required fields:

### o Artifact Signing Account Endpoint

Enter the **Artifact Signing Account Endpoint** corresponding to your Azure region. Use the following table to find the correct endpoint:

| Region          | Region Class  | Endpoint URI                       |
|-----------------|---------------|------------------------------------|
| East US         | EastUS        | https://eus.codesigning.azure.net  |
| West US 3       | WestUS3       | https://wus3.codesigning.azure.net |
| West Central US | WestCentralUS | https://wcus.codesigning.azure.net |
| West US 2       | WestUS2       | https://wus2.codesigning.azure.net |
| North Europe    | NorthEurope   | https://neu.codesigning.azure.net  |
| West Europe     | WestEurope    | https://weu.codesigning.azure.net  |

Make sure to select the endpoint matching the region where your Azure Artifact Signing account was created.

### o Artifact Signing Account Name (your Azure account name)

Home

Artifact Signing Accounts ☆ ... List recently modified Trusted Signing Accounts using ARG. What other details can I retrieve about Trusted Signing Accounts using ARG? Identify non-compliant Trusted Signing Accounts in my envi

+ Create Manage view Refresh Export to CSV Open query Assign tags Delete Add to service group

Filter for any field... Subscription equals all Resource Group equals all Location equals all Add filter

| Name ? | Type                     |
|--------|--------------------------|
| g      | Artifact Signing Account |

### o Certificate Profile Name (your signing certificate profile)

Microsoft Azure

Home > Artifact Signing Accounts

Certificate Profiles ...

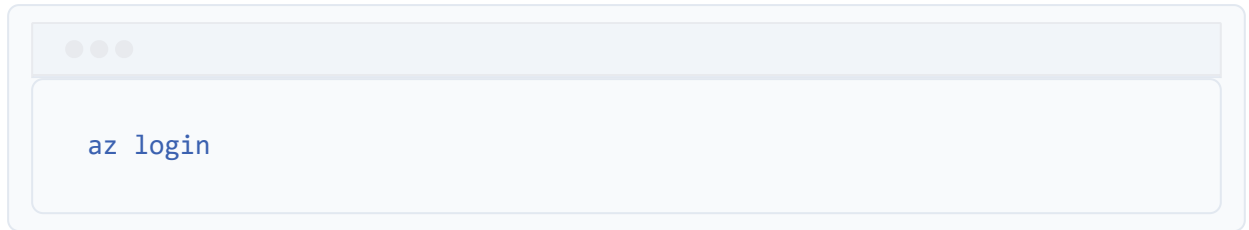
+ Create Refresh Delete

Search any field... Type: All Status: All

| Name | Subject name | Profile type   | Expiry date | Status |
|------|--------------|----------------|-------------|--------|
|      | CN=          | Public Trust   | 4/          | Active |
| Test | CN=          | Public Trus... | 4/          | Active |

5

**Before signing, authenticate in Azure CLI:** Run the following command in **PowerShell** or **Command Prompt**:

A terminal window with a light blue header and three small circles on the left. The text 'az login' is displayed in a monospaced font within the terminal area.

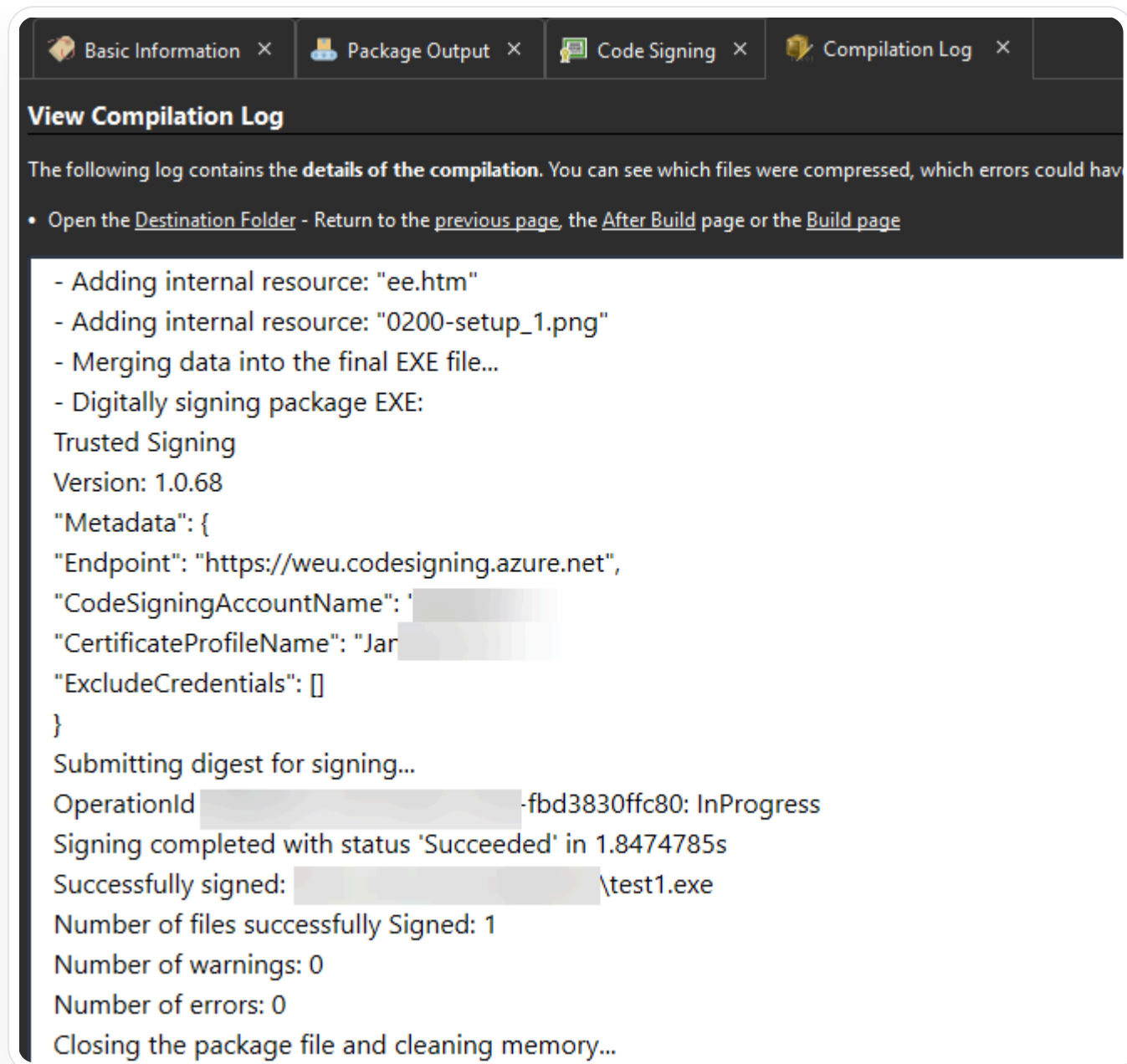
```
az login
```

This will open a **browser window** to authenticate your **Azure account**.

### **Caution**

Your Azure account must have the Artifact Signing Certificate Profile Signer role. You can verify that in the Access control (IAM) > Role assignments page of your Artifact Signing account.

Once this is configured, Paquet Builder will automatically invoke the Microsoft SignTool utility to sign your installer EXE file. All signing operations are listed in the compilation log:



## 38.6 Configuring SignTool and Azure Artifact Signing Paths in Paquet Builder

To ensure that **Paquet Builder** can correctly sign your installer packages, you must configure the paths to **SignTool.exe** and its **Azure Artifact Signing extension (dlib)** in the **Environment Options**. Paquet Builder should automatically do that.

### 38.6.3 Setting the Path to SignTool.exe

- Navigate to **Environment Options** → **Code Signing**.
- Locate the **SignTool.exe Path** field.

- **Click on the magnifying glass** to let Paquet Builder automatically detect the installed **SignTool.exe**.
- If the path is not found, enter it manually. The default installation location is:

```
C:\Program Files (x86)\Windows  
Kits\10\bin\10.0.26100.0\x64\signtool.exe
```

### Tip

Ensure that the Windows SDK is installed so that SignTool.exe is available.

## 38.6.4 Setting the Path to Azure Artifact Signing dlib

If you are using **Azure Artifact Signing**, you must configure the **Artifact Signing dlib Path** field. The Azure.CodeSigning.Dlib.dll file is installed by the Artifact Signing Client Tools Installer (see above).

- Locate the **Artifact Signing dlib Path** field.
- **Click on the magnifying glass** to let Paquet Builder automatically find the extension.
- If it is not found, enter the path manually:

```
C:\Users\LOGIN  
NAME\AppData\Local\Microsoft\MicrosoftArtifactSigningClientTools\Azure.CodeSign:
```

Replace `LOGIN NAME` with your actual Windows username.

### Note

- If the paths are not detected automatically, you must enter them manually.
- Ensure that the Artifact Signing Client Tools are installed before configuring the dlib path.
- Once the correct paths are set, Paquet Builder will automatically invoke SignTool in the background when signing your packages.

## 38.7 Related Topics

## Digital Signature Overview →

General code signing guide for Paquet Builder

## Signing with JSign →

Free, open-source alternative supporting Azure Key Vault, AWS KMS, and Google Cloud KMS

## SignTool Commands →

Custom signing with Microsoft SignTool

## About Directive Files →

Automate Azure Artifact Signing in build scripts

## 39. How to Sign Installers with JSign in Paquet Builder

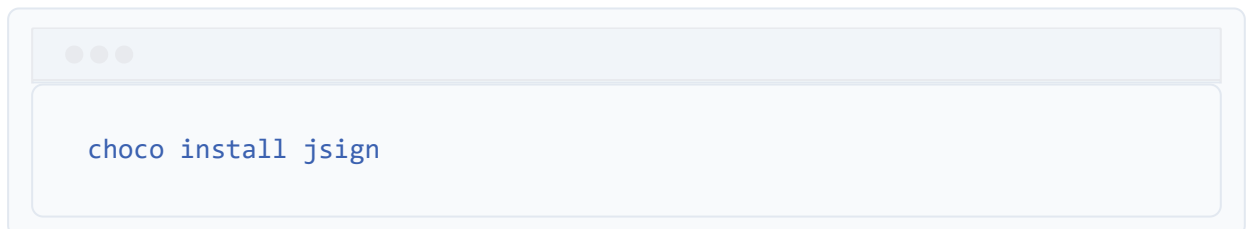
**JSign** is a **free, open-source**, Java-based code signing tool that supports a wide variety of keystores and cloud signing services:

- **PFX / PKCS#12** certificate files
- **Hardware tokens** (ETOKEN, YUBIKEY, and other PKCS#11 devices)
- **Azure Key Vault**
- **AWS KMS** (Key Management Service)
- **Google Cloud KMS**
- **HashiCorp Vault**
- And **many more**

JSign is an excellent alternative when you need cloud-based key management or when the built-in signing tools (GSignCode, SignTool) don't support your keystore type.

### 39.1 Prerequisites

- 1 **Java** must be installed and accessible on your system PATH (`java.exe`)
- 2 **JSign JAR file** — install via **Chocolatey**:



```
choco install jsign
```

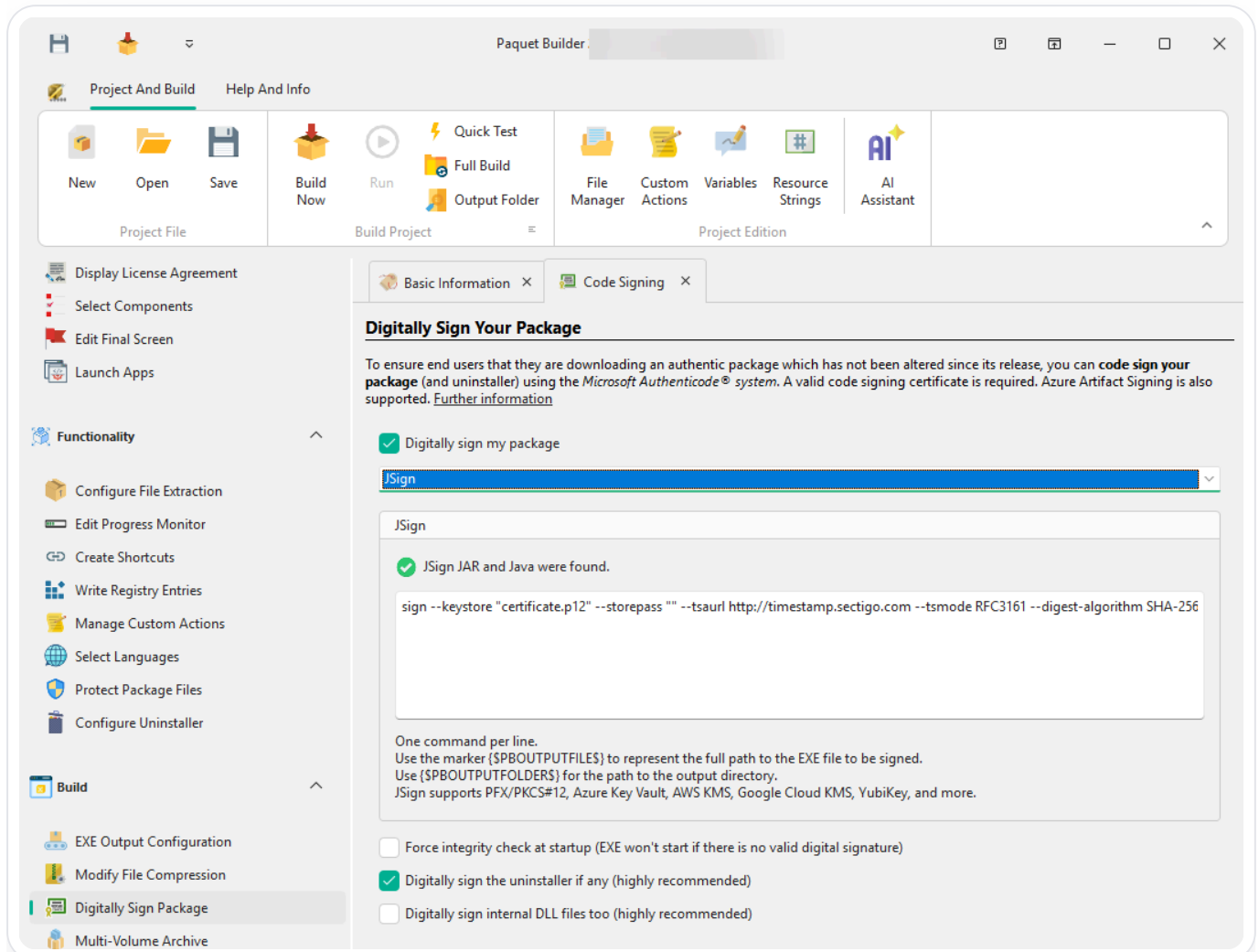
Or download the JAR file manually from the [JSign releases page](#).

- 3 Configure the **JSign JAR path** in Paquet Builder's **Environment Options** under the Code Signing section.

### 39.2 Configuring JSign in Paquet Builder

- 1 Open the **Digital Signature** settings page
- 2 Enable **"Digitally sign my package"**

- 3 In the **Certificate Location** dropdown, select **JSign** (option 5)
- 4 Enter your JSign signing command(s) in the command field



### 39.3 Command Placeholders

Use these placeholders in your JSign commands — they are replaced with actual values at build time:

| Placeholder            | Replaced With                             |
|------------------------|---|
| { \$PBOUTPUTFILE\$ }   | Full path to the output file being signed |
| { \$PBOUTPUTFOLDER\$ } | Path to the output folder                 |

### 39.4 Examples

```
sign --keystore "C:\certs\mycert.p12" --storepass "mypassword" --tsurl  
http://timestamp.sectigo.com --tsmode RFC3161 --digest-algorithm SHA-256 "  
{ $PBOUTPUTFILE$ }"
```

```
sign --storetype AZUREKEYVAULT --keystore "myvaultname" --storepass  
"client_id|client_secret|tenant_id" --alias "my-certificate" --tsurl  
http://timestamp.digicert.com --tsmode RFC3161 "{ $PBOUTPUTFILE$ }"
```

```
sign --storetype AWS --keystore "us-east-1" --alias "alias/my-signing-key"  
--certfile "C:\certs\chain.pem" --tsurl http://timestamp.digicert.com --  
tsmode RFC3161 "{ $PBOUTPUTFILE$ }"
```

```
sign --storetype GOOGLECLOUD --keystore "projects/my-  
project/locations/global/keyRings/my-keyring" --alias "my-  
key/cryptoKeyVersions/1" --certfile "C:\certs\chain.pem" --tsurl  
http://timestamp.digicert.com --tsmode RFC3161 "{ $PBOUTPUTFILE$ }"
```

```
sign --storetype YUBIKEY --storepass "123456" --alias "X.509 Certificate  
for Digital Signature" --tsurl http://timestamp.sectigo.com --tsmode  
RFC3161 "{ $PBOUTPUTFILE$ }"
```

## Tip

Always include the `--tsurl` parameter to add a timestamp to your digital signature. This ensures the signature remains valid even after the certificate expires.

## 39.5 Directive Support

When automating builds with **directive files**, set `SignCertifLocation=5` in the `[General]` section and provide your JSig commands in a `[JSigCmd]` section:

```
[General]
DigitalSign=1
SignCertifLocation=5

[JSigCmd]
0=sign --keystore "certificate.p12" --storepass "mypass" --tsurl
http://timestamp.sectigo.com --tsmode RFC3161 --digest-algorithm SHA-256 "
${PBOUTPUTFILE}"
```

Each line in `[JSigCmd]` is a separate signing command, executed in order. This allows applying multiple signatures if needed.

## 39.6 Comparison with Other Signing Methods

| Method                 | Cost        | Cloud KMS                    | Hardware Tokens | Notes                        |
|------------------------|-------------|------------------------------|-----------------|------------------------------|
| GSignCode (built-in)   | Free        | No                           | Yes             | Included with Paquet Builder |
| SignTool Commands      | Free        | No                           | Yes             | Requires Windows SDK         |
| Azure Artifact Signing | Pay-per-use | Azure only                   | No              | Microsoft's managed service  |
| <b>JSig</b>            | <b>Free</b> | <b>Yes</b> (Azure, AWS, GCP) | <b>Yes</b>      | Open-source, cross-platform  |

## Caution

JSign requires Java (`java.exe`) to be installed and accessible on your system PATH. If Java is not found, the signing step will fail during compilation. You can verify by running `java -version` in a terminal.

## 39.7 Related Topics

### Digital Signature Overview [→](#)

General code signing guide

### Azure Artifact Signing [→](#)

Microsoft's cloud signing service

### SignTool Commands [→](#)

Custom SignTool configuration

### About Directive Files [→](#)

Automate signing in builds

### Environment Options [→](#)

Configure JSign JAR path and timestamp servers

### Set Output EXE File Path [→](#)

Build settings and compilation

### Multi-volume Archive [→](#)

Note: Windows cannot sign files larger than 2 GB

## 40. Go Further

---

Once you have mastered the basic package flow, use these advanced features to build highly flexible, professional, and dynamic installers.

### Typical audience

These topics are essential for moving from simple data archives to full-featured application installers with conditional logic and polished presentation.

### 40.1 Core Logic & Variables

#### Working with Variables [→](#)

Create dynamic behavior by using system and custom variables throughout your project.

#### Global Variable Reference [→](#)

A complete list of built-in variables for paths, system info, and runtime states.

#### Resource Strings & Localization [→](#)

Centralize your project's text to easily support multiple languages.

### 40.2 Folder & Component Management

#### Organizing Files into Components [→](#)

Group related files together and allow users to select optional features.

#### Individual File Properties [→](#)

Override destination paths and set extraction conditions for specific files.

#### Automating Component Creation [→](#)

Use the Component Wizard to instantly mirror your disk folder structure.

### 40.3 System Integration & UI

## UAC and Elevation →

Configure Windows User Account Control manifests and permission requirements.

## Designing Rich Content →

Create professional Welcome and Readme screens with the built-in editor.

## Managing the Uninstaller →

Ensure a clean removal process by including a registered uninstaller engine.

# 41. Working with Variables

Variables are **dynamic containers** for system paths, user inputs, or runtime values. They make your installer adapt to different environments without hardcoding paths or settings.

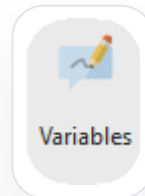
## 41.1 How Variables Work

Variables are identified by a name enclosed in % signs (e.g. %DESTPATH%). When the package runs, each identifier is replaced by its actual value.

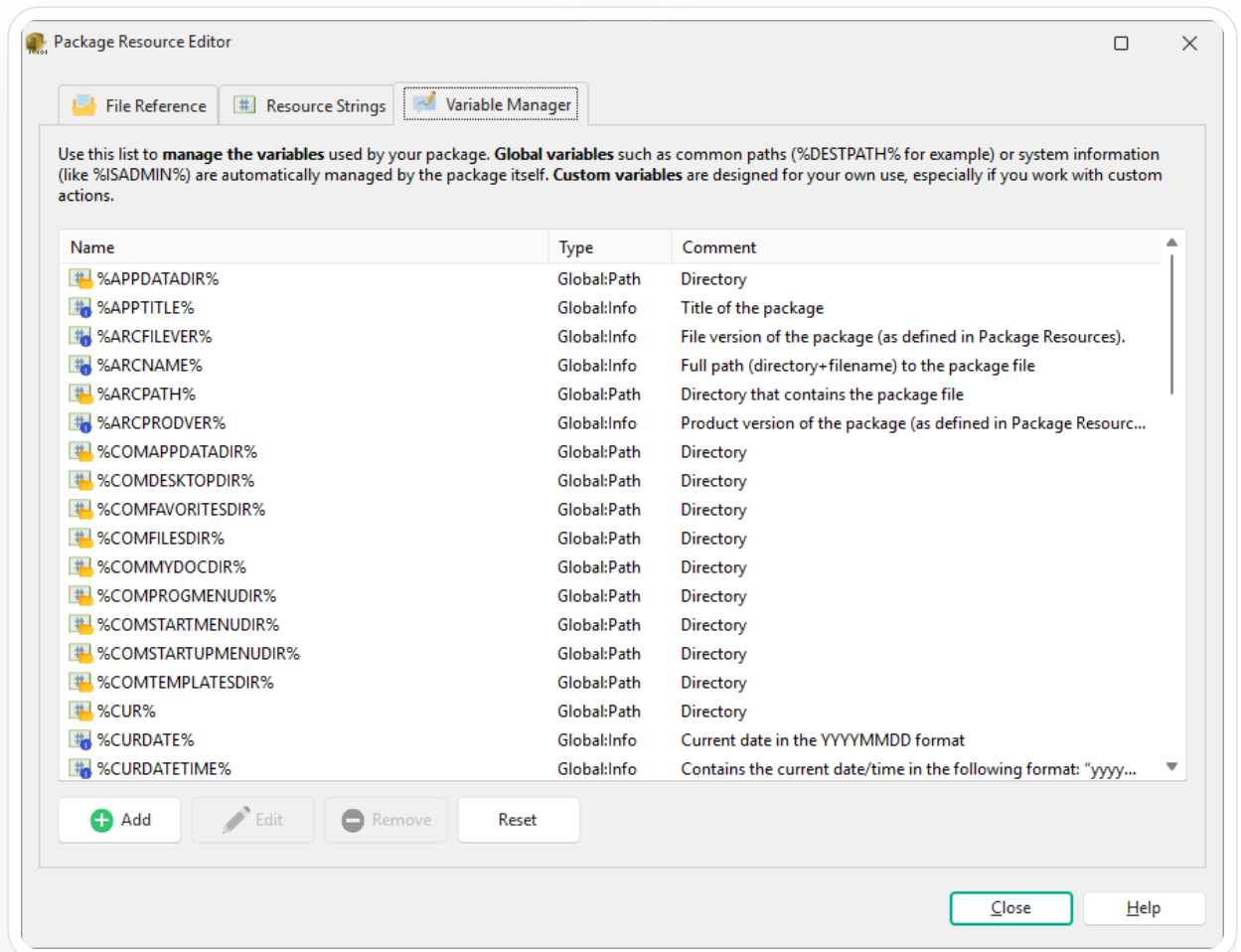
| Feature          | Requirement  |
|------------------|--|
| <b>Syntax</b>    | Must start and end with % (e.g. %MYVAR%)                                 |
| <b>Naming</b>    | Letters, numbers, and underscores only — no spaces or special characters |
| <b>Reserved</b>  | Names starting with PB are reserved for internal use                     |
| <b>Data Type</b> | Unicode text (max 4096 wide characters)                                  |
| <b>Booleans</b>  | Logic handled via 1 (True) and 0 (False)                                 |

## 41.2 Defining Custom Variables

- 1 Click the **Variables** button



in the toolbar to open the Variable Manager.



- 2 Click **Add** to create a new variable.
- 3 Configure the **Name**, **Type** (Path, String, or Boolean), and **Initial Value**.
- 4 Click **OK** to register the variable.

## 41.3 Built-in Variable Categories



### System Paths

Locations like %WIN%, %SYS%, %TMP%, %APPDATADIR%, and hundreds more — initialized automatically at runtime.



## Package Information

Details about the installer: %APPTITLE%, %ARCNAME%, %ARCFILEVER%, %ARCPRODVER%, etc.



## Installation Scope

Scope-aware paths that adapt to per-user or all-user installations: %PBINSTALLSCOPEDIR%, %PBSHORTCUTROOT%, %PBDESKTOPROOT%.



## Runtime State

Flags and values set during execution: %DESTPATH%, %ISADMIN%, %SILENT%, %TOTALEXTFILE%, etc.

## 41.4 Best Practices



### Trailing backslashes

Paquet Builder does **not** append trailing backslashes to path variables. Always include the backslash manually:

- **Correct:** %DESKTOPDIR%\MyFile.exe
- **Incorrect:** %DESKTOPDIR%MyFile.exe



### Tip

Use Boolean variables (1/0) in **Conditional Logic** blocks to toggle features or skip extraction steps.

## 41.5 Related Pages

### Global Variable Reference →

Complete list of all built-in variables.

### Modify Variable →

Change variable values at runtime with custom actions.

### Installation Scope →

Scope-aware variables for per-user and all-users installs.

## 42. Global Variable Reference

Global variables are pre-defined by Paquet Builder and initialized automatically at runtime. They provide essential information about the system environment, the logged-in user, and the package's own state.

### Note

For an introduction to how to use these identifiers, see [Working with Variables](#).

### 42.1 Profile Resolution

In the tables below, the **Profile** column indicates whether the variable points to a user-specific or system-wide (common) location:

- **User:** Points to the profile of the user currently running the package.
- **Common:** Points to the "All Users" profile.
- **Note:** On systems without administrator rights, "Common" variables often resolve to the same location as "User" variables.

### 42.2 Package and Archive Information

These variables provide details about the installer file and its versioning.

| Variable     | Description  |
|--------------|--|
| %APTITLE%    | The <a href="#">Package Title</a> defined in your project. |
| %ARCFILEVER% | The file version of the package (e.g., 1.2.0.0).           |
| %ARCPRODVER% | The product version of the package.                        |
| %ARCNAME%    | The full filename of the running self-extracting package.  |
| %ARCPATH%    | The directory path where the package is currently located. |

### 42.3 Core Session Paths

| Variable   | Description  |
|------------|--|
| %DESTPATH% | The final destination folder selected by the user. <b>Note:</b> This is only initialized after the file extraction process begins. |
| %TMP%      | Path to the current user's temporary folder.   |
| %CUR%      | Points to the current working directory of the package process.  |

## 42.4 User Profile Folders (Current User)

| Variable             | Description   | Profile |
|----------------------|---|---------|
| %APPDATADIR%         | Path to the Roaming Application Data folder.                  | User    |
| %LOCALAPPDATADIR%    | Path to the Local (non-roaming) Application Data folder.      | User    |
| %LOCALAPPDATALOWDIR% | Path to the LocalLow Application Data folder.                 | User    |
| %DESKTOPDIR%         | Path to the current user's Desktop.                           | User    |
| %MYDOCDIR%           | Path to the "My Documents" folder.                            | User    |
| %PROGMENUDIR%        | Path to the "Programs" folder on the Start Menu.              | User    |
| %STARTMENUDIR%       | Path to the top level of the Start Menu.                      | User    |
| %STARTUPMENUDIR%     | Path to the "Startup" folder on the Start Menu.               | User    |
| %FAVORITESDIR%       | Path to the Favorites folder.                                 | User    |
| %TEMPLATESDIR%       | Path to the Templates folder.                                 | User    |
| %QUICKLAUNCHDIR%     | Path to the Quick Launch folder.                              | User    |
| %SENDTOMENUDIR%      | Path to the "Send To" context menu folder.                    | User    |
| %USERPROFILE%        | Path to the root of the user's profile (e.g., C:\Users\John). | User    |

## 42.5 Common Folders (All Users)

| Variable             | Description   | Profile |
|----------------------|---|---------|
| %COMAPPDATADIR%      | Path to the common Application Data folder.             | Common  |
| %COMLOCALAPPDATADIR% | Path to the common Local Application Data folder.       | Common  |
| %COMDESKTOPDIR%      | Path to the common (shared) Desktop.                    | Common  |
| %COMMYDOCDIR%        | Path to the common "Documents" folder.                  | Common  |
| %COMPROGMENUDIR%     | Path to the common "Programs" folder on the Start Menu. | Common  |
| %COMSTARTMENUDIR%    | Path to the common Start Menu root.                     | Common  |
| %COMSTARTUPMENUDIR%  | Path to the common "Startup" folder.                    | Common  |
| %COMFAVORITESDIR%    | Path to the common Favorites folder.                    | Common  |
| %COMTEMPLATESDIR%    | Path to the common Templates folder.                    | Common  |

## 42.6 System Folders

| Variable        | Description                                       |
|-----------------|---|
| %WIN%           | Path to the Windows directory (e.g., C:\Windows). |
| %SYS%           | Path to the Windows System32 directory.           |
| %SYSROOT%       | The system drive (e.g., c:).                      |
| %PROGFILES_DIR% | Path to the "Program Files" directory.            |
| %COMFILES_DIR%  | Path to the "Common Files" directory.             |
| %FONTS_DIR%     | Path to the Windows Fonts folder.                 |

## 42.7 Operating System and Hardware

| Variable | Description  |
|----------|--|
| %IS64OS% | 1 if the OS is 64-bit (x64 or ARM64), 0 if 32-bit.                                 |
| %OSARCH% | Processor architecture: x86, x64, or ARM64.  |
| %OSVER%  | Integer representing the OS name (see the <a href="#">OS Version Table</a> below). |

## 42.8 Installation Scope

These variables are specifically useful when using the [Installation Scope](#) feature.

| Variable            | Description   |
|---------------------|---|
| %PBINSTALLSCOPE%    | 0 = Per-user installation, 1 = All-users installation.                      |
| %PBINSTALLSCOPEDIR% | Resolves to %LOCALAPPDATA%\Programs or %PROGFILES%\Programs based on scope. |
| %PBSHORTCUTROOT%    | Resolves to the user or common Start Menu based on scope.                   |
| %PBDESKTOPROOT%     | Resolves to the user or common Desktop based on scope.                      |
| %PBSCOPEELEVATED%   | 1 if the installer re-launched with UAC elevation, 0 otherwise.             |

## 42.9 Runtime State and Logic

| Variable      | Description   |
|---------------|---|
| %ISADMIN%     | 1 if the user has administrative rights, 0 otherwise.                                 |
| %ISPOWERUSER% | 1 if the user belongs to the Power Users group, 0 otherwise.                          |
| %NEEDREBOOT%  | 1 if a file replacement requires a system restart, 0 otherwise.                       |
| %SILENT%      | 1 if the installer was launched in silent mode, 0 otherwise. Can be manually toggled. |

| Variable         | Description  |
|------------------|--|
| %EXITCODE%       | The integer value returned by the package process upon closing.  |
| %CURDATE%        | Current date in yyyyMMdd format.   |
| %CURDATETIME%    | Current date and time in yyyyMMdd hh:mm:ss format.   |
| %TOTALEXTFILE%   | The total number of files successfully unpacked during this session.   |
| %SHORTCUTPATH%   | Path to the Start Menu folder selected by the user (when <b>prompted</b> ). Available after file extraction. |
| %PBSFXPATH%      | Path to the temporary folder where the <b>SFX overhead files</b> are extracted at startup.                   |
| %PBUNINSTREGKEY% | The unique registry key used by the <b>Package Uninstaller</b> to store uninstall information.               |
| %PBDLGNEXTC%     | Caption of the "Next" button on the current dialog page. Can be changed at runtime.                          |
| %PBLASTDLG%      | Returns the dialog page ID of the previous dialog shown to the user.   |

## 42.10 Command-line Parameters

| Variable           | Description  |
|--------------------|--|
| %PARAMCOUNT%       | The number of parameters passed to the installer.        |
| %PARAMS%           | The entire command-line string passed to the package.    |
| %PARAM%            | The first parameter (argument) in the command line.      |
| %PARAM2%, %PARAM3% | Specific parameters by position.                         |
| %PARAMx%           | The parameter at position x (from 2 up to %PARAMCOUNT%). |

## 42.11 Localization

| Variable   | Description  |
|------------|--|
| %PBLANGID% | The <b>Locale ID (LCID)</b> of the current display language. |
| %SELLANG%  | The display name of the selected language.                   |

## 42.12 OS Version Reference

The %OSVER% variable returns an integer corresponding to the Windows version:

| Value | Windows Version               |
|-------|-------------------------------|
| 0     | Unidentified (Future version) |
| 1     | Windows NT 3.5                |
| 2     | Windows NT 4.0                |
| 3     | Windows 2000                  |
| 4     | Windows XP                    |
| 5     | Windows XP 64-Bit             |
| 6     | Windows Server 2003           |
| 7     | Windows Home Server           |
| 8     | Windows Server 2003 R2        |
| 9     | Windows Storage Server 2003   |
| 10    | Windows Vista Starter         |
| 11    | Windows Vista                 |
| 12    | Windows Server 2008           |
| 13    | Windows 7                     |

| Value | Windows Version        |
|-------|------------------------|
| 14    | Windows Server 2008 R2 |
| 15    | Windows 8              |
| 16    | Windows 8.1            |
| 17    | Windows Server 2012    |
| 18    | Windows Server 2012 R2 |
| 19    | Windows Server 2016    |
| 20    | Windows 10             |
| 21    | Windows 11             |
| 22    | Windows Server 2022    |

### Tip

To retrieve the exact build number or feature update version (e.g., "22H2"), use the **Retrieve Windows Version Action Template**.

## 42.13 Special Formatting

### 42.13.1 Directory Slashing

#### Caution

All path variables are returned **without** a trailing backslash.

- **Correct:** %DESKTOPDIR%\App.exe
- **Incorrect:** %DESKTOPDIR%App.exe

### 42.13.2 How to insert a carriage-return line-feed

The character sequence for a carriage-return line-feed (CRLF) is `/r/n`.

For example, if you want a message box to display two lines:

```
Line1/$Line2
```

This will be rendered as:

```
Line1
```

```
Line2
```

## 42.14 Windows Known Folders

The variables above cover the most commonly used paths. However, if you need to access additional Windows paths that are not available as built-in variables, you can use the **Get Full Path of a Windows Known Folder** [action template](#) to retrieve any path identified by a `KNOWNFOLDERID` constant.

## 42.15 Regarding %ISADMIN% and %ISPOWERUSER%

The `%ISADMIN%` variable returns `1` if the current user has administrative rights, and `0` otherwise. The `%ISPOWERUSER%` variable returns `1` if the current user belongs to the Power Users group.

These variables are especially useful when you need to conditionally execute actions depending on the privilege level of the user running the package. For instance, you can use them in [conditional expressions](#) to show different messages or skip certain installation steps.

### Note

On modern versions of Windows (Vista and later) with UAC enabled, a user who is a member of the Administrators group will still have `%ISADMIN%` return `0` unless the package is running elevated.

## 42.16 See Also

- [Working with Variables](#)
- [How to get Windows Known Folders](#)
- [Installation Scope Paths](#)

## 43. Organizing Files into Components

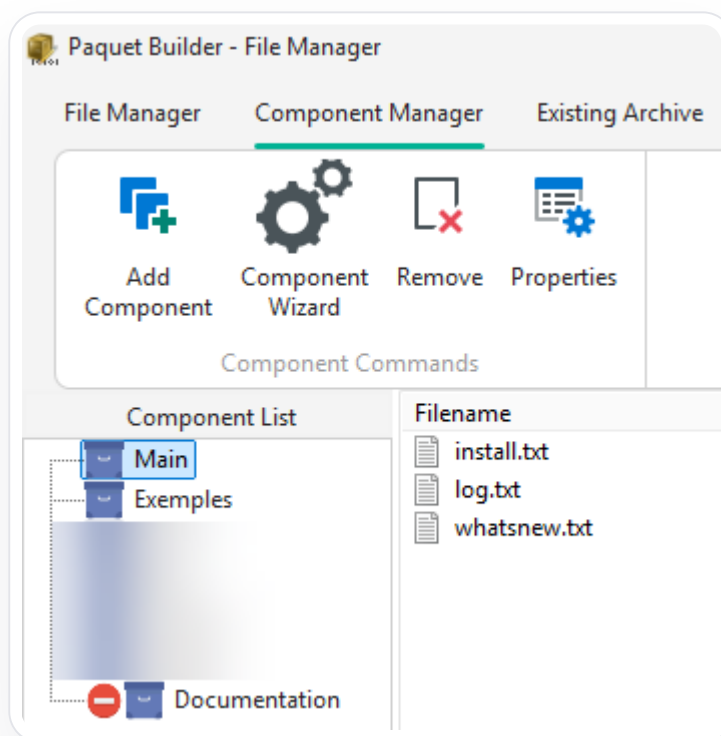
Components are **logical groups of files** within your package. They let you organize complex projects and give end users the flexibility to select optional features during installation.

### ⚠ Caution

Every project must contain at least one component named **Main**. This component cannot be renamed or deleted.

### 43.1 Component Management

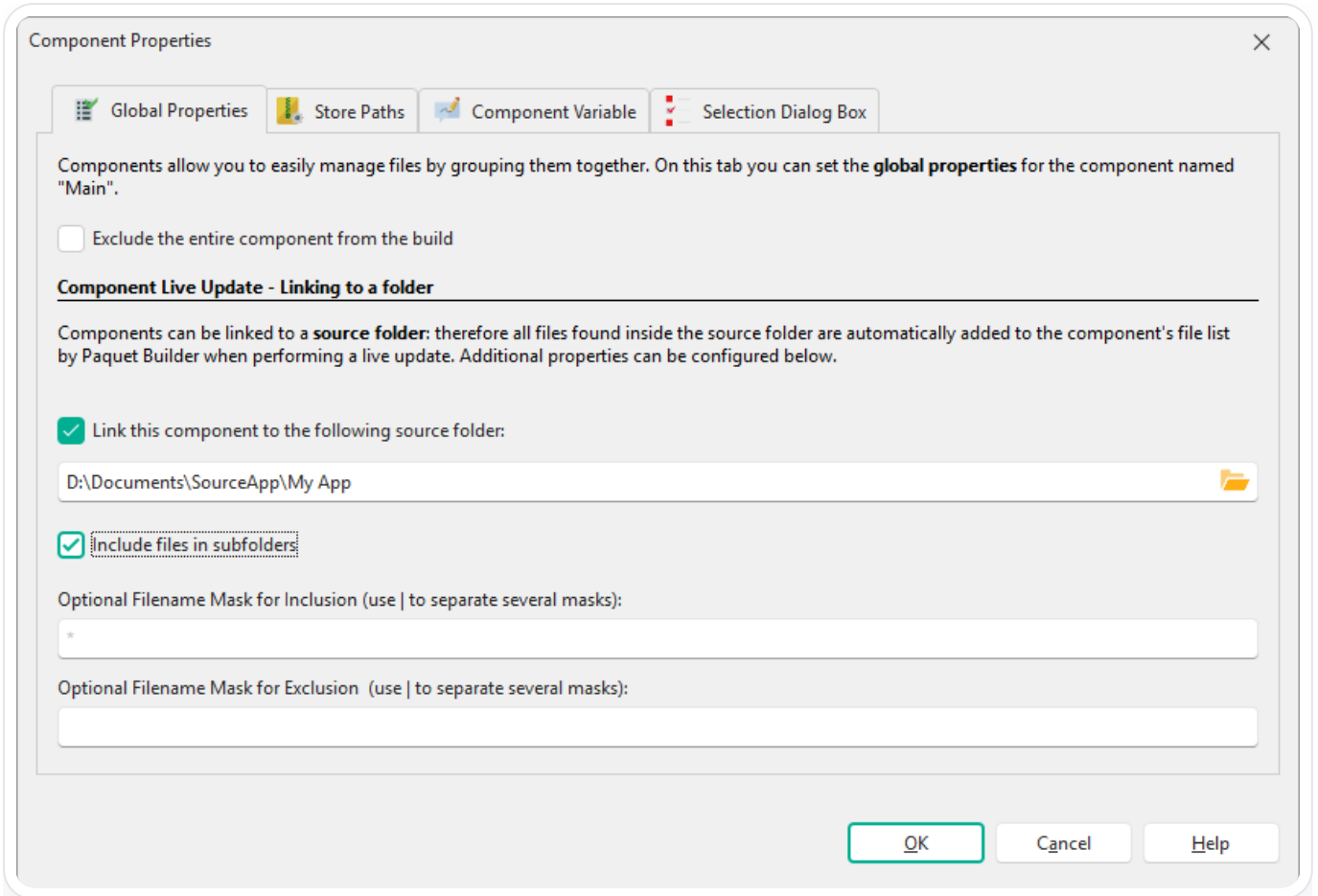
Manage components through the **Component Manager** within the File Manager.



- 1 Click **File Manager** in the toolbar, then choose **Component Manager**.
- 2 Use the ribbon to **Add**, **Remove**, or **Rename** components.
- 3 Double-click a component or click **Properties** to configure its behavior.

### 43.2 Component Properties

Double-click a component or click **Properties** to open the Component Properties dialog. It is organized into four tabs.



### 43.2.1 Global Properties

#### Exclude the entire component from the build

Prevents the component from being compiled into the package. Useful for managing editions (Trial vs. Full) or automated builds with **Directives**.

#### Live Update — Linking to a folder

Link the component to a **source folder** on disk. When a live update is performed, Paquet Builder scans the folder and automatically synchronizes the component's file list — adding new files and removing deleted ones.

| Setting   | Description   |
|---|---|
| <b>Link this component to the following source folder</b> | Enable the link and specify the source folder path.         |
| <b>Include files in subfolders</b>                        | When checked, files in nested subfolders are also included. |

| Setting                            | Description   |
|------------------------------------|---|
| <b>Filename Mask for Inclusion</b> | Only files matching this pattern are added (e.g. `*.exe`) |
| <b>Filename Mask for Exclusion</b> | Files matching this pattern are skipped (e.g. `*.pdb`)    |

### Tip

Right-click a component in the File Manager and select **Live update this component** to refresh the file list manually at any time.

## 43.2.2 Store Paths

Controls how path information is stored in the package for files in this component. Four options are available:

| Option  | Description  |
|---|--|
| <b>Use the default settings from the "Compression Options" page</b> | Inherits the global path storage setting.  |
| <b>Do not store path information</b>                                | Only filenames are stored — all files extract to the same folder with no subdirectory structure.                                   |
| <b>Store paths relative to the source folder</b>                    | Paths are stored relative to the linked source folder defined in Global Properties. Only available when a source folder is linked. |
| <b>Keep path information relative to a root folder</b>              | Paths are stored relative to a custom root folder that you specify.  |

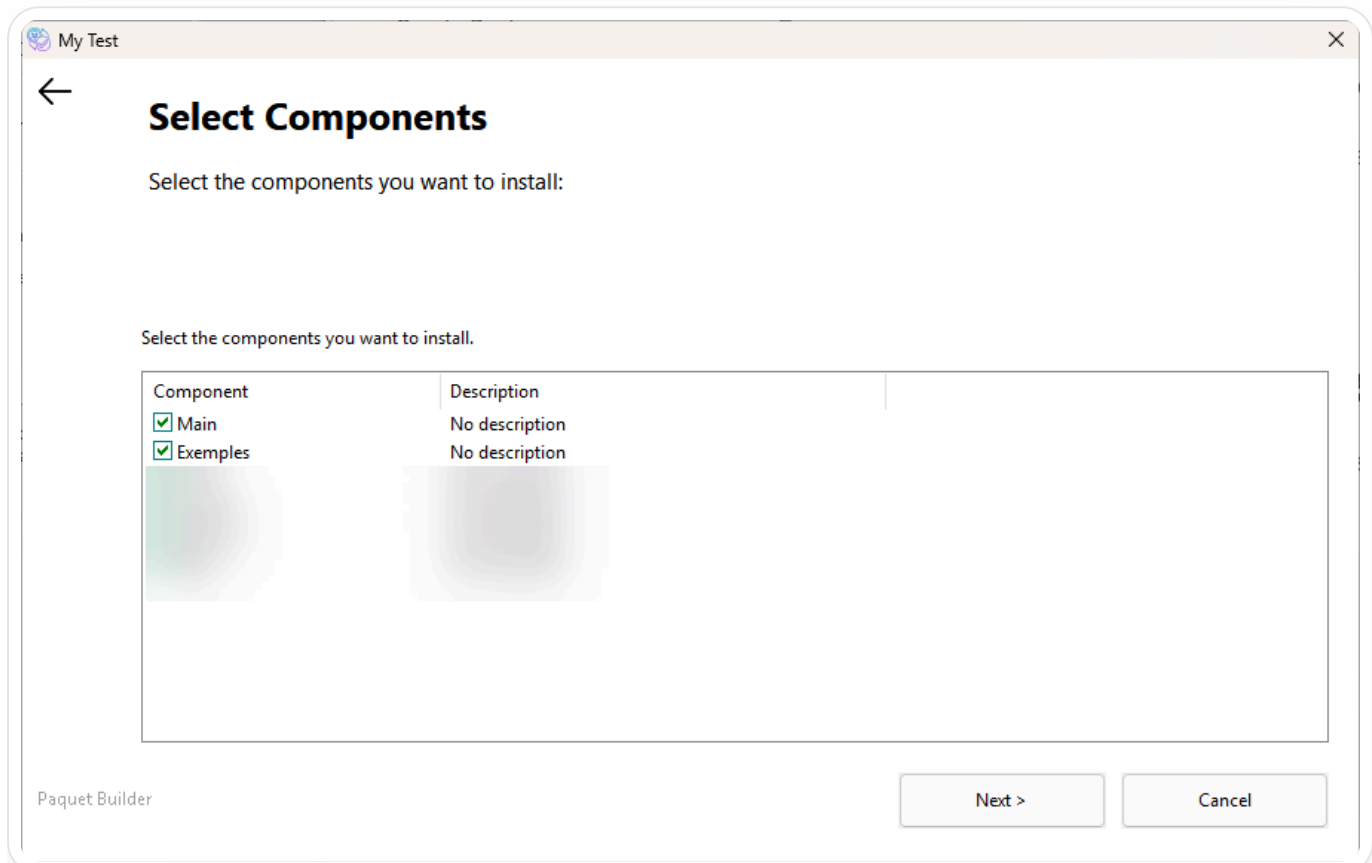
## 43.2.3 Component Variable

Each component is associated with a Boolean **variable** (typically prefixed `vCOMP`). The package evaluates this variable at runtime to determine whether the component's files should be extracted.

| Setting                   | Description   |
|---------------------------|---|
| <b>Component Variable</b> | The variable name (required). Can be set via <b>custom actions</b> or end-user selection. |

| Setting                                 | Description  |
|---|--|
| <b>Install the component by default</b> | When checked, the variable defaults to 1 (True) and files are extracted unless changed at runtime. |

### 43.2.4 Selection Dialog Box



Configure how the component appears in the [Component Selection Dialog](#):

| Setting                      | Description   |
|------------------------------|---|
| <b>Component Title</b>       | A short label describing the component (e.g. "Additional Movies"). Supports <a href="#">resource strings</a> .                              |
| <b>Component Description</b> | A longer explanation of the files the component installs. Supports resource strings.  |
| <b>Allow users to decide</b> | When enabled, the component appears in the selection list with a toggleable checkbox. When disabled, the component is hidden from the list. |
| <b>Required component</b>    | The component appears in the list but the checkbox is locked — end users cannot deselect it.  |

## 43.3 Advanced Usage



### Custom destinations

Assign individual files to different folders (e.g. %SYS%, %APPDATA%) via [File Properties](#).



### AI-powered organization

Use the [AI Assistant](#) to manage components via natural language: *"Create a component for Plugins and link it to C:\Dev\Plugins."*

## 43.4 Related Pages

### Automating Component Creation →

Use the Component Wizard to generate components from folder structures.

### File Manager →

Manage files within components.

### Select Components →

Configure the runtime component selection dialog.

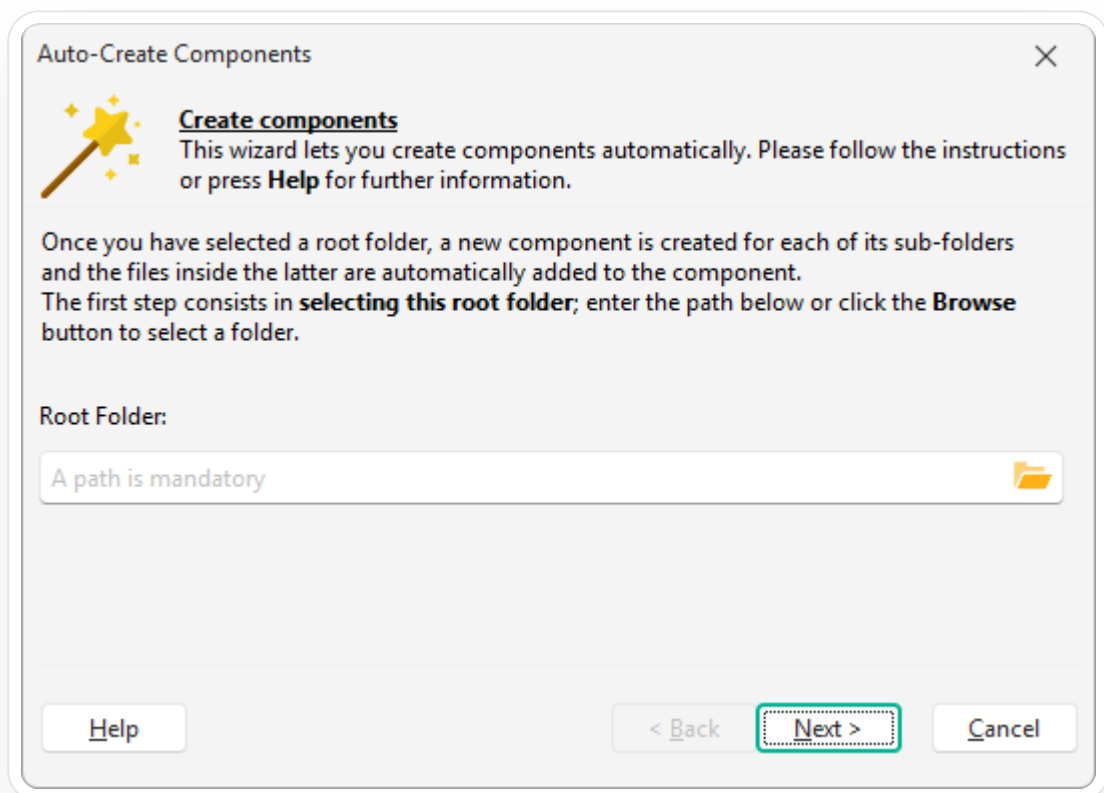
## 44. Automating Component Creation

If your project files are already organized into logical folders, the **Component Wizard** generates corresponding components in Paquet Builder automatically — saving time and keeping your package in sync with your development environment.

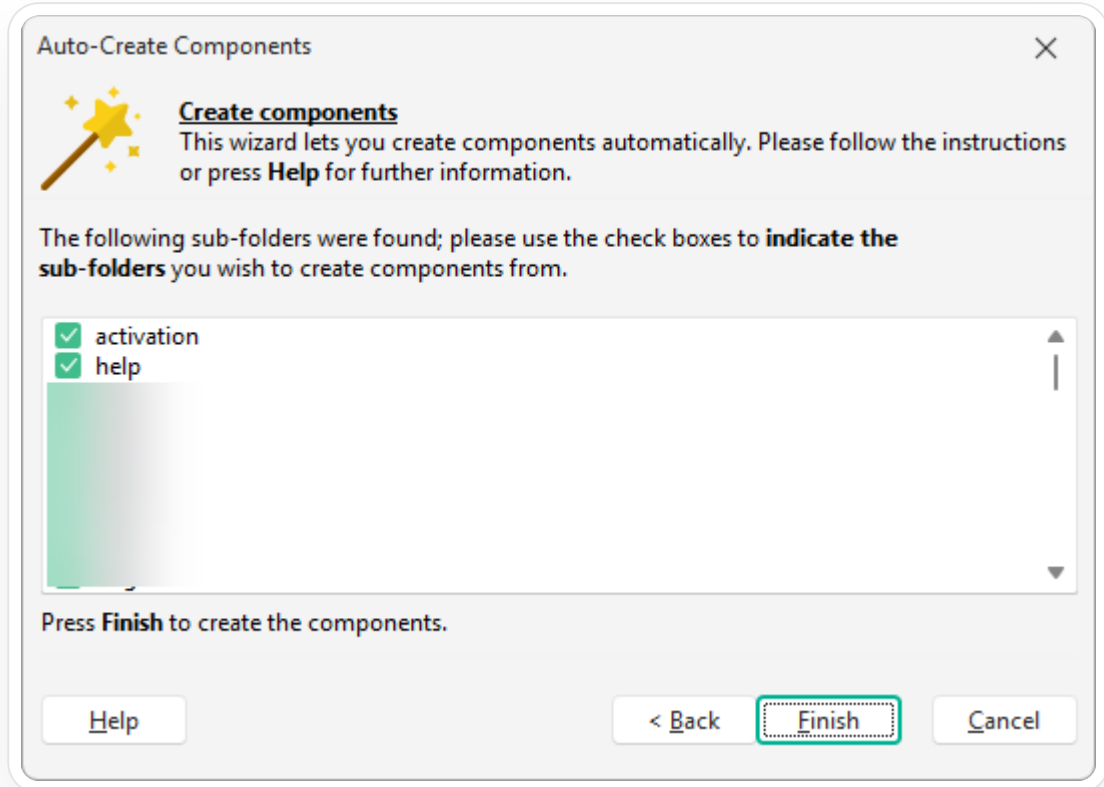
### 44.1 Running the Wizard

Access this feature by clicking **Component Wizard** in the **File Manager**.

- 1 **Select the root folder** containing your component subfolders.



- 2 **Choose subfolders** — the wizard displays all immediate sub-directories. Check the ones you want to convert into components.



- 3 Click **Finish** to create the components.

## 44.2 What the Wizard Does

For each selected folder, Paquet Builder automatically:

- Creates a new component **named after the folder**.
- Adds **all files** from that folder to the component.
- Enables **Source-Linking (Live Update)** by default.

### Note

**Source-Linked Components** are synchronized with your disk. If you add or remove files in the physical folder, the component updates automatically during the next build or manual refresh.

### Caution

The wizard only processes **immediate subfolders** of the selected root. Deeper nested folders are included as part of their parent component, preserving their internal directory structure.

## 44.3 Related Pages

### Components →

Learn how components work and configure their properties.

### File Manager →

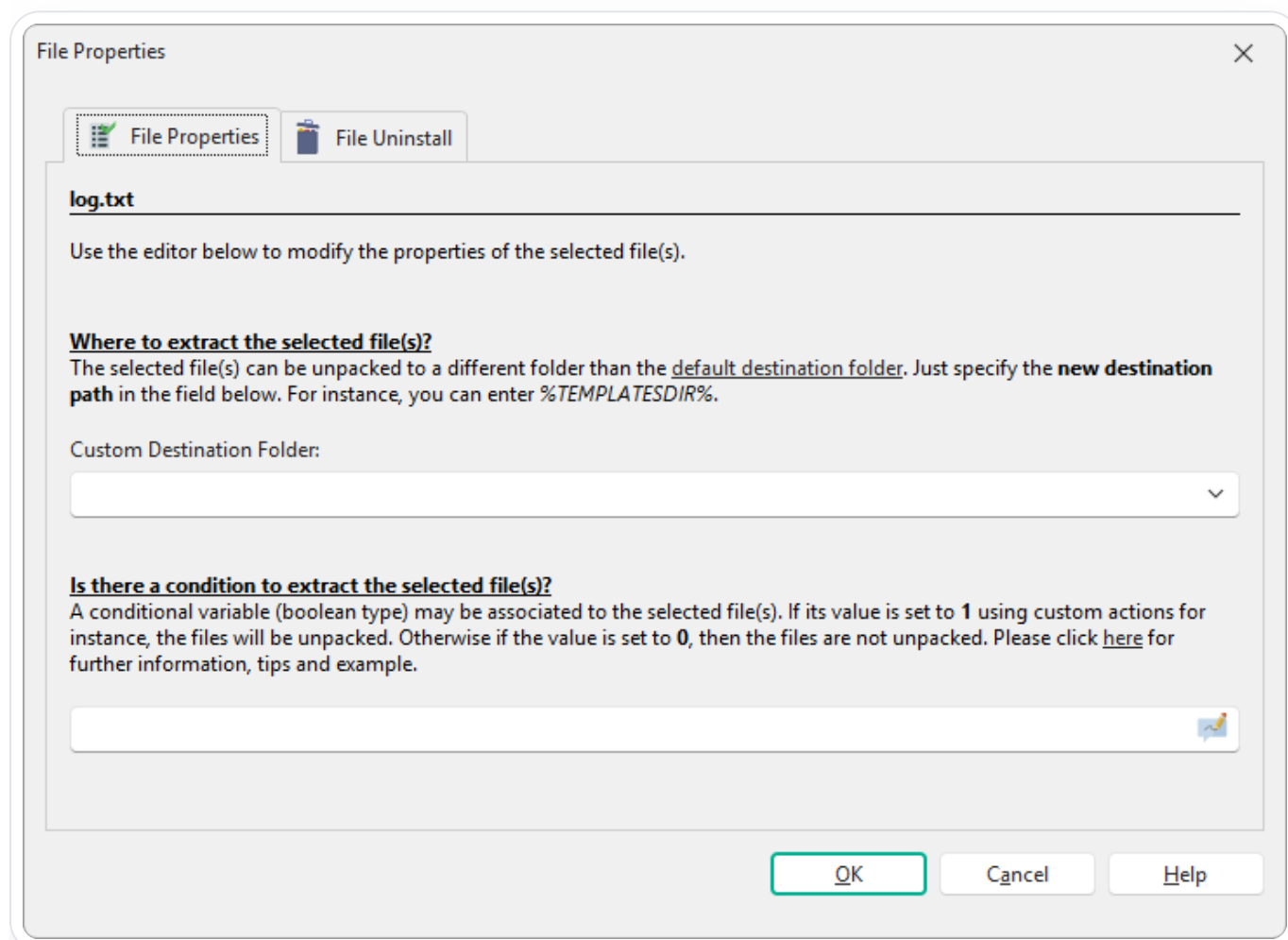
Manage files and components in your package.

## 45. Individual File Properties

While global settings define default behavior, you can **override properties for individual files** — place them in specific system directories, extract them conditionally, or prevent them from being uninstalled.

### 45.1 Accessing File Properties

- 1 Open the **File Manager**.
- 2 Select the file(s) to modify.
- 3 Click **Properties** in the toolbar, or double-click / right-click for the context menu.



### 45.2 Customizing File Destination

By default, files extract to the **Main Destination Folder**. You can override this per file:



## Variable-based paths

Use **variables** like %DESTPATH%\My Custom Path, %SYS%, %APPDATA%, OR %MYDOCDIR%\My Software\Assets to target dynamic system locations.



## Runtime paths

Set the destination dynamically at runtime using **Modify Variable** or **Ask User One Choice** custom actions.



### Caution

A custom destination folder **completely overrides** any path information stored within the archive for those files.

## 45.3 Conditional File Extraction

Control whether a file is extracted at runtime based on a **variable** value:

- 1 In the File Properties editor, enter a Boolean variable name (e.g. %EXTRACT\_DOCS%).
- 2 Use **Custom Actions** earlier in the flow to set it to 1 (extract) or 0 (skip).

### 45.3.1 Priority Rules

| Scenario                         | Behavior  |
|----------------------------------|---|
| <b>Component variable only</b>   | File extracts if the component is selected  |
| <b>Conditional variable only</b> | File extracts if the variable is 1  |
| <b>Both defined</b>              | <b>Conditional variable takes precedence</b> — the file extracts if its variable is 1, even if the parent component is not selected |

## 45.4 Uninstaller Options

If you have enabled the **Package Uninstaller**, you can set **"Do not remove the file"** for user-generated data, log files, or shared components that should remain on the system after removal.

## 45.5 Reset Properties

Right-click files in the File Manager and select **Clear Files Properties** to revert all custom settings (destination, condition, uninstaller) to default behavior.

## 45.6 Related Pages

### Main Destination Folder →

Set the default extraction path.

### Components →

Group files and control extraction at the component level.

### Variables →

Use variables in destination paths and conditions.

## 46. Resource Strings and Localization

Resource strings are **named text constants** used throughout your project. By using strings instead of hardcoded text, you can create a single installer that supports multiple languages seamlessly.

### 46.1 Using Resource Strings

Resource strings are identified by a # prefix (e.g. #BrowseFolders). At runtime, the package replaces them with the actual text for the user's selected language.

Click the Resource Button



next to any text field to insert a string.

#### Tip

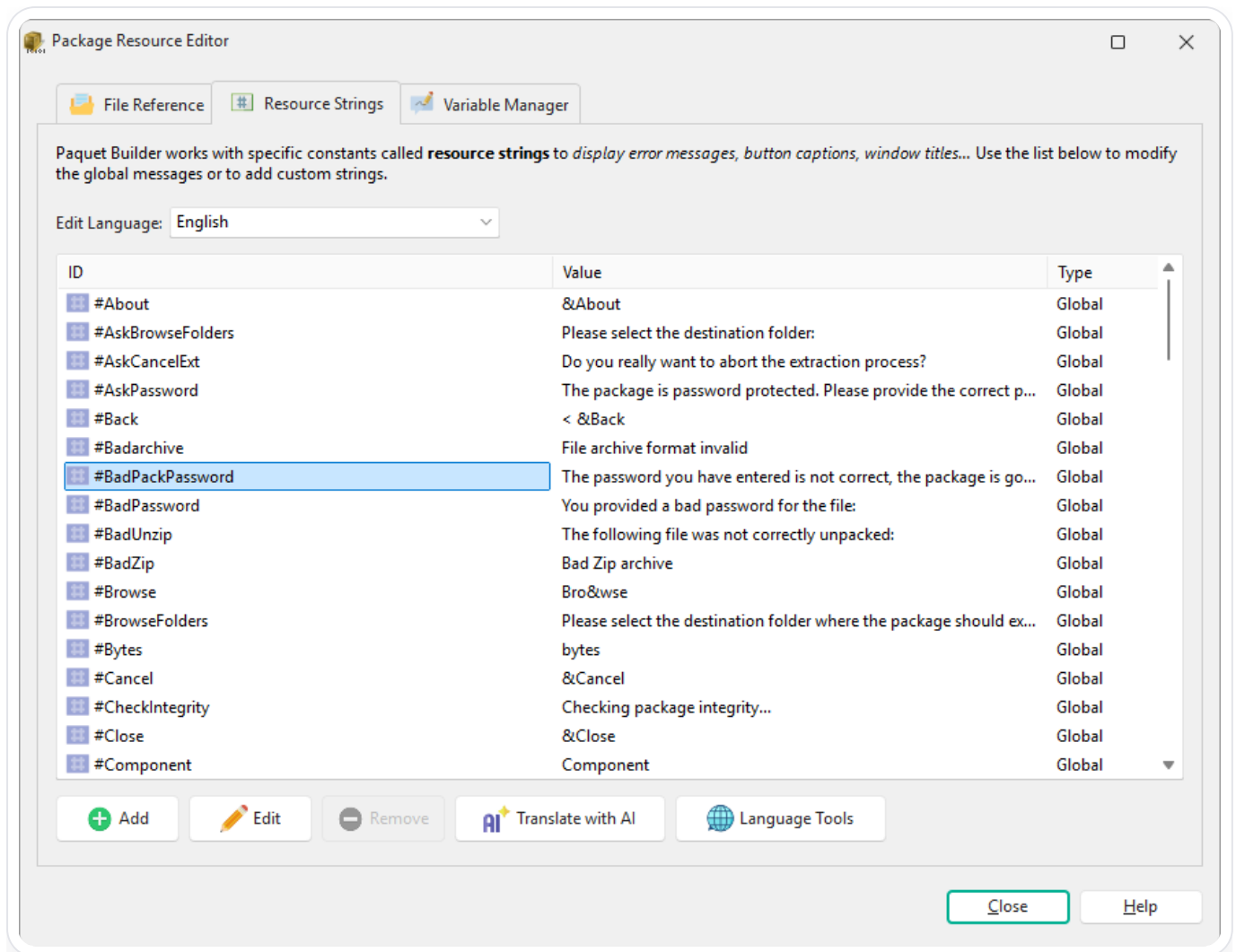
You can use resource strings almost anywhere in your project, including custom action properties and wizard dialogs.

#### Note

Resource strings **cannot** be used in: Package Title, Company Name, Email Address, and Destination Folder.

### 46.2 The Resource String Editor

Manage all your project's text and translations through the **Resource String Editor**, accessible via the toolbar or the [Language Manager](#).



## 46.2.1 Editing Strings

Double-click a string or click **Edit** to open its properties:

Resource String Properties
✕

---

#
String Editor

Use the editor below to modify the current string and press **OK** to save changes.

ID (must begin by #):

Value:

| Field        | Description  |
|--------------|--|
| <b>ID</b>    | Unique name starting with # (e.g. #UserWelcome). Custom IDs can be renamed; Global IDs cannot.   |
| <b>Value</b> | The actual text displayed to the user. For example, #UserWelcome might be <i>"Welcome!"</i> in English and <i>"Bienvenue !"</i> in French. |

## 46.2.2 Actions



### Multi-selection

Select multiple strings for batch **Edit**, **Delete**, or **Translate** operations.



### Translate with AI

Automatically translate selected strings into all other project languages using your configured **AI provider**.



## Fill Missing Strings

Copy all identifiers from the current language to other languages where they are missing — ensures all translation sets have the same structure.



## Language Tools

Access import/export options and additional language management utilities.

### 46.2.3 Language Properties

Each language file has its own settings:

| Property                 | Description   |
|--------------------------|---|
| <b>LCID</b>              | Windows Locale Identifier (e.g. 1033 for English, 1036 for French) — used for automatic detection |
| <b>Author</b>            | Name of the translator  |
| <b>Email</b>             | Contact email for the translator  |
| <b>Reading Direction</b> | LTR or RTL (for Arabic, Hebrew, etc.)   |

### 46.3 Creating Multilingual Packages

- 1 **Add languages** via the [Language Manager](#).
- 2 **Use resource strings** (identifiers) for all text fields in your package.
- 3 **Provide translations** in the Resource String Editor for each language.

### 46.4 Technical Details

| Detail             | Value            |
|--------------------|------------------|
| <b>File format</b> | .pbr — UTF-8 XML |

| Detail            | Value                               |
|-------------------|-------------------------------------|
| Reading direction | LTR and RTL supported               |
| Import/Export     | .pbr files reusable across projects |

## 46.5 Related Pages

### Language Manager →

Add and manage languages in your project.

### AI Assistant Setup →

Configure AI for automated translations.

### Variable List →

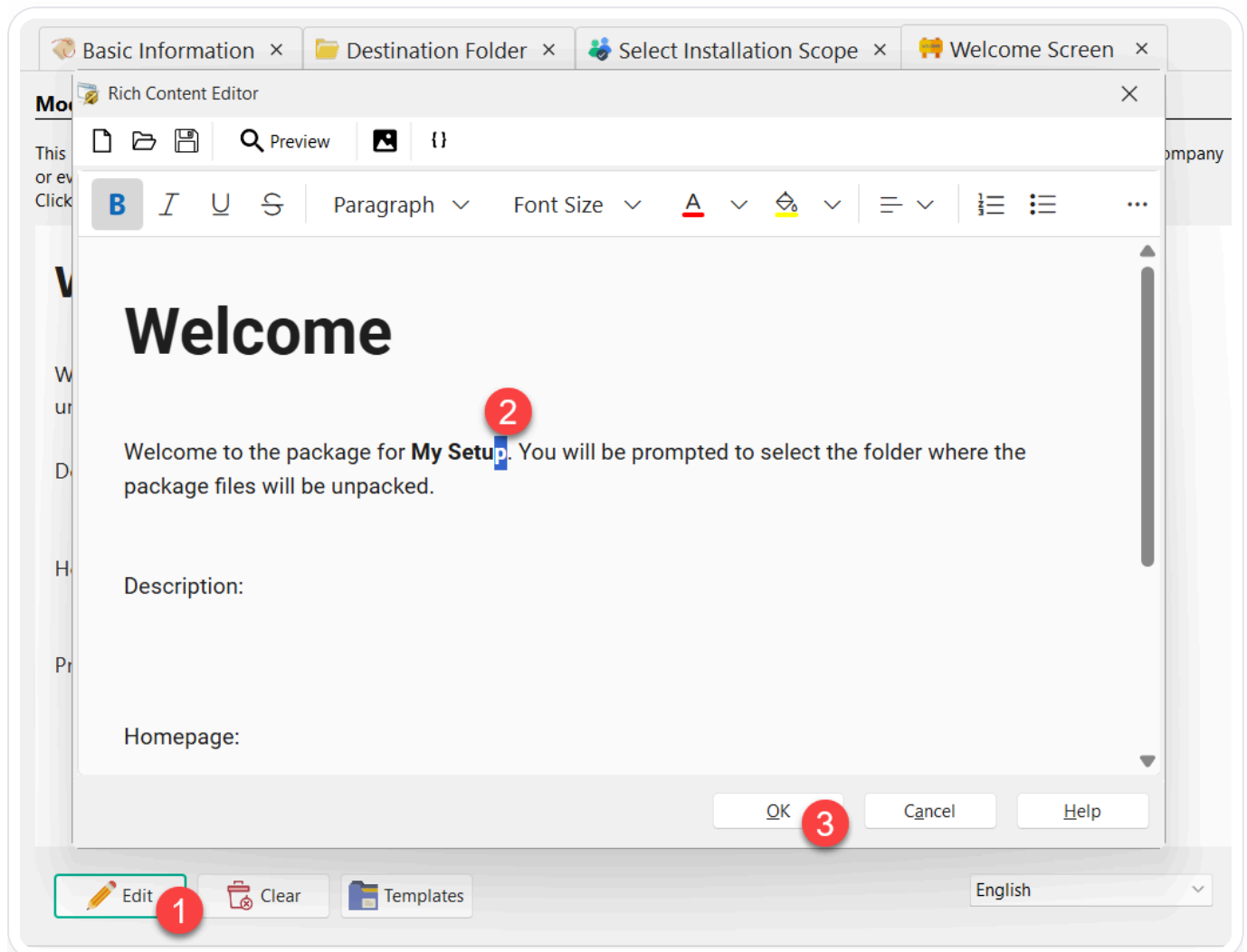
All built-in variables you can combine with resource strings.

## 47. Designing Rich Content

Paquet Builder includes a built-in **Rich Content Editor** for designing professional, visually engaging screens. Use it to customize your [Welcome Page](#), [Readme/EULA](#), and [Final Screen](#).

### 47.1 The Editor

The editor provides a **visual WYSIWYG interface** similar to a word processor. Use the toolbar to apply bold, italics, colors, alignments, and insert images without touching any code. You can also switch to the **HTML source view** for full control.



#### **Note**

The **Preview** rendering engine may differ slightly from the actual installer's rendering. Minor visual differences in fonts or spacing may appear between the preview and the final result.

### 47.2 Key Features



## Live Preview

Click **Preview** to see exactly how your content will look inside the final installer window.



## Import / Export

Load and save content in multiple formats: HTML (.htm, .html), Rich Text (.rtf), Markdown (.md), and plain text (.txt).



## Variables and Strings

Insert **Resource Strings** and **Variables** (e.g. %APPTITLE%) — they resolve to actual values at runtime.



## UTF-8 Support

All content is stored in UTF-8 format for full Unicode compatibility with global languages.

## 47.3 Adding Images

Enhance your dialogs by embedding images (PNG or BMP format):

| Setting                   | Detail  |
|---------------------------|---|
| <b>Recommended format</b> | PNG — best balance of quality and file size                           |
| <b>Storage</b>            | Images must reside in the same directory as your project (.pbpx) file |
| <b>Alignment</b>          | Images are automatically centered relative to surrounding text        |



### Caution

If you move your project file to a new location, ensure all associated image files are moved with it to maintain the links.



### Tip

Use the **Preview** button frequently when adding images to ensure they fit correctly within the available width of the installer dialog.

## 47.4 Related Pages

## Welcome Screen



Configure the first dialog seen by end users.

## Readme and License Agreement



Add EULA and readme text screens.

## Edit Final Screen



Configure the completion dialog.

## 48. Managing the Package Uninstaller

Paquet Builder includes a built-in uninstaller engine that ensures your application can be completely removed from the user's system. When enabled, it appears in **Settings > Apps > Installed Apps** (or **Add/Remove Programs**).

### 48.1 What the Uninstaller Removes



#### Files and folders

All extracted files and any empty directories created by the package.



#### Shortcuts

Start Menu, Desktop, and Taskbar links created during installation.



#### System configuration

.ini file entries and registry keys (under **HKCU** or **HKLM**).

### 48.2 Enabling the Uninstaller

- 1 Navigate to **Functionality > Configure Uninstaller**.
- 2 Activate **Include the Uninstaller engine**.
- 3 Configure the **Product Name**, **Unique Registry Key**, and **EXE Path**.

#### Caution

Without this engine, your package provides no automatic way for users to remove the software.

### 48.3 Advanced Options



#### Prevent file removal

Select a file in the **File Manager** and enable *"Do not remove"* in **File Properties** — useful for user data, logs, or shared components.



## Shared file support

The uninstaller checks the system usage counter for shared DLLs/OCXs and only removes them if no other application is using them. See [Registering Shared Files](#).



## Custom uninstall commands

Define additional cleaning tasks with [Custom Uninstall Commands](#) — run scripts, de-register ActiveX controls, or prompt for feedback.

## 48.4 UAC and Elevation

The uninstaller inherits the same **UAC level** as the main installer. If your installer required admin rights to place files in `Program Files`, the uninstaller also requires them.



### Tip

See [UAC and Elevation](#) and [Installation Scope](#) for details on privilege management.

## 48.5 Related Pages

### Configure Uninstaller →

Set the uninstaller properties, registry key, and options.

### File Properties →

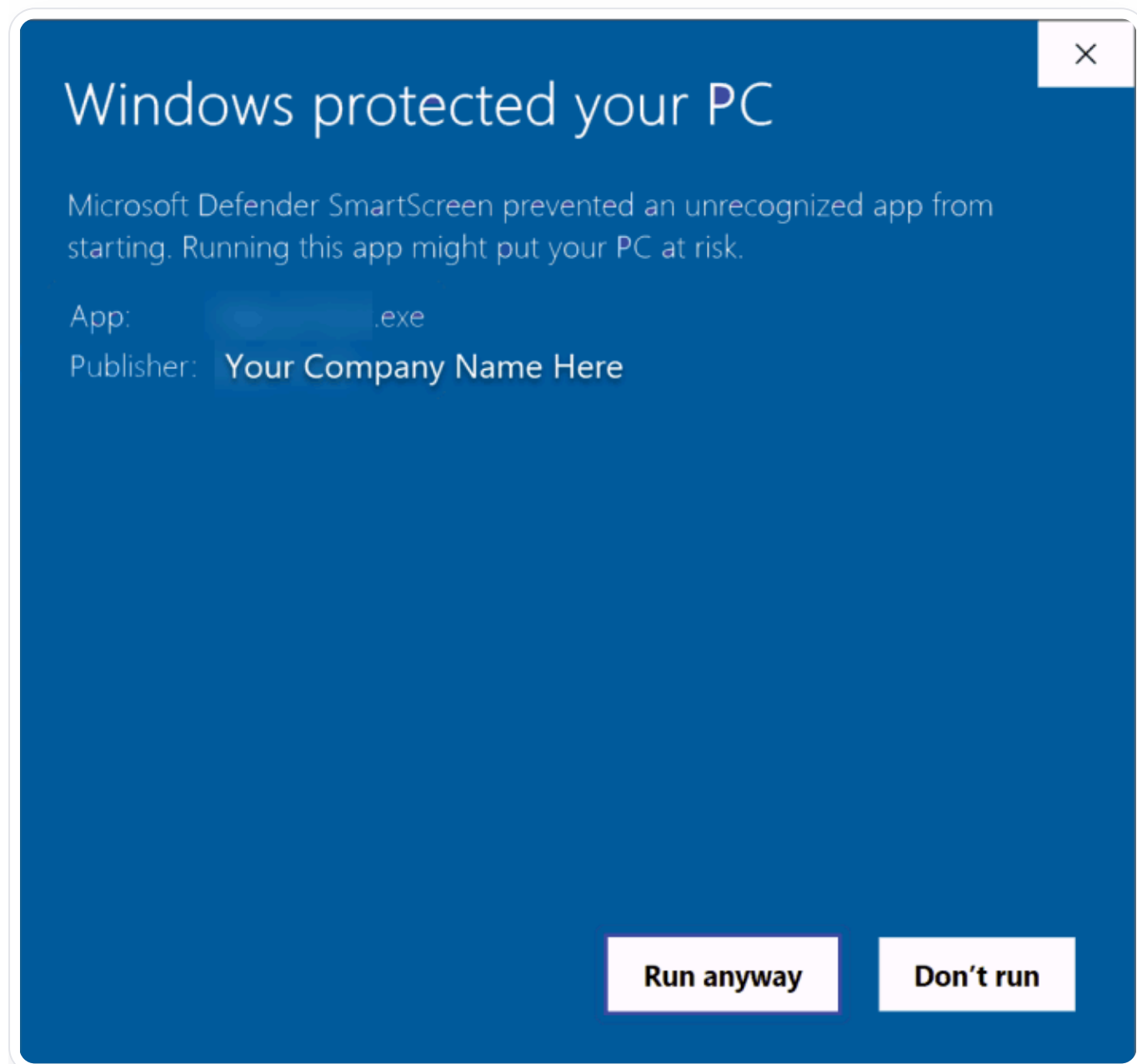
Control per-file uninstaller behavior.

### Custom Actions →

Add logic to the uninstall flow.

## 49. User Account Control (UAC) and Elevation

Windows User Account Control (UAC) ensures applications run with standard user privileges by default. If your installer needs admin tasks — writing to `Program Files`, modifying `HKLM`, or registering services — it must explicitly request elevated rights.



### 49.1 Requested Execution Levels

Configure elevation behavior in **Build > EXE Output Configuration**. Paquet Builder embeds the appropriate XML manifest into the final `.exe`.

Runs with the **same privileges** as the process that launched it. No UAC prompt appears.

Best for portable apps, simple file extractors to user folders, or packages that don't modify system settings.

Always attempts to run with **full administrative tokens**. A UAC prompt always appears.

Best for standard installers that need full system access.

Runs with the **highest privileges** the current user can obtain. Admins get full privileges; standard users get standard privileges (no UAC prompt).

Best for installers that work in both scenarios.

## 49.2 Best Practices



### Use Installation Scope

Instead of managing UAC manually, use **Installation Scope**. Paquet Builder handles the manifest, path variables, and registry logic automatically.



### Sign your installer

Always **digitally sign** elevated installers. Windows displays a prominent *"Unknown Publisher"* warning for unsigned packages.



### Test on Standard User

Always test the final `.exe` on a **Standard User** account to ensure the UAC prompt appears correctly and permissions are sufficient.

## 49.3 Related Pages

### Installation Scope



Automatic UAC and path management based on scope selection.

### Digital Signature



Sign your installer to avoid Unknown Publisher warnings.

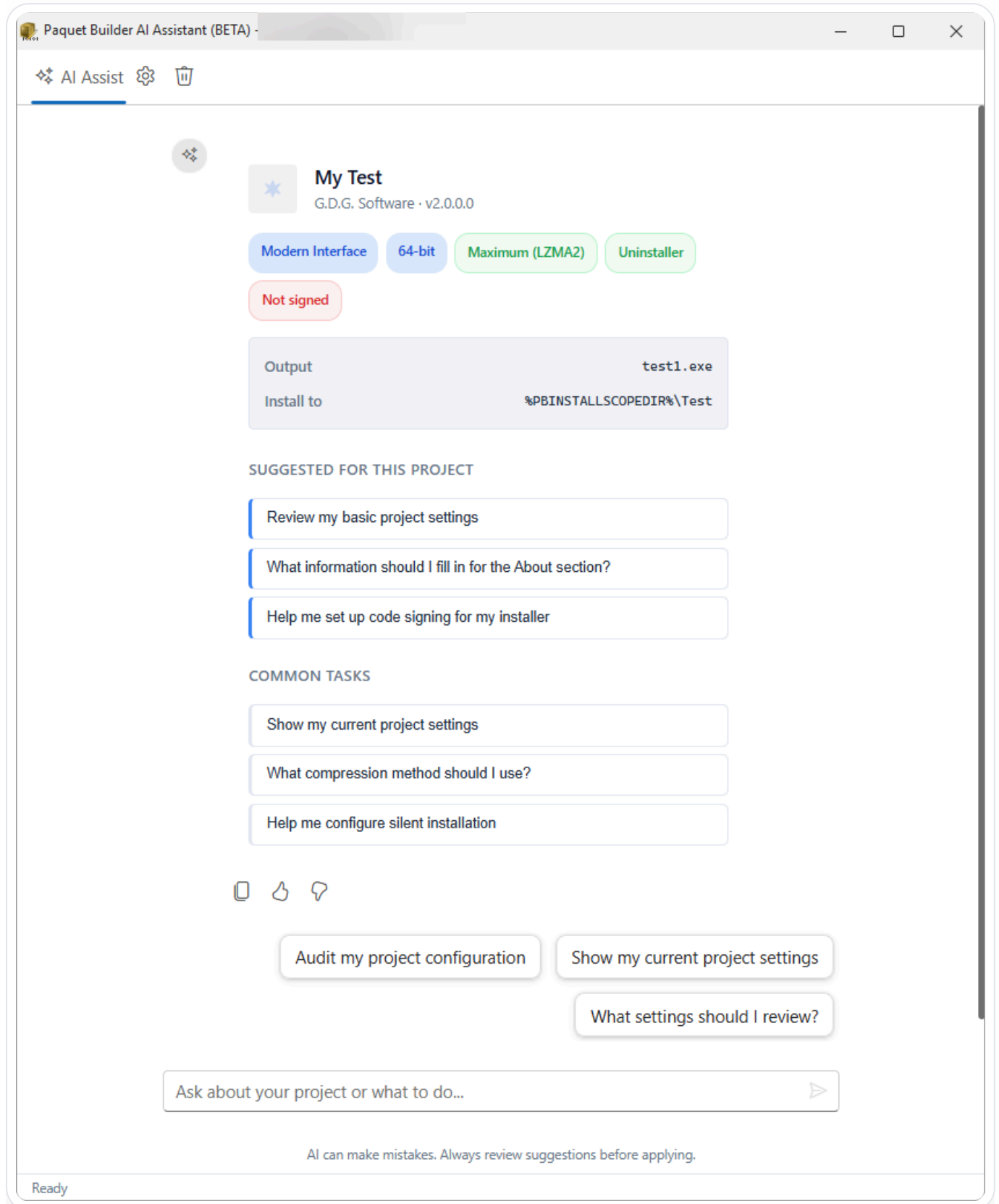
## EXE Output Configuration



Set elevation levels and other build options.

# 50. AI Assistant Overview

Paquet Builder includes a built-in **AI-powered assistant** that helps you configure your installer projects through natural language conversation. Instead of navigating dozens of settings pages, you can simply describe what you want — and the AI will propose the right changes.



## 50.1 Key Capabilities

The AI Assistant can:

- **Configure 100+ project settings** via natural language — **compression**, **signing**, **encryption**, **scope**, **interface**, and more
- **Manage files and components** — add source folders, **create components**, set file masks and properties via the **File Manager**
- **Create and edit custom actions** — **shortcuts**, **registry entries**, conditions, dialogs, across all 15 events
- **Manage variables and localization** — add variables, translate resource strings to multiple languages
- **Search product documentation** — find answers grounded in official docs, not guesswork
- **Navigate the application** — open any settings page, **File Manager**, or **Custom Actions editor**
- **Compile projects** and troubleshoot build errors
- **Capture screenshots** for visual context analysis

## 50.2 How It Works

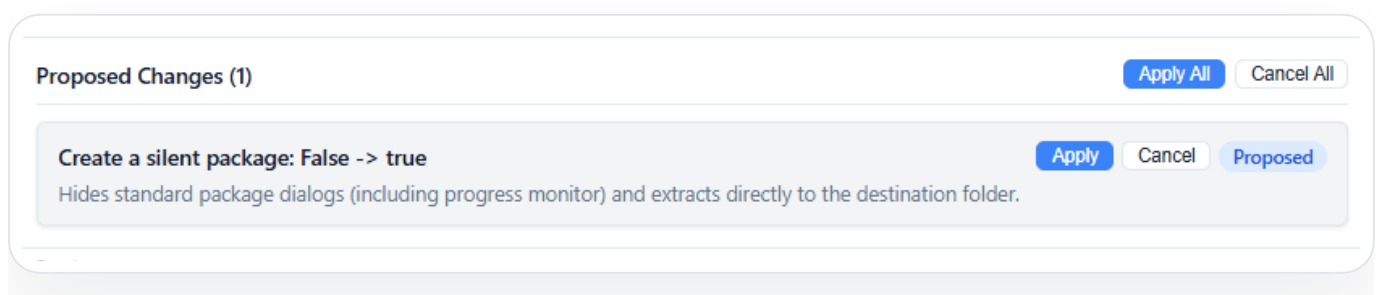
The assistant connects to a **cloud AI provider** (OpenAI, Claude, Gemini, and others) or a **local AI model** (Ollama, LM Studio). When you ask a question or make a request, the AI:

- 1 Analyzes your **project context** — current settings, active page, project state
- 2 Uses **specialized tools** to read your configuration and search documentation
- 3 Proposes changes as **reviewable action cards**
- 4 Waits for **your approval** before applying anything

### **Note**

The AI never modifies your project directly. Every change is proposed as an action card that you can review, accept, or reject.

## 50.3 Proposed Action Cards



When the AI suggests changes, they appear as action cards in the chat window. Each card shows:

- A clear description of what will change
- **Apply** and **Cancel** buttons for individual control
- **Apply All** and **Cancel All** buttons for batch operations

### Caution

Potentially destructive actions (like deleting components or clearing events) are flagged with a warning badge so you can review them carefully.

After applying changes, the AI may display **follow-up suggestion chips** — clickable prompts for related tasks you might want to do next.

## 50.4 Context Awareness

The AI assistant automatically detects:

- Your **current project** — name, version, company, output path, architecture, compression, signing status
- Which **settings page** you are viewing — and tailors suggestions accordingly
- Which **dialog** you just closed — so it knows what you were working on
- Whether your project has **potential issues** — unsigned installer, missing uninstaller, etc.

### Tip

Navigate to the relevant settings page before asking a question. The AI detects your current page and provides more targeted suggestions.

## 50.5 Dark and Light Theme

The AI assistant window automatically follows your application theme preference, providing a consistent visual experience in both light and dark modes.

## 50.6 Getting Started

- 1 Set up your AI provider and API key
- 2 Learn what the AI can do
- 3 See prompt examples and usage tips

4

Use local AI models for privacy

## 50.7 Related Topics

### MCP Server →

Headless project management for AI tools like Claude Code and Cursor

### About Directive Files →

Script-based automation alternative


### Console Command-line Compiler →

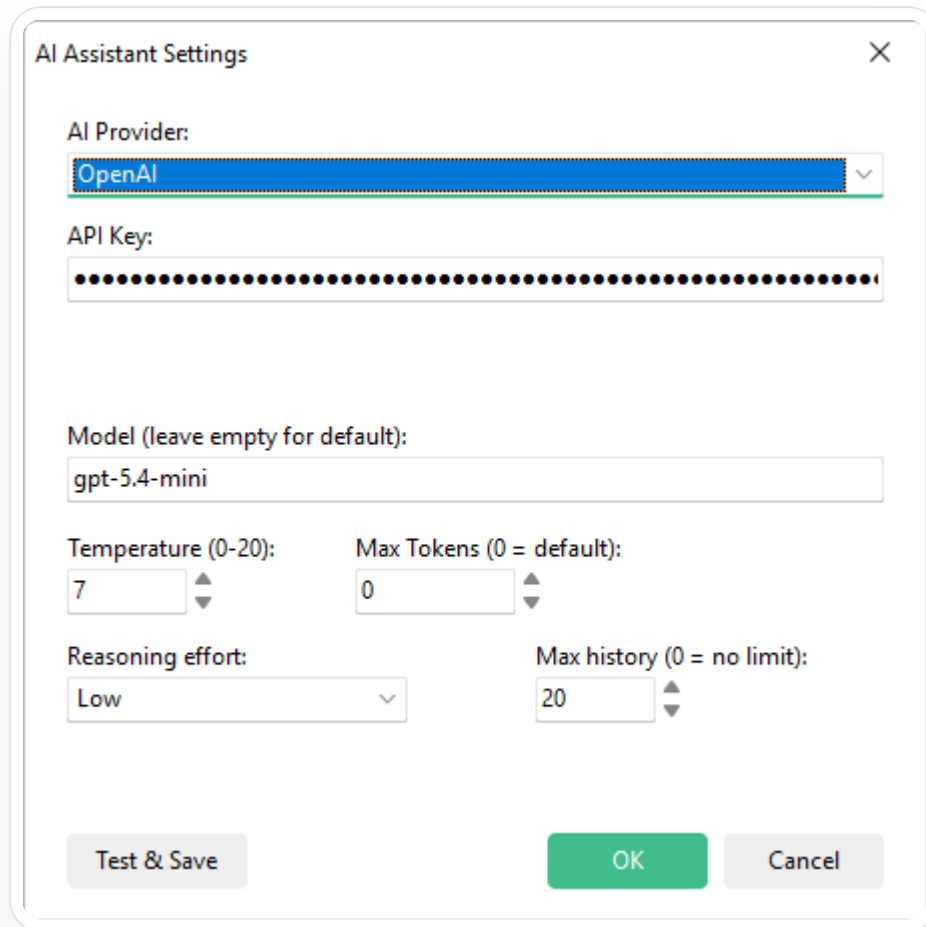
Batch compilation without GUI

### Environment Options →

Global application settings including AI configuration

# 51. Setting Up the AI Assistant

The first time you open the AI Assistant, the settings dialog appears automatically. You can also access it later by clicking the  **gear icon** in the AI Assistant window.



The image shows a dialog box titled "AI Assistant Settings" with a close button (X) in the top right corner. The dialog contains the following fields and controls:

- AI Provider:** A dropdown menu with "OpenAI" selected.
- API Key:** A text input field filled with black dots, indicating a masked password.
- Model (leave empty for default):** A text input field containing "gpt-5.4-mini".
- Temperature (0-20):** A numeric input field with a value of "7" and up/down arrow controls.
- Max Tokens (0 = default):** A numeric input field with a value of "0" and up/down arrow controls.
- Reasoning effort:** A dropdown menu with "Low" selected.
- Max history (0 = no limit):** A numeric input field with a value of "20" and up/down arrow controls.

At the bottom of the dialog, there are three buttons: "Test & Save" (disabled), "OK" (highlighted in green), and "Cancel" (disabled).

## 51.1 Quick Setup

- 1 **Choose a provider** from the dropdown list (see [available providers](#) below).
- 2 **Enter your API key** from your provider's website into the **API Key** field.
- 3 **Select a model** or leave empty to use the provider's default.
- 4 **Click OK** — you're ready to chat with the AI Assistant.

## 51.2 Available Providers

Cloud providers require an API key and an internet connection. They offer the most capable models.

| Provider           | API Key Required | Sign-Up               |
|--------------------|------------------|-----------------------|
| OpenAI             | Yes              | platform.openai.com   |
| Anthropic (Claude) | Yes              | console.anthropic.com |
| Google (Gemini)    | Yes              | aistudio.google.com   |
| Grok (xAI)         | Yes              | console.x.ai          |
| Mistral            | Yes              | console.mistral.ai    |
| DeepSeek           | Yes              | platform.deepseek.com |

Local providers run on your machine — no API key needed, no data leaves your computer.

| Provider             | API Key Required | More Info   |
|----------------------|------------------|-------------|
| Ollama               | No               | ollama.com  |
| LlamaCpp / LM Studio | No               | lmstudio.ai |

For setup instructions, see [Using Local AI Models](#).

## 51.3 API Key Security

### Note

API keys are stored **encrypted** on your machine in %APPDATA%\GDG Software\Paquet Builder\aikeys.cfg. They are never included in project files and never sent anywhere except to the selected AI provider.

## 51.4 Choosing a Model

Enter the model name in the **Model** field. If left empty, the provider's default model is used.

## Recommended Models

For the best experience with Paquet Builder's AI Assistant:

- **OpenAI:** `gpt-5.4-mini` — Fast, affordable, excellent tool support
- **Anthropic:** `claude-sonnet-4-6` (balanced) or `claude-opus-4-6` (most capable)
- **Google:** `gemini-2.5-flash` — Free tier available, good for getting started

For a detailed comparison of all providers and models, see [AI Providers and Models](#).

## 51.5 Advanced Settings

Controls response creativity on a scale from **0.0 to 1.0**. Default: **0.7**.

| Range     | Behavior                  | Best For                      |
|-----------|---------------------------|-------------------------------|
| 0.0 – 0.4 | Deterministic, consistent | Precise configuration tasks   |
| 0.5 – 0.7 | Balanced                  | General usage                 |
| 0.8 – 1.0 | Creative, varied          | Brainstorming and exploration |

Maximum response length. Leave at **0** to use the provider's default limit.

Only increase this if responses are being cut off.

Number of previous messages included in context. Default: **20**.

| Value        | Effect  |
|--------------|---|
| Lower (5-10) | Saves tokens and reduces costs                        |
| Default (20) | Good balance for most tasks                           |
| Higher (30+) | Better multi-turn context retention                   |
| 0            | Unlimited history (not recommended for long sessions) |

Controls thinking depth for OpenAI's o-series and GPT-5 models.

Options: **Default, None, Minimal, Low, Medium, High**.

### Note

This setting is only visible when **OpenAI** is selected as the provider.

## 51.6 Mock Mode

When no API key is configured, the assistant automatically runs in **mock mode** with simulated responses. This lets you explore the interface and understand the action card workflow without connecting to any AI provider.

## 51.7 Independent Storage

AI settings are stored separately from your project files and main Paquet Builder preferences. Switching projects does not affect your AI configuration.

| File   | Contents                               |
|--|--|
| %APPDATA%\GDG Software\Paquet Builder\pbprefer.xml | Provider, model, and advanced settings |
| %APPDATA%\GDG Software\Paquet Builder\aikeys.cfg   | Encrypted API keys                     |

## 51.8 Related Topics

### AI Providers and Models →

Detailed provider comparison.

### Using Local AI Models →

Ollama, LM Studio, llama.cpp setup.

### What the AI Assistant Can Do →

Full capabilities reference.

### Using the AI Assistant →

Prompt examples and tips.

## 52. AI Providers and Models

The Paquet Builder AI Assistant supports multiple AI providers — both cloud-based services and locally hosted models. This page helps you choose the right provider and model for your needs.

### 52.1 Cloud Providers

| Provider                  | Recommended Model              | Function Calling | Best For                     | Notes                                       |
|---------------------------|--------------------------------|------------------|------------------------------|---|
| <b>OpenAI</b>             | <code>gpt-5.4-mini</code>      | Excellent        | Fast tasks, affordable usage | Reasoning effort option for o-series models |
| <b>Anthropic (Claude)</b> | <code>claude-sonnet-4-6</code> | Excellent        | Complex multi-step tasks     | Best accuracy for tool calling              |
| <b>Anthropic (Claude)</b> | <code>claude-opus-4-6</code>   | Excellent        | Most demanding tasks         | Premium, highest capability                 |
| <b>Google (Gemini)</b>    | <code>gemini-2.5-flash</code>  | Good             | Budget-conscious usage       | Free API tier available for low volume      |
| <b>Grok (xAI)</b>         | —                              | Good             | Alternative provider         |   |
| <b>Mistral</b>            | —                              | Good             | European-hosted provider     | EU data residency                           |
| <b>DeepSeek</b>           | <code>deepseek-chat</code>     | Good             | Very affordable              | Competitive pricing                         |

### 52.2 Local AI Providers

Run AI models on your own machine — no API key, no internet, no data leaves your computer.

| Software         | Provider Setting | Default Port | Notes                             |
|------------------|------------------|--------------|-----------------------------------|
| <b>Ollama</b>    | Ollama           | 11434        | Native integration, easiest setup |
| <b>LM Studio</b> | LlamaCpp         | 1234         | User-friendly model browser       |

| Software         | Provider Setting | Default Port | Notes                           |
|------------------|------------------|--------------|---------------------------------|
| llama.cpp server | LlamaCpp         | 8080         | Lightweight, for advanced users |

See [Using Local AI Models](#) for step-by-step setup instructions.

## 52.3 Choosing the Right Model

Changing settings, adding files, querying project state — use a **fast, affordable** model:

- **GPT 5.4-mini** (OpenAI) — Best speed-to-quality ratio
- **Gemini 2.5 Flash** (Google) — Free tier, good for experimentation

Creating custom action workflows, configuring scope + signing + compression in one conversation:

- **Claude Sonnet 4.6** (Anthropic) — Excellent tool-calling accuracy
- **GPT 5.4** (OpenAI) — Strong multi-step reasoning

Full project setup from scratch, complex conditional custom actions, debugging build issues:

- **Claude Opus 4.6** (Anthropic) — Highest overall capability

When data privacy is paramount or you want to avoid API costs:

- **Ollama** with Llama 3, Qwen 2.5, or Mistral — Requires a model with function calling support

## 52.4 Cost Considerations

Usage costs depend on the provider and model you choose. Most cloud providers offer **pay-as-you-go** pricing based on the number of tokens (words) processed.

A typical Paquet Builder session with 10–20 exchanges costs:

- **GPT 5.4-mini**: A few cents
- **Claude Sonnet 4.6**: Around \$0.05–0.15
- **Claude Opus 4.6**: Around \$0.30–0.80
- **Gemini 2.5 Flash**: Free tier covers most usage

### Tip

Start with a free-tier provider like **Google Gemini** to explore the AI assistant's capabilities before committing to a paid plan.

### Caution

Local AI models vary widely in their support for function calling (tool use). If the assistant frequently fails to use its tools correctly with a local model, try a larger model or switch to a cloud provider.

## 52.5 Provider-Specific Notes

### 52.5.1 OpenAI

- **Reasoning Effort:** Available for o-series and GPT-5 models. Controls how much “thinking” the model does before responding. Higher effort = better results for complex tasks but slower and more expensive.
- The default model (when the field is empty) is `gpt-4o`.

### 52.5.2 Anthropic (Claude)

- Generally provides the most reliable tool-calling behavior for multi-step operations.
- Supports both Sonnet (faster, cheaper) and Opus (more capable) tiers.

### 52.5.3 Google (Gemini)

- Free API tier available at [aistudio.google.com](https://aistudio.google.com) with generous usage limits.
- Some complex tool-calling scenarios may require retry (handled automatically by Paquet Builder).

### 52.5.4 DeepSeek

- Very competitive pricing, often 10-20x cheaper than equivalent Western providers.
- Good function calling support for most operations.

## 52.6 Related Topics

### Setting Up the AI Assistant

Configuration guide

## Using Local AI Models →

Ollama, LM Studio, llama.cpp

## What the AI Assistant Can Do →

Full capabilities reference

## Using the AI Assistant →

Prompt examples and tips

## MCP Server Overview →

Use AI tools like Claude Code with Paquet Builder

## 53. What the AI Assistant Can Do

The AI Assistant has access to a comprehensive set of tools that let it read, analyze, and propose changes to virtually every aspect of your installer project. This page describes each capability area in detail.

### 53.1 Project Settings

The AI can read and modify **over 100 project settings** across 21 categories:

- **Package identity** — title, company, email, homepage, copyright
- **Output and version info**
- **Interface theme** — wizard, standard, modern
- **Installation scope** — per-user, all-users, user choice
- **Compression** — LZMA, LZMA2, level, threads, solid mode
- **Encryption and password protection**
- **Code signing** — PFX, SignTool, Azure Artifact Signing, JSign
- **Destination paths** and installation directories
- **Execution options** — setup loader, silent mode, UAC
- **Shortcut configuration**
- **Uninstaller settings**

#### Tip

The AI understands **setting dependencies, effects, and constraints**. For example, asking “Enable LZMA2 compression with ultra level” will propose changes to multiple related settings at once.

### 53.2 File and Component Management

The AI can work with your project’s file collection through the **File Manager**:

| Capability               | Details  |
|--------------------------|--|
| <b>List components</b>   | View all <b>components</b> with their properties and file counts |
| <b>Create components</b> | Set title, description, and other properties                     |

| Capability                    | Details   |
|-------------------------------|---|
| <b>Link to source folders</b> | Enable <b>automatic synchronization</b>                                   |
| <b>Set file masks</b>         | Include or exclude specific file types (e.g., *.dll;*.exe or *.pdb;*.log) |
| <b>Configure properties</b>   | Required, default selection, display visibility, <b>storing path mode</b> |
| <b>Search files</b>           | Find files across all components  |

### 53.3 Variable Management

| Action        | Details  |
|---------------|--|
| <b>List</b>   | View all project variables with their types and values |
| <b>Add</b>    | Create new variables (string, path, or boolean types)  |
| <b>Edit</b>   | Modify existing variable values and properties         |
| <b>Delete</b> | Remove custom variables                                |

The AI understands how variables are used in **custom actions** and settings.

### 53.4 Localization and Resource Strings

- **Query resource strings** across all languages in the project
- **Add or modify translations** for specific strings
- **Manage language collections** — list languages, view string coverage
- Works with both global (predefined) and custom resource strings

#### Tip

For bulk translation of resource strings, you can also use the dedicated **"Translate with AI" button** in the Resource Strings editor.

## 53.5 Registry Entries

- **View** all **registry entries** configured in the project
  - **Add new registry keys and values** (string, DWORD, binary, expandable string)
  - **Modify** existing registry entries
  - **Delete** registry entries
  - Automatically adapts registry root (HKCU/HKLM) based on **installation scope**
- 

## 53.6 Shortcuts

- **Query** all configured Start Menu and Desktop shortcuts
  - **Add new shortcuts** with target path, arguments, icon, working directory
  - **Modify** shortcut properties
  - **Delete** shortcuts
  - Shortcut locations adapt to **installation scope** (user vs. common Start Menu/Desktop)
- 

## 53.7 Screen Content

The AI can **read and modify HTML content** for the five installer screens:

| Screen                       | Link                      |
|------------------------------|---------------------------|
| Welcome screen               | <a href="#">Configure</a> |
| License / End-User Agreement | <a href="#">Configure</a> |
| Readme (before installation) | <a href="#">Configure</a> |
| Readme (after installation)  | <a href="#">Configure</a> |
| Final screen                 | <a href="#">Configure</a> |

- **Set, clear, or copy** content between screens using the [Rich Content Editor](#)
- Support for template variables:

| Variable       | Description         |
|----------------|---------------------|
| [ARCHIVETITLE] | Package title       |
| [COMPANYNAME]  | Company name        |
| [COMPANYEMAIL] | Contact email       |
| [HOMEPAGE]     | Homepage URL        |
| [COPYRIGHT]    | Copyright notice    |
| [DESCRIPTION]  | Package description |

## 53.8 Custom Actions

The AI has full access to the [custom action system](#):

| Capability           | Details   |
|----------------------|---|
| <b>Query actions</b> | List and inspect actions across all 15 events (Initialization, Before Extraction, After Extraction, Finalization, etc.) |

| Capability               | Details   |
|--------------------------|---|
| <b>Add actions</b>       | Choose from over <b>60 action types</b> — shortcuts, registry, conditions, dialogs, file operations, and more |
| <b>Update properties</b> | Modify any published property via its exact name  |
| <b>Organize</b>          | Delete, move, duplicate, and enable/disable actions   |
| <b>Validate</b>          | Check If/Then nesting, For/Next loops, and required property completeness                                     |

### Note

Custom action tools are dynamically loaded based on context. They activate when the Custom Actions editor is open or when you mention actions, events, or scripts in your prompt.

## 53.9 Documentation Search

The AI can perform **full-text search** across the entire Paquet Builder documentation. It searches the docs before answering how-to questions, ensuring answers are grounded in official documentation rather than guesswork.

Simply ask "How do I...?" and the AI will look up the relevant help topic.

## 53.10 UI Navigation

The AI can open pages and dialogs directly in the application:

- **Open any settings page** — compression, signing, scope, output, and all others
- **Open the File Manager** — for managing files and components
- **Open the Custom Actions editor** — for working with custom actions
- Propose opening **modal editors** via action cards (so you can confirm before the dialog appears)

## 53.11 Application Actions

| Action                    | Description  |
|---------------------------|--|
| <b>Compile</b>            | Build the project into an installer package        |
| <b>Full Rebuild</b>       | Force a complete rebuild (ignoring cache)          |
| <b>Save</b>               | Save the current project                           |
| <b>Run</b>                | Execute the compiled package                       |
| <b>Quick Test</b>         | Build without compression or signing (for testing) |
| <b>Open Output Folder</b> | Open the output directory in Windows Explorer      |

After compilation, the AI can also retrieve and analyze the **build log** to help troubleshoot errors and warnings.

## 53.12 Screenshot Capture

Send a **screenshot of the current window** to the AI for visual context analysis. This is useful when:

- You need help understanding a dialog or settings page
- You want the AI to see what you're looking at
- You're troubleshooting a visual issue in your installer preview

The screenshot captures the active settings page or dialog window.

## 53.13 Complete Tool Reference

| Tool                                | Description   |
|-------------------------------------|---|
| <code>get_project_info</code>       | Comprehensive overview of the current project               |
| <code>get_setting</code>            | Read the value of any project setting                       |
| <code>set_setting</code>            | Propose a change to a project setting                       |
| <code>get_setting_info</code>       | Detailed info about a setting (type, effects, dependencies) |
| <code>find_relevant_settings</code> | Search settings by intent (English or French)               |
| <code>navigate_to_page</code>       | Open a settings page or dialog in the application           |

| Tool                           | Description   |
|--------------------------------|---|
| <code>collection_query</code>  | Query files, variables, localization, registry, shortcuts, or screens |
| <code>collection_mutate</code> | Add, update, or delete items in any collection                        |

| Tool                                 | Description  |
|--------------------------------------|--|
| <code>custom_actions_query</code>    | List and inspect custom actions across all events      |
| <code>custom_actions_mutate</code>   | Add, update, delete, move, or duplicate custom actions |
| <code>custom_actions_validate</code> | Validate action configuration and control flow         |

| Tool                         | Description                                   |
|------------------------------|---|
| <code>search_doc</code>      | Full-text search across product documentation |
| <code>perform_action</code>  | Trigger compile, save, run, or other actions  |
| <code>send_screenshot</code> | Capture and analyze a screenshot              |

| Tool       | Description                 |
|------------|-----------------------------|
| filesystem | Read or write files on disk |

## 53.14 Related Topics

### Using the AI Assistant →

Examples and Tips.

### Setting Up the AI Assistant →

Provider and model configuration.

### MCP Server Tool Reference →

Same tools available for headless automation.

### List of Custom Actions →

All 60+ action types the AI can create.

### About Directive Files →

Alternative automation via script files.

### Global Variable List →

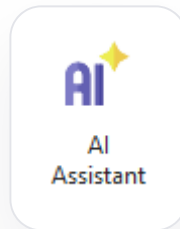
All predefined runtime variables.

## 54. Using the AI Assistant — Examples and Tips

This page covers the day-to-day use of the AI Assistant with plenty of real-world prompt examples to get you started quickly.

### 54.1 Opening the AI Assistant

- 1 **Open or create a project** in Paquet Builder.
- 2 **Click the AI Assistant button** in the **Project Edition** ribbon toolbar.
- 3 The assistant opens as a floating window that you can move and resize freely.



#### **Note**

The AI Assistant works with the currently loaded project. Make sure to open or create a project before asking configuration questions.

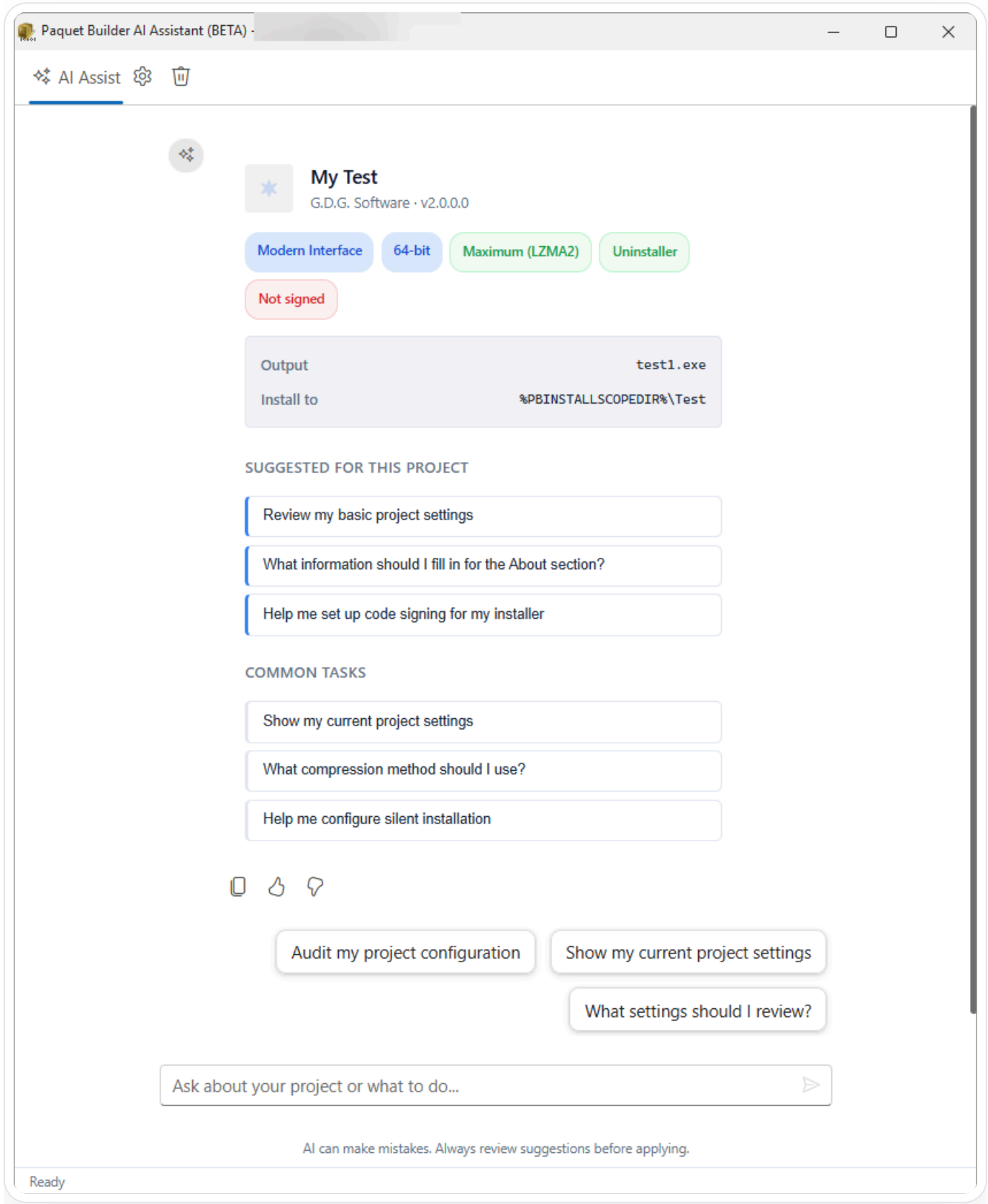
### 54.2 The Welcome Screen

When you open the assistant with a project loaded, it displays a contextual **welcome screen** with:

| Element                  | Description  |
|--------------------------|--|
| <b>Project summary</b>   | Name, company, version   |
| <b>Status indicators</b> | Interface type (wizard/standard), architecture (32-bit/64-bit/ARM64), compression, signing, encryption |
| <b>Output details</b>    | Output file path and installation directory  |
| <b>Suggested prompts</b> | Context-aware suggestions based on your current settings page and project state                        |

| Element             | Description  |
|---------------------|--|
| <b>Common tasks</b> | General-purpose suggestion chips for frequent operations |

The welcome screen refreshes each time you re-open the assistant and clears automatically when you send your first message.



## 54.3 Prompt Examples

Here are dozens of example prompts organized by category. You can type these (or similar requests) directly in the AI Assistant chat.

### 54.3.1 Package Configuration — About Your Package

*“Change the package title to My Application 2.0”*

*“Set the company name to Acme Software and the email to [support@acme.com](mailto:support@acme.com)”*

*“Update the product version to 3.1.0 and the file version to 3.1.0.0”*

*“Make this a wizard-style installer instead of standard”*

*“Set the destination folder to %PROGFILES\_DIR%\Acme\MyApp”*

*“Change the package icon to C:\MyProject\resources\app.ico”*

### 54.3.2 Compression Settings — Compression Options

*“Enable LZMA2 compression with ultra level”*

*“Use 4 CPU threads for compression”*

*“What compression settings do you recommend for a 2 GB installer?”*

*“Switch to solid compression for smaller file size”*

*“Disable compression entirely”*

### 54.3.3 Code Signing — Digital Signature

*“Enable digital signing with my PFX file at C:\certs\mykey.pfx”*

*“Set up SHA-256 only signing (no dual signing)”*

*“Configure Azure Artifact Signing with endpoint <https://eus.codesigning.azure.net>”*

*“Enable signing for both the installer and uninstaller”*

*“Set up JSign with my Azure Key Vault”*

### 54.3.4 Building and Testing — Package Output

*“Compile my project”*

*“What errors did the last build produce?”*

*“Do a quick test build”*

*“Open the output folder”*

*"Save my project"*

---

### 54.3.5 Files and Components — **File Manager / Components**

*"Show me all components in my project"*

*"Create a new component called Documentation with description 'User guides and manuals'"*

*"Link the Plugins component to C:\MyProject\plugins with subfolders enabled"*

*"Add a file mask to exclude \*.pdb and \*.log files from the Main component"*

*"What files are in the Documentation component?"*

*"Remove the Extras component from my project"*

### 54.3.6 Variables and Localization — **Variables / Resource Strings**

*"Add a boolean variable called INSTALL\_PLUGINS with default value 1"*

*"List all my project variables"*

*"Translate all resource strings to French and German"*

*"Add a new resource string #WelcomeText with value 'Welcome to the installer'"*

*"What languages are configured in my project?"*

### 54.3.7 Installation Scope — **Installation Scope**

*"Set the installation scope to User Choice so end users can pick"*

*"Configure per-user installation (current user only)"*

*"What is the current installation scope?"*

*"Enable UAC elevation for all-users installation"*

### 54.3.8 Custom Actions — **Action Manager / Action List**

*"Add a shortcut on the Desktop after file extraction pointing to %DESTPATH%\MyApp.exe"*

*"Create a registry key HKCU\Software\Acme\MyApp with a Version string value set to 2.0"*

*"Add a condition to check if .NET 8 Runtime is installed before extraction"*

*"Show me all actions in the After Extraction event"*

*"Add a license agreement dialog before file extraction"*

*"Disable all actions in the Finalization event"*

*"Create an environment variable MYAPP\_HOME pointing to %DESTPATH%"*

---

### 54.3.9 Navigation

*"Open the compression settings"*

*"Open the File Manager"*

*"Open the Custom Actions editor"*

*"Go to the digital signature settings"*

*"Show me the installation scope settings"*

### 54.3.10 Project Information

*"Show me a summary of my project settings"*

*"What is the current output path?"*

*"What architecture is my package targeting?"*

*"Is my installer digitally signed?"*

*"How many files are in my project?"*

### 54.3.11 Troubleshooting and How-To

*"Why is my installer requesting admin rights?"*

*"How do I make a silent installer?"*

*"How do I create a multi-language installer?"*

*"What is the difference between LZMA and LZMA2?"*

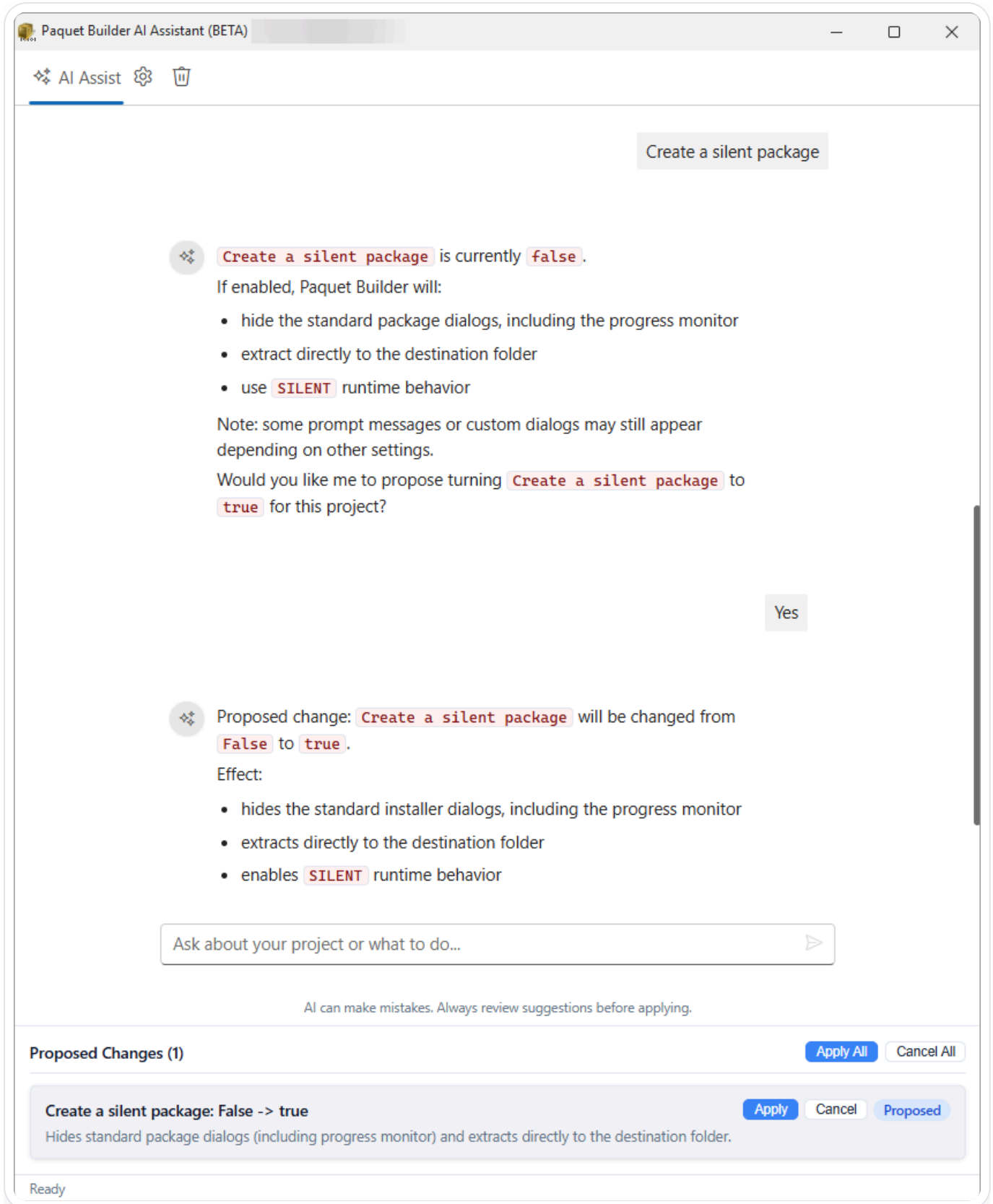
*"How do I add an environment variable during installation?"*

*"How can I check if a process is running before extraction?"*

---

## 54.4 Reviewing and Applying Changes

When the AI proposes changes, they appear as **action cards** in the chat:



Each card shows:

- **What** will change (setting name, component, action, etc.)
- The **proposed value**
- **Apply** and **Cancel** buttons

Click **Apply** on a specific card to accept that change, or **Cancel** to reject it.

Use **Apply All** to accept every pending change at once, or **Cancel All** to reject them all.

### Tip

Review each proposed change before applying. The AI provides descriptions to help you understand the impact of each modification.

### Caution

Potentially destructive actions (like deleting components or clearing all actions in an event) are flagged with a warning badge. Pay extra attention to these before clicking Apply.

## 54.5 Context-Aware Suggestions

After applying changes, the AI may display **follow-up suggestion chips** — clickable prompts for related tasks. For example:

| After...           | Suggestions  |
|--------------------|--|
| Adding a component | "Link a source folder to this component", "Set a file mask"  |
| Enabling signing   | "Configure the timestamp server", "Sign the uninstaller too" |
| Adding a shortcut  | "Add another shortcut", "Set the shortcut icon"              |

Click any chip to send it as your next prompt.

## 54.6 Tips for Best Results

- 1 Be specific** — "Set LZMA2 compression with ultra level and 4 threads" works better than "make compression better".
- 2 Navigate first** — Open the relevant settings page before asking. The AI detects your current page and provides targeted suggestions.

- 3 **Use screenshots** — Ask the AI to capture a screenshot when you need help understanding a visual element.
- 4 **Ask follow-up questions** — The AI maintains conversation history, so you can refine your requests incrementally.
- 5 **Ask how-to questions** — The AI searches the official documentation to give accurate answers.
- 6 **Use both languages** — The AI understands prompts in English and French.

## 54.7 Dark and Light Theme

The AI assistant window follows your application theme automatically. Switch themes in Paquet Builder and the assistant adapts instantly.

## 54.8 Clearing the Conversation

Click the **clear** button to reset the conversation history and all pending action cards. The welcome screen reappears with updated project information.

## 54.9 Related Topics

### What the AI Assistant Can Do

Full capabilities reference.



### Setting Up the AI Assistant

Provider and model configuration.



### AI Providers and Models

Choosing the right model.



### Using Local AI Models

Ollama, LM Studio, llama.cpp.



## MCP Server



Headless alternative for AI-driven automation.

## Create a Simple Package



Quick tutorial to follow along with the AI.

## 55. Using Local AI Models

The AI Assistant can connect to locally hosted AI models instead of cloud services. This offers several advantages:

- **Privacy** — Your project data never leaves your machine
- **No API costs** — Run as many queries as you want for free
- **Offline use** — Works without an internet connection
- **Full control** — Choose and customize your own models

### 55.1 Supported Local Providers

| Software         | Provider Setting | Default Host | Default Port |
|------------------|------------------|--------------|--------------|
| Ollama           | Ollama           | localhost    | 11434        |
| LM Studio        | LlamaCpp         | localhost    | 1234         |
| llama.cpp server | LlamaCpp         | localhost    | 8080         |

### 55.2 Setting Up a Local Provider

**Ollama** is the easiest way to run local AI models on your machine.

### 1 Install Ollama

Download and install Ollama from [ollama.com](https://ollama.com).

### 2 Pull a model

Open a terminal and download a model that supports function calling:

```
ollama pull llama3
```

### 3 Configure Paquet Builder

- Open the AI Settings dialog in Paquet Builder
- Select **Ollama** as the provider
- The default host (`localhost`) and port (`11434`) work automatically
- Enter the model name (e.g., `llama3`) in the **Model** field

### 4 Start chatting

The AI Assistant will connect to your local Ollama instance. Ollama starts automatically when a request is received.



## Tip

For best results with function calling, use models that explicitly support tool use: `llama3`, `qwen2.5`, `mistral`.

**LM Studio** provides a user-friendly desktop application for downloading and running local models.

### 1 Install LM Studio

Download and install from [lmstudio.ai](https://lmstudio.ai).

### 2 Download a model

Use the built-in model browser to search and download a model. Look for models labeled with “tool use” or “function calling” support.

### 3 Start the local server

- Go to the **Local Server** tab in LM Studio
- Load your downloaded model
- Click **Start Server** — it will listen on port **1234** by default

### 4 Configure Paquet Builder

- Open AI Settings
- Select **LlamaCpp** as the provider
- Set the port to **1234**
- The Model field can be left empty (LM Studio uses the currently loaded model)

For advanced users, [llama.cpp](#) provides a lightweight, high-performance server.

### 1 Download or build llama.cpp

Get the latest release from the GitHub repository or build from source.

### 2 Start the server

```
llama-server -m your-model.gguf --port 8080
```

### 3 Configure Paquet Builder

- Select **LlamaCpp** as the provider
- Set the port to **8080** (or whatever port you used)
- Leave the Model field empty

## 55.3 Recommended Local Models

For the AI Assistant's function calling (tool use) to work properly, the model must support structured tool calls. Here are recommended models:

| Model         | Size   | Function Calling | Notes                                |
|---------------|--------|------------------|--------------------------------------|
| Llama 3 (8B)  | ~5 GB  | Good             | Best balance of speed and capability |
| Llama 3 (70B) | ~40 GB | Very Good        | Requires 48+ GB RAM or powerful GPU  |

| Model         | Size  | Function Calling | Notes                           |
|---------------|-------|------------------|---------------------------------|
| Qwen 2.5 (7B) | ~4 GB | Good             | Strong tool use, fast inference |
| Mistral (7B)  | ~4 GB | Good             | Reliable function calling       |

### Note

Model sizes refer to the GGUF quantized versions typically used with Ollama and llama.cpp. Actual download size depends on the quantization level (Q4, Q5, Q8, etc.).

## 55.4 Limitations of Local Models

### Caution

Local AI models may produce less reliable results than cloud providers, especially for complex multi-step operations:

- **Weaker function calling** — Some models may not correctly format tool calls, leading to errors or missed actions
- **Less context understanding** — Smaller models may struggle with nuanced requests
- **Slower responses** — Speed depends on your hardware (CPU/GPU, RAM)
- **No multimodal support** — Screenshot analysis may not work with most local models

If the assistant frequently fails to use its tools correctly, try a larger model or switch to a cloud provider for complex tasks.

## 55.5 Custom Host and Port

If your local AI server runs on a different machine or a custom port:

- 1 Open AI Settings
- 2 Modify the **Host** field (e.g., `192.168.1.100` for a server on your local network)
- 3 Modify the **Port** field as needed

### Note

When connecting to a remote host, ensure the AI server is accessible from your machine and that no firewall blocks the connection on the configured port.

## 55.6 Hybrid Approach

You can switch between local and cloud providers at any time by changing the provider in AI Settings. A practical approach is to:

- Use a **local model** for quick, simple tasks (changing a setting, listing components)
- Switch to a **cloud provider** for complex tasks (building custom action workflows, full project setup)

## 55.7 Related Topics

### Setting Up the AI Assistant →

General configuration guide

### AI Providers and Models →

Cloud provider comparison

### Using the AI Assistant →

Prompt examples and tips

### What the AI Assistant Can Do →

Full capabilities reference

### Environment Options →

Global application settings

## 56. MCP Server Overview

---

**PBMCPServer.exe** is a standalone console application that exposes Paquet Builder's full project engine via the **Model Context Protocol (MCP)**. It allows external AI tools — such as Claude Code, Cursor, Windsurf, or any MCP-compatible client — to open, modify, compile, and save installer projects **programmatically, without a GUI**.

Here, Claude Code is connected to the Paquet Builder MCP Server and updates all version numbers, copyright, and other project settings in a single command through a natural language conversation:

Chat Cowork Code

Open Paquet Builder project file ▾

Can you open the Paquet Builder project "████████████████████\pb3cmdsetup1x64.pbpx" using the appropriate MCP server?

> Used the pbmcpserver integration

Project is now open: Paquet Builder Console Compiler 64-bit (pb3cmdsetup1x64.pbpx).

Update all needed fields to version 2026

Let me first check the current proj info and find all version-related settings.

> Used 2 tools

Let me get the current value of all version-related settings, plus the copyright and output filename.

> Used 8 tools

Here's what needs updating from 2025 to 2026:

| Setting         | Current                  | New                     |
|-----------------|--------------------------|-------------------------|
| File version    | 25.2.0.0                 | 26.0.0.0                |
| Product version | 25.2.0.0                 | 26.0.0.0                |
| Output filename | pbcmdompack2025-2x64.exe | pbcmdcompack2026x64.exe |
| Copyright       | ...2013-2025...          | ...2013-2026...         |

Applying all changes now.

> Used 4 tools

All four fields have been updated to version 2026:

- File version: 25.2.0.0 → 26.0.0.0
- Product version: 25.2.0.0 → 26.0.0.0
- Output filename: pbcmdompack2025-2x64.exe → pbcmdcompack2026x64.exe
- Copyright: 2013-2025 → 2013-2026

The project has not been saved yet — let me know if you'd like me to save it.

Reply...

+ 🤖 Request permissions ▾

Opus 4.6 1M ▾



## 56.1 What is MCP?

The **Model Context Protocol** (MCP) is an open standard that enables AI assistants to connect to external tools and data sources. It uses JSON-RPC 2.0 messages exchanged over standard input/output (stdio). Paquet Builder implements the **MCP 2025-03-26** specification.

## 56.2 Use Cases

| Use Case                          | Description  |
|-----------------------------------|--|
| <b>AI-driven project creation</b> | Use Claude Code or Cursor to set up a complete installer project via natural language conversation |
| <b>CI/CD automation</b>           | Integrate installer builds into your deployment pipeline   |
| <b>Batch modifications</b>        | Apply changes to multiple projects programmatically  |
| <b>Headless compilation</b>       | Compile installers on a build server without opening the GUI                                       |
| <b>Scripted workflows</b>         | Combine MCP tools into complex automation sequences  |

## 56.3 AI Assistant vs. MCP Server

Paquet Builder offers two AI integration points. Choose the one that fits your workflow:

| Feature                    | Embedded AI Assistant        | MCP Server                  |
|----------------------------|------------------------------|-----------------------------|
| <b>Interface</b>           | Built-in chat window         | External AI client          |
| <b>Requires GUI</b>        | Yes (Paquet Builder open)    | No (fully headless)         |
| <b>Change confirmation</b> | Action cards (Apply/Cancel)  | Direct mutations            |
| <b>Project saving</b>      | Manual (click Save)          | Automatic after each change |
| <b>AI provider</b>         | Configured in Paquet Builder | Managed by the AI client    |

| Feature         | Embedded AI Assistant           | MCP Server                       |
|-----------------|---------------------------------|----------------------------------|
| <b>Best for</b> | Interactive, conversational use | Automation, CI/CD, scripted work |

### Note

Unlike the embedded **AI Assistant** which proposes changes as reviewable action cards, the MCP server applies all modifications immediately. The AI client (e.g., Claude Code) manages any confirmation workflow on its side.

## 56.4 Available Tools (20+)

The MCP server exposes over 20 tools organized into categories:

| Tool                 | Description                          |
|----------------------|--------------------------------------|
| open_project         | Open an existing .pbpx project file  |
| close_project        | Close the current project            |
| save_project         | Save the project to disk             |
| new_project          | Create a new blank project           |
| list_recent_projects | List recently opened project files   |
| get_project_info     | Get a comprehensive project overview |

| Tool                   | Description   |
|------------------------|---|
| get_setting            | Read any project setting                                      |
| set_setting            | Modify a project setting                                      |
| get_setting_info       | Get documentation for a setting (type, effects, dependencies) |
| find_relevant_settings | Search settings by natural language intent                    |

| Tool              | Description   |
|-------------------|---|
| collection_query  | Query files, variables, localization, registry, shortcuts, or screens |
| collection_mutate | Add, update, or delete items in any collection                        |

| Tool                  | Description                                     |
|-----------------------|---|
| custom_actions_query  | List and inspect custom actions                 |
| custom_actions_mutate | Add, update, delete, move, or duplicate actions |

| Tool                    | Description                                    |
|-------------------------|--|
| custom_actions_validate | Validate action configuration and control flow |

| Tool              | Description   |
|-------------------|---|
| compile           | Compile the project into an installer                 |
| get_build_log     | Retrieve the compilation log with errors and warnings |
| search_doc        | Full-text search across product documentation         |
| filesystem        | Read or write files on disk                           |
| execute_directive | Run a <b>directive file</b> (.pbd)                    |

### Tip

For full parameter documentation on each tool, see the [MCP Server Tool Reference](#).

## 56.5 Getting Started

### Note

The MCP server is a **free add-on** available for all Paquet Builder editions (Freeware, Trial, Pro, and Ultimate). Download it from **Help** → **Check for Updates & Addons** in Paquet Builder, or from the [MCP Server download page](#).

- 1 **Install Paquet Builder** (GUI) on the target computer.
- 2 **Download the MCP server** — In Paquet Builder, go to **Help** → **Check for Updates & Addons** to download the MCP server installer. You can also download it directly from the [MCP Server download page](#).
- 3 **Run the MCP server installer.** It will install `PBMCPServer.exe` into the Paquet Builder installation folder.
- 4 **Configure your AI client** — Add the MCP server to Claude Code, Cursor, or Windsurf settings.

## 5 Start chatting — Ask the AI to open a project and make changes.

### Edition restrictions

The MCP server works with every edition, but the same restrictions apply as in the GUI:

- **Trial:** compiled packages display a “Trial Edition” message box
- **Freeware:** custom actions are ignored during compilation, and some features (uninstaller, encryption, component selection) are disabled
- **Pro / Ultimate:** no restrictions

### Set up the MCP Server [→](#)

Step-by-step configuration for Claude Code, Cursor, and Windsurf.

## 56.6 Related Topics

### Full tool reference [→](#)

All 20+ tools with parameters.

### AI Assistant Overview [→](#)

Built-in chat-based alternative with visual action cards.

### About Directive Files [→](#)

Script-based automation with .pbd files.

### Console Command-line Compiler [→](#)

Batch compilation without GUI.

### Command-line Options [→](#)

Paquet Builder command-line switches.

## 57. Setting Up the MCP Server

---

This guide shows you how to connect the Paquet Builder MCP server to your preferred AI coding assistant.

### 57.1 How to Obtain the MCP Server

The MCP server is a **free add-on** for all Paquet Builder editions (Freeware, Trial, Pro, and Ultimate).

### 57.2 Installation

- 1 **Install Paquet Builder** (GUI) on the target computer.
- 2 **Download the MCP server** — In Paquet Builder, go to **Help** → **Check for Updates & Addons** to download the MCP server installer. You can also download it directly from the [MCP Server download page](#).
- 3 **Run the MCP server installer.** It will install `PBMCPServer.exe` into the Paquet Builder installation folder.

The MCP server is now ready to use by AI clients.

#### Note

You also need an MCP-compatible AI client such as Claude Code, Cursor, or Windsurf.

### 57.3 Configuration

Add the following to your project's `.mcp.json` file or to Claude Code's global MCP settings:

```
{
  "mcpServers": {
    "pbmcpserver": {
      "type": "stdio",
      "command": "C:\\Program Files\\Paquet Builder\\PBMCPServer.exe",
      "args": []
    }
  }
}
```

In Cursor, open the MCP settings (Settings → MCP Servers) and add a new server:

```
{
  "mcpServers": {
    "paquet-builder": {
      "command": "C:\\Program Files\\Paquet Builder\\PBMCPServer.exe",
      "args": []
    }
  }
}
```

In Windsurf's MCP configuration:

```
{
  "mcpServers": {
    "paquet-builder": {
      "command": "C:\\Program Files\\Paquet Builder\\PBMCPServer.exe",
      "args": []
    }
  }
}
```

The MCP server uses the **stdio transport** — it communicates via standard input and output using newline-delimited JSON-RPC 2.0 messages with UTF-8 encoding.

To integrate with any MCP client:

- 1 Start `PBMCPServer.exe` as a child process
- 2 Send JSON-RPC requests to its **stdin**
- 3 Read JSON-RPC responses from its **stdout**

### Tip

Adjust the path to match your actual MCP server installation directory. Use double backslashes (`\\`) in JSON paths on Windows.

## 57.4 Verifying the Connection

Once configured, your AI client should be able to list the available Paquet Builder tools. Try asking:

*"List the available Paquet Builder tools"*

Or open a project:

*"Open the project at C:|MyProject|installer.pbpX"*

If the MCP server is correctly configured, the AI will use the `open_project` tool and report the project details.

## 57.5 Typical Workflow

A typical MCP session follows this pattern:

- 1 **Open a project** — `open_project` with the path to a `.pbpx` file
- 2 **Query current state** — `get_project_info`, `get_setting`, `collection_query`
- 3 **Make modifications** — `set_setting`, `collection_mutate`, `custom_actions_mutate`
- 4 **Compile** — `compile`, then `get_build_log` to check results
- 5 **Close** — `close_project`

## Note

The project is **automatically saved** after every modification (`set_setting`, `collection_mutate`, `custom_actions_mutate`, `new_project`). You do not need to call `save_project` explicitly between changes.

## 57.6 Example: Creating an Installer with Claude Code

Here's an example conversation with Claude Code using the MCP server:

**You:** *Open my project at `D:\Projects\myapp.pbpx` and show me its current settings*

**Claude Code:** *Uses `open_project` and `get_project_info` — Your project "My Application" targets x64, uses LZMA2 compression with ultra level. Output: `D:\Output\Setup.exe`. 3 components, 142 files. Not signed.*

**You:** *Enable code signing with SHA-256 and sign the uninstaller too*

**Claude Code:** *Uses `set_setting` for `digitalsign`, `signmethod`, and `csignuninst` — Done. Digital signing is now enabled with SHA-256. The uninstaller will also be signed.*

**You:** *Now compile it and check for errors*

**Claude Code:** *Uses `compile` then `get_build_log` — Build completed successfully. Output: `D:\Output\Setup.exe` (12.4 MB). No errors, 0 warnings.*

## 57.7 Protocol Details

| Property              | Value                          |
|-----------------------|--------------------------------|
| <b>Protocol</b>       | MCP 2025-03-26                 |
| <b>Transport</b>      | stdio (standard input/output)  |
| <b>Message format</b> | Newline-delimited JSON-RPC 2.0 |
| <b>Encoding</b>       | UTF-8                          |

## 57.8 Troubleshooting

### 57.8.1 "Server not found" or "Connection failed"

- Verify that `PBMCPServer.exe` exists at the configured path

- Check that the path uses double backslashes in JSON: `c:\\Program Files\\...`
- Ensure Paquet Builder is installed in the same directory as `PBMCPServer.exe`. The MCP server works with all editions — if no license is found, it runs in Trial mode

## 57.8.2 Tools not responding

- The MCP server loads the full Paquet Builder engine on startup, which may take a few seconds
- Check that no other instance of `PBMCPServer.exe` is already running

## 57.8.3 Build errors

- Use `get_build_log` after `compile` to see detailed error messages
- Ensure all source files referenced by the project are accessible from the build machine

## 57.9 Related Topics

### Full tool reference →

All tools with parameters and examples

### MCP Server Overview →

Use cases and feature comparison

### AI Assistant Overview →

Built-in chat-based alternative

### About Directive Files →

Script-based project automation

### Console Command-line Compiler →

Headless compilation tool

### Create a Project →

Understanding project structure

## 58. MCP Server Tool Reference

---

The Paquet Builder MCP Server exposes over 20 tools for managing installer projects programmatically. This page documents each tool's purpose, parameters, and behavior.

### Note

All mutations are applied immediately and auto-saved. The MCP server does not use action cards or confirmation dialogs — changes take effect as soon as the tool is called.

### 58.1 Tools at a Glance

| Tool                 | Description                          |
|----------------------|--------------------------------------|
| open_project         | Open an existing .pbpx project file  |
| close_project        | Close the current project            |
| save_project         | Save the project to disk             |
| new_project          | Create a new blank project           |
| list_recent_projects | List recently opened project files   |
| get_project_info     | Get a comprehensive project overview |

| Tool                   | Description                         |
|------------------------|-------------------------------------|
| get_setting            | Read any project setting            |
| set_setting            | Modify a project setting            |
| get_setting_info       | Get documentation for a setting     |
| find_relevant_settings | Search settings by natural language |

| Tool              | Description   |
|-------------------|---|
| collection_query  | Query files, variables, localization, registry, shortcuts, or screens |
| collection_mutate | Add, update, or delete items in any collection                        |

| Tool                  | Description                                     |
|-----------------------|---|
| custom_actions_query  | List and inspect custom actions                 |
| custom_actions_mutate | Add, update, delete, move, or duplicate actions |

| Tool                    | Description                                    |
|-------------------------|--|
| custom_actions_validate | Validate action configuration and control flow |

| Tool              | Description                           |
|-------------------|---------------------------------------|
| compile           | Compile the project into an installer |
| get_build_log     | Retrieve the compilation log          |
| search_doc        | Full-text search across documentation |
| filesystem        | Read or write files on disk           |
| execute_directive | Run a directive file (.pbd)           |

## 58.2 Project Management

### 58.2.1 `open_project`

Opens an existing Paquet Builder project file.

| Parameter | Type   | Required | Description                         |
|-----------|--------|----------|-------------------------------------|
| path      | string | Yes      | Full path to the .pbpx project file |

**Returns:** Project summary including title, company, version, output path, and key settings.

### 58.2.2 `close_project`

Closes the currently loaded project. No parameters.

### 58.2.3 `save_project`

Saves the current project to disk. Usually not needed since changes are auto-saved after every mutation.

### 58.2.4 `new_project`

Creates a new blank project.

| Parameter                | Type   | Required | Description                               |
|--------------------------|--------|----------|---|
| <code>name</code>        | string | Yes      | Project title                             |
| <code>output_path</code> | string | No       | Output directory for the compiled package |
| <code>install_dir</code> | string | No       | Default installation directory            |

### 58.2.5 `list_recent_projects`

Returns a list of recently opened project files with their paths. No parameters.

### 58.2.6 `get_project_info`

Returns a comprehensive overview of the current project: title, company, version, output settings, architecture, compression, signing status, component count, variable count, and more. No parameters.

## 58.3 Settings

### 58.3.7 `get_setting`

Reads the current value of a project setting.

| Parameter        | Type   | Required | Description  |
|------------------|--------|----------|--|
| <code>key</code> | string | Yes      | The setting key (e.g., <code>compressionlzma2</code> , <code>digitalsign</code> , <code>title</code> ) |

### 58.3.8 `set_setting`

Modifies a project setting. The change is applied and saved immediately.

| Parameter        | Type   | Required | Description     |
|------------------|--------|----------|-----------------|
| <code>key</code> | string | Yes      | The setting key |

| Parameter | Type   | Required | Description   |
|-----------|--------|----------|---------------|
| value     | string | Yes      | The new value |

### 58.3.9 `get_setting_info`

Returns detailed metadata about a setting: data type, allowed values, effects on other settings, dependencies, warnings, and documentation link.

| Parameter | Type   | Required | Description     |
|-----------|--------|----------|-----------------|
| key       | string | Yes      | The setting key |

### 58.3.10 `find_relevant_settings`

Searches for settings matching a natural language description. Supports both English and French queries.

| Parameter | Type   | Required | Description  |
|-----------|--------|----------|--|
| intent    | string | Yes      | Natural language description (e.g., "compression options", "options de signature") |

**Returns:** A ranked list of matching settings with keys, labels, and descriptions.

## 58.4 Collections

### 58.4.11 `collection_query`

Queries items in one of 6 collection domains.

| Parameter | Type   | Required | Description   |
|-----------|--------|----------|---|
| domain    | string | Yes      | files, variables, localization, registry, shortcuts, or screens |
| query     | string | Yes      | list, details, search, or schema                                |
| id        | string | No       | Item identifier (for details)                                   |

| Parameter          | Type    | Required | Description                                   |
|--------------------|---------|----------|---|
| search             | string  | No       | Search term (for search)                      |
| limit              | number  | No       | Maximum results to return                     |
| include_properties | boolean | No       | Include full property details in list results |

| Domain              | Description   |
|---------------------|---|
| <b>files</b>        | Components and their files (list components, view file lists, search files) |
| <b>variables</b>    | Project variables (list, search, view details)                              |
| <b>localization</b> | Resource strings and languages  |
| <b>registry</b>     | Registry entries created by the installer                                   |
| <b>shortcuts</b>    | Start Menu and Desktop shortcuts  |
| <b>screens</b>      | HTML content for installer screens (Welcome, License, Readme, Final)        |

| Query                | Description  |
|----------------------|--|
| <code>list</code>    | List all items in the domain                                     |
| <code>details</code> | Get full details for a specific item (requires <code>id</code> ) |
| <code>search</code>  | Search for items matching a term (requires <code>search</code> ) |
| <code>schema</code>  | Get the schema describing available properties                   |

### 58.4.12 `collection_mutate`

Adds, updates, or deletes items in a collection. Changes are applied and saved immediately.

| Parameter | Type   | Required | Description  |
|-----------|--------|----------|--|
| domain    | string | Yes      | Collection domain (same as above)                                |
| action    | string | Yes      | add, update, delete, set_content, clear_content, or copy_content |
| targets   | array  | No       | Target items (component names, variable names, etc.)             |
| values    | object | No       | Properties to set or update                                      |
| options   | object | No       | Additional options (varies by domain and action)                 |

## 58.5 Custom Actions

### 58.5.13 `custom_actions_query`

Lists and inspects custom actions across the 15 available events.

| Parameter | Type   | Required | Description  |
|-----------|--------|----------|--|
| query     | string | Yes      | list, details, search, or schema                                 |
| event     | string | No       | Event name (e.g., afterextraction, initialization, finalization) |
| action_id | string | No       | Action identifier (for details)                                  |
| search    | string | No       | Search term  |
| limit     | number | No       | Maximum results  |

#### > All 15 event names

### 58.5.14 `custom_actions_mutate`

Adds, updates, deletes, moves, duplicates, or toggles custom actions.

| Parameter           | Type    | Required | Description  |
|---------------------|---------|----------|--|
| action              | string  | Yes      | add, update, delete, move, duplicate, set_enabled, or clear_event        |
| event               | string  | Yes      | Target event name  |
| action_id           | string  | No       | Target action identifier   |
| action_type         | string  | No       | Action class for add (e.g., TCreateShortcutInfo, TConditionalIfThenInfo) |
| values              | object  | No       | Properties to set  |
| insert              | string  | No       | Insert position: before or after   |
| reference_action_id | string  | No       | Reference action for positioning   |
| enabled             | boolean | No       | Enable/disable state (for set_enabled)                                   |

### 58.5.15 custom\_actions\_validate

Validates custom action configuration and control flow before applying changes.

| Parameter   | Type    | Required | Description   |
|-------------|---------|----------|---|
| scope       | string  | Yes      | action, event, or all   |
| event       | string  | No       | Event to validate   |
| action_id   | string  | No       | Specific action to validate                                       |
| action_type | string  | No       | Action type (for property validation)                             |
| values      | object  | No       | Property values to check  |
| deep        | boolean | No       | Include control flow validation (If/Then nesting, For/Next loops) |

## 58.6 Build

### 58.6.16 `compile`

Triggers a full compilation of the current project. Returns success/failure status with basic build information (output path, file size). No parameters.

### 58.6.17 `get_build_log`

Retrieves the detailed compilation log from the last build, including all phases, errors, warnings, and timing information. No parameters.

## 58.7 Documentation and Filesystem

### 58.7.18 `search_doc`

Performs a full-text search across the Paquet Builder documentation.

| Parameter          | Type   | Required | Description                  |
|--------------------|--------|----------|------------------------------|
| <code>query</code> | string | Yes      | Search query                 |
| <code>limit</code> | number | No       | Maximum results (default: 5) |

**Returns:** Ranked results with file path, section title, text snippet, and relevance score.

### 58.7.19 `filesystem`

Performs file operations on the local filesystem.

| Parameter            | Type   | Required | Description   |
|----------------------|--------|----------|---|
| <code>action</code>  | string | Yes      | <code>read</code> , <code>write</code> , <code>list</code> , or <code>exists</code> |
| <code>path</code>    | string | Yes      | File or directory path  |
| <code>pattern</code> | string | No       | Glob pattern (for <code>list</code> )   |
| <code>content</code> | string | No       | File content (for <code>write</code> )  |

| Parameter | Type    | Required | Description             |
|-----------|---------|----------|-------------------------|
| append    | boolean | No       | Append mode (for write) |
| limit     | number  | No       | Max entries (for list)  |
| offset    | number  | No       | Offset (for list)       |

### 58.7.20 `execute_directive`

Executes a **directive file** (`.pbd`) for batch project creation or modification.

| Parameter | Type   | Required | Description                                       |
|-----------|--------|----------|---|
| path      | string | Yes      | Full path to the <code>.pbd</code> directive file |

## 58.8 Example Session

Here is an example showing how an AI client might use the MCP tools to configure and build an installer:

### 1 Open the project

```
open_project("C:\\Projects\\myapp.pbp")
→ "My Application 1.0 by Acme Software – 3 components, 142 files"
```

### 2 Update settings

```
set_setting("title", "My Application 2.0")
→ Setting updated and saved

set_setting("compressionlzma2", "1")
→ LZMA2 compression enabled
```

### 3 Add a component

```
collection_mutate(domain="files", action="add",
  targets=["Plugins"],
  values={"title": "Plugins", "description": "Optional plugins"})
→ Component "Plugins" created
```

### 4 Add a custom action

```
custom_actions_mutate(action="add", event="afterextraction",
  action_type="TCreateShortcutInfo",
  values={"ShortcutName": "My Application",
    "ShortcutTarget": "%DESTPATH%\myapp.exe"})
→ Shortcut action added to After Extraction
```

### 5 Compile and check

```
compile()
→ Build successful: C:\Output\Setup.exe (12.4 MB)

get_build_log()
→ Full compilation log with timing and status
```

#### Tip

Use `get_project_info` after opening a project to understand its current state before making modifications. Use `find_relevant_settings` when you're not sure of the exact setting key name.

## 58.9 Related Topics

### MCP Server Overview



Use cases and comparison with AI Assistant.

## Setting Up the MCP Server



Configuration for AI clients.

## What the AI Assistant Can Do



Same tools in the embedded chat interface.

## About Directive Files



Script-based project automation with .pbd files.

## Custom Actions Overview



Understanding the custom action system.

## List of Custom Actions



All 60+ available action types.

# 59. Custom Action System

Custom actions allow you to execute runtime logic, such as modifying the registry, checking for files, or calling C functions, without complex scripting. This section covers how the action system works, how to use the editor, and the full action catalog.

## Key Feature

Custom actions are one of the most powerful features of Paquet Builder, covering a wide range of administrative and deployment tasks through a straightforward, event-based interface.

### Overview →

How custom actions and the event-based execution model work.

### Managing Actions →

How to use the Custom Action Manager to organize your logic.

### Editing Properties →

Configure parameters for each action within the properties editor.

### Action Templates →

Reuse logic sequences with built-in or custom templates.

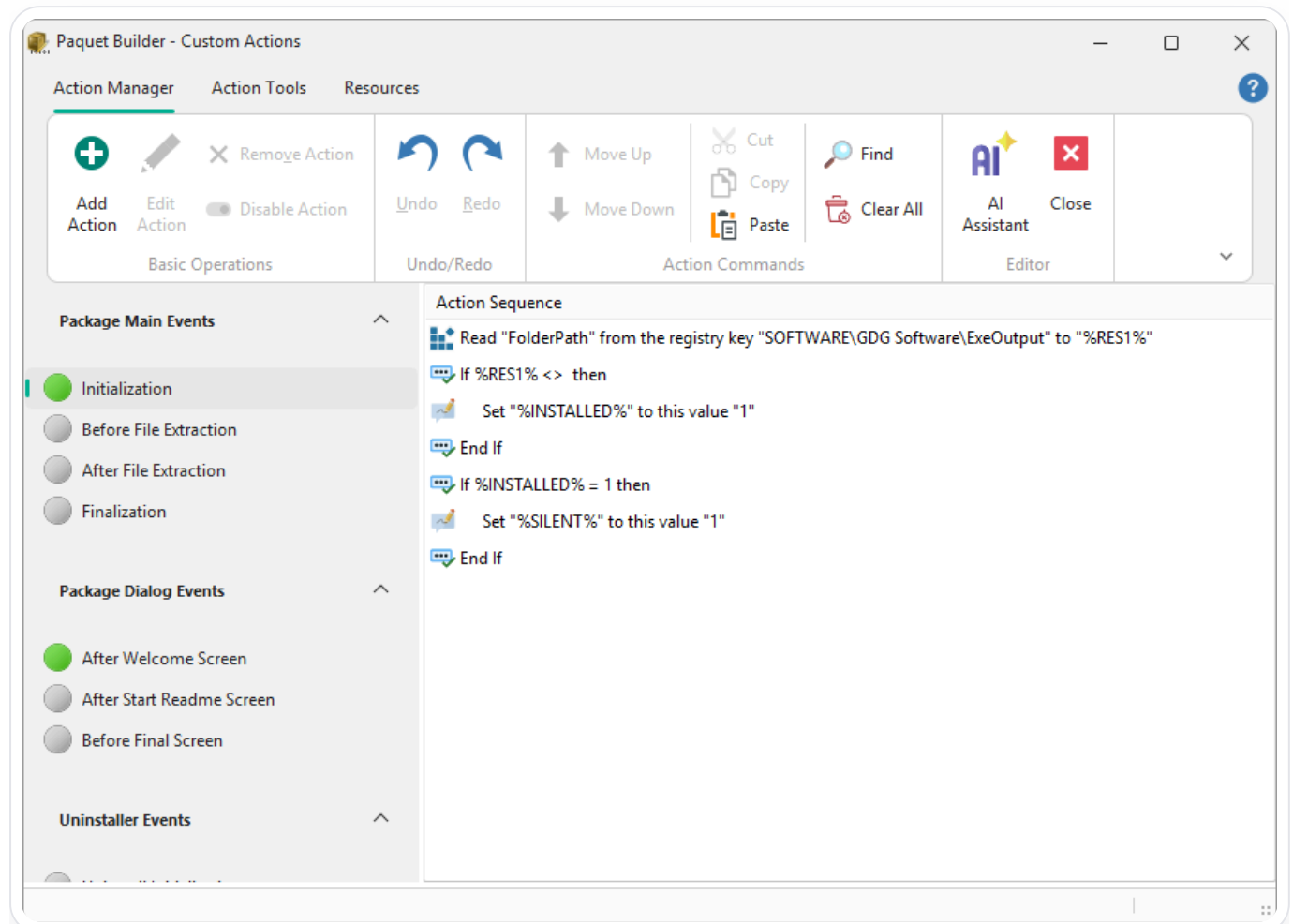
### Action Catalog →

Full list of all built-in custom actions with detailed documentation.

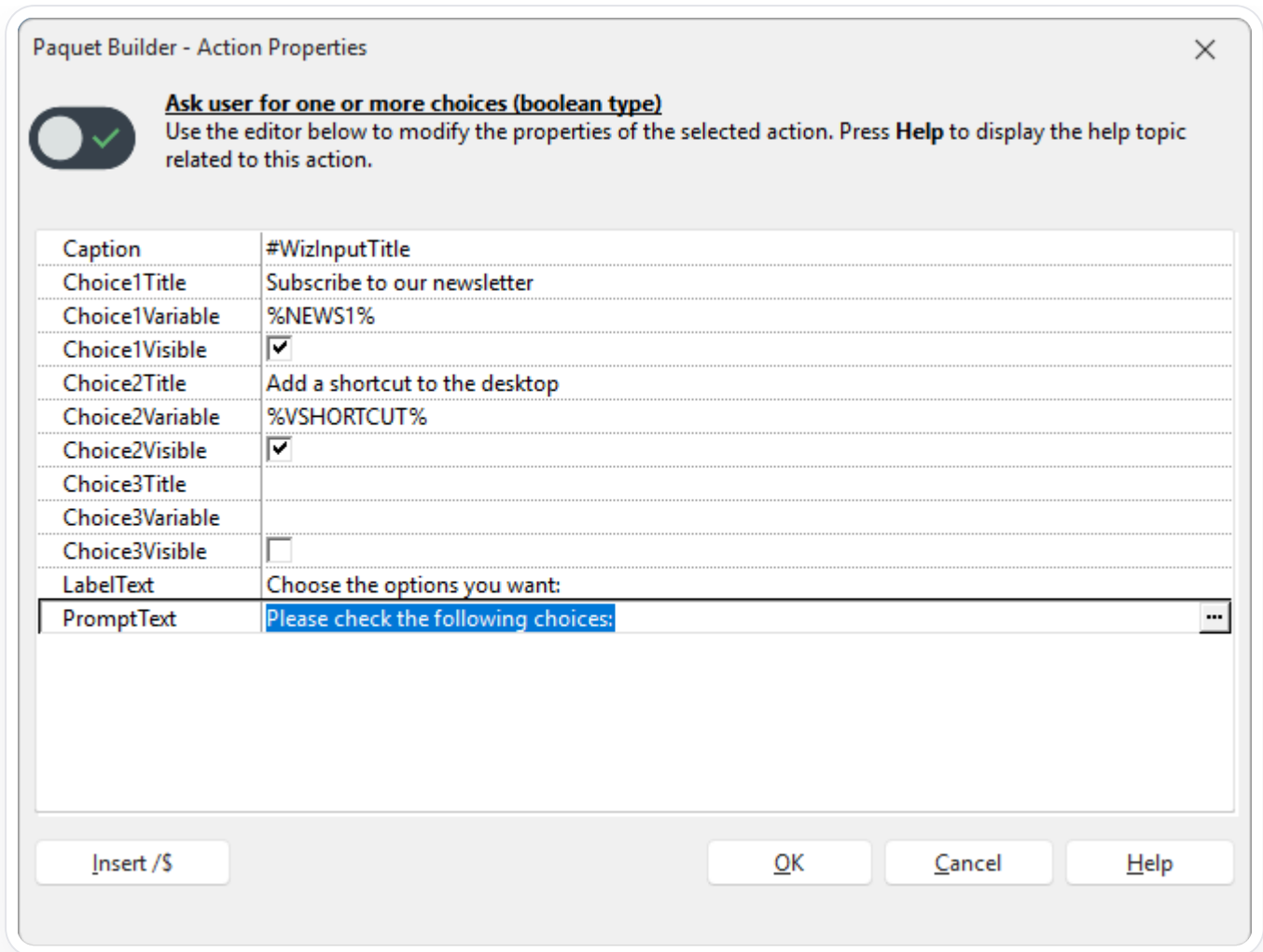
## 60. Custom Actions Overview

Custom actions are the advanced behavior layer of Paquet Builder. They let you extend what happens at runtime without writing scripts by hand.

To add custom actions, click the **Custom Actions** button in the toolbar. This opens the **Custom Action Manager**, where you select an event, click **Add Action**, and choose from the **action catalog**.



Each action exposes properties that you edit through the **Action Properties Editor**.



## 60.1 Why Use Custom Actions



### Drive package logic

Ask questions, set variables, branch with conditions, jump to labels, or stop the package when prerequisites are not met.



### Work with the system

Read or write files, registry keys, XML nodes, INI entries, shortcuts, and environment variables.



### Control the user experience

Show dialogs, readme pages, license agreements, information panels, and wait messages at specific stages.

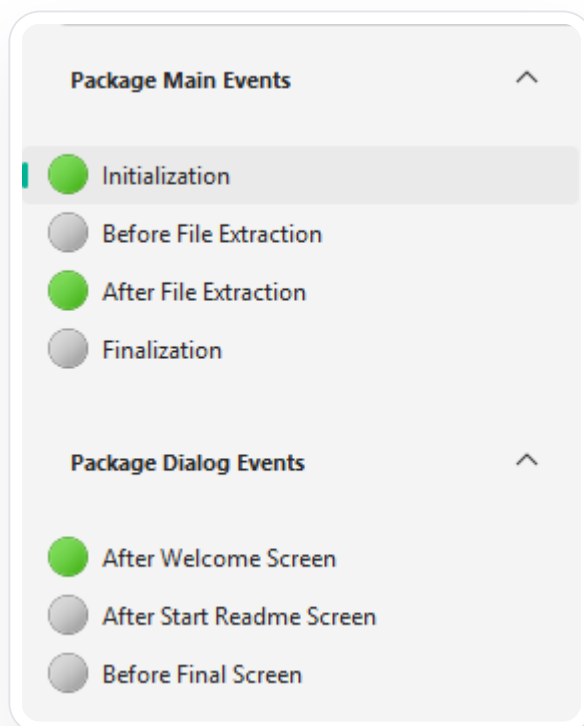
## 60.2 When Actions Run

Custom actions belong to specific execution phases called **events**. Actions inside the same event run in the order they appear in the action list.

- 1 **Initialization** — runs immediately when the package starts; useful for setting variables.
- 2 **After Welcome Screen**
- 3 **After Start Readme Screen**
- 4 **Before File Extraction** — last chance to cancel before files are written to disk.
- 5 **After File Extraction** — files are now on disk.
- 6 **After Shortcut and Registry Creation**
- 7 **After Setup Run** — for setup packages, fires after the child installer finishes.
- 8 **Before Final Screen**
- 9 **Finalization** — fires as the package exits.

In addition to these package events, Paquet Builder also supports dialog-related events and uninstaller events.

Events that contain at least one action show a green LED indicator.



## 60.3 Key Rules to Remember



## Validation is partial

Paquet Builder checks whether required properties are filled in, but it cannot guarantee that the action logic is correct for your scenario.



## Conditions must be balanced

Actions such as **Conditional If/Then/Else** must be paired correctly, including the matching end statement.



## Variables are everywhere

Most action properties can use **variables** and **resource strings** to keep packages dynamic.



## Some actions are property-free

A few actions, such as **Exit Package**, do not open the properties editor because they do not need additional configuration.

To annotate an action list for yourself or your team, use the **User Comment** action.

## 60.4 Undo and Redo

The editor supports **Undo** (Ctrl+Z) and **Redo** (Ctrl+Y) with up to 30 levels. This includes adding, deleting, moving, enabling, disabling, drag-and-drop reordering, paste operations, and changes proposed by the **AI Assistant**.

## 60.5 Modern Rendering for Dialog Actions

Dialog-based actions such as **Show a Readme Dialog**, **Show a License Agreement**, and **Show an Information Panel** use **WebView2** when it is available on the destination system. That gives these dialogs modern HTML and CSS rendering, consistent with the built-in installer screens.

If WebView2 is not installed, Paquet Builder automatically falls back to the legacy Markdown viewer. You can also force the Markdown viewer explicitly by enabling the `ForceMarkdownViewer` property on the action.



### Need examples?

Visit the **G.D.G. Software Forum** for sample projects, practical usage tips, and help with custom action flows.

## 60.6 Explore Further

### Custom Action Manager →

Learn how the action editor is organized and how events are structured.

### Action Properties Editor →

See how individual action parameters are edited and validated.

### List of Custom Actions →

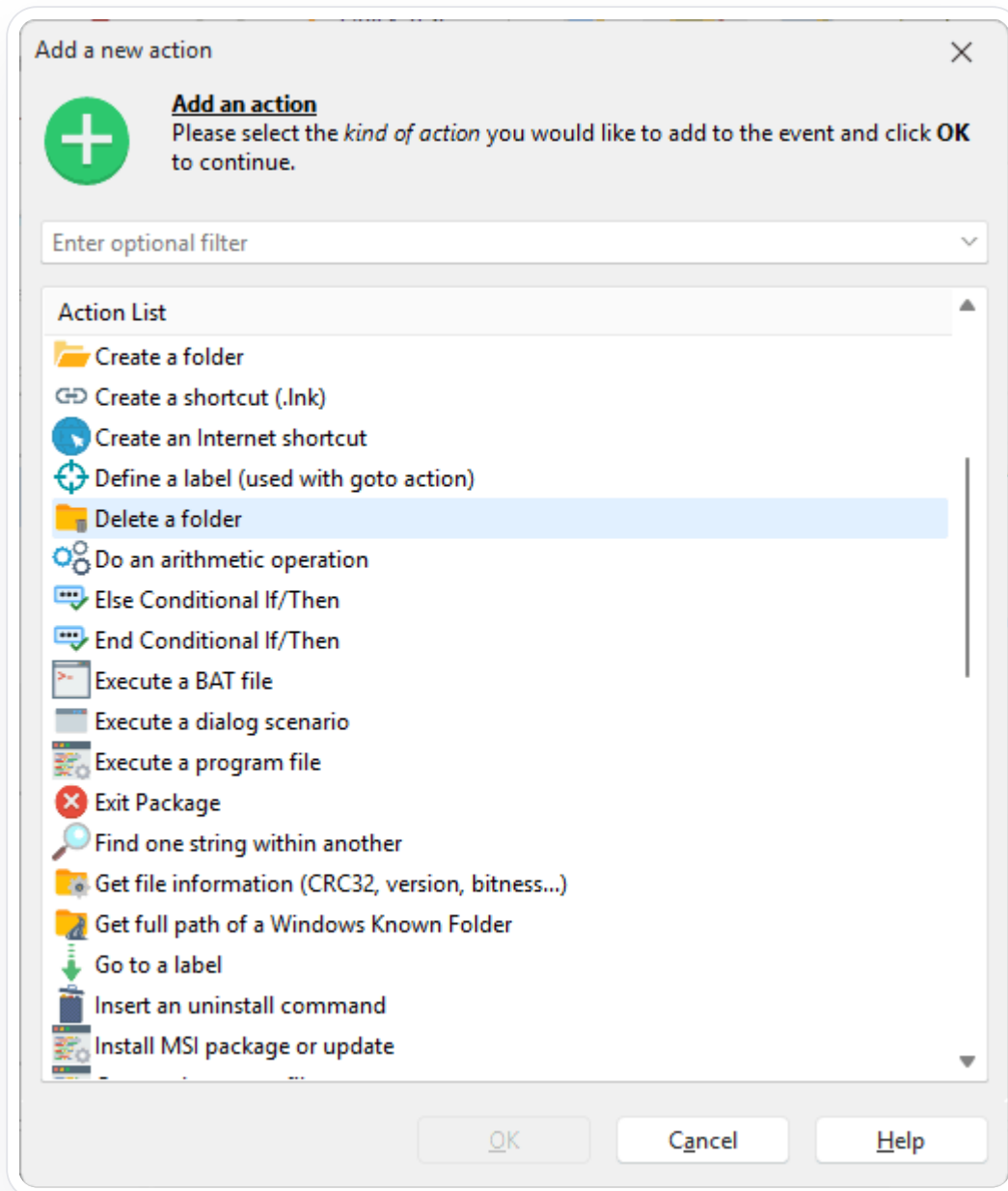
Open the full catalog of built-in actions.

### Templates →

Reuse action templates instead of rebuilding common sequences manually.

# 61. Custom Action Catalog

This catalog lists all built-in custom actions. We recommend reading the [Custom Action System](#) and [Managing Actions](#) guides first.



## Tip

Press the **Help** button in the Action Properties Editor to jump directly to any action's documentation.

## 61.1 User Interface

### Ask User (Text Input)



**Ask User for One Choice (Radio Buttons)** →

**Ask User for Choices (Checkboxes)** →

**Ask User for a Choice (Drop-down)** →

**Ask User for a Folder** →

**Show a Message Box** →

**Show an Information Panel** →

**Show a Readme Dialog** →

**Show a License Agreement** →

**Show/Hide a Wait Message** →

**Execute a Dialog Scenario** →

## 61.2 Variables and Logic

**Modify a Variable** →

**Conditional If/Then/Else** →

**Do an Arithmetic Operation** →

**Find String Within Another** →

**Define a Label** →

**Go to a Label** →

**Exit Package** →

**Sleep (Pause Execution)** →

User Comment →

## 61.3 File System

Check if a File/Folder Exists →

Get File Information →

Copy a File/Folder →

Rename a File →

Remove Files (Single or Wildcard) →

Set File Attributes →

Create a Folder →

Delete a Folder →

Read Data From a File →

Write Text to a File →

Replace Variables in a File →

Get Windows Known Folder Path →

## 61.4 Execution

Execute a Program →

Execute a BAT File →

Open a Document/File →

**Install MSI Package or Update** →

**Call C Function** →

**Check if a Process is Running** →

**Restart Computer** →

## 61.5 Registry and Configuration

**Write a Registry Entry** →

**Read from the Registry** →

**Remove a Registry Key** →

**Check if a Registry Key Exists** →

**Read from an .ini File** →

**Write to an .ini File** →

**Write Environment Variable** →

## 61.6 XML Editing

**XML: Begin Editing File** →

**XML: Read Value from Node** →

**XML: Read Attribute from Node** →

**XML: Write Node** →

**XML: Write Node Attribute** →

**XML: End Editing and Save**



## 61.7 Shortcuts and System

**Create a Shortcut**



**Create an Internet Shortcut**



**Self-register a File (ActiveX)**



**Register a Shared File**



**Refresh Shell Icons**

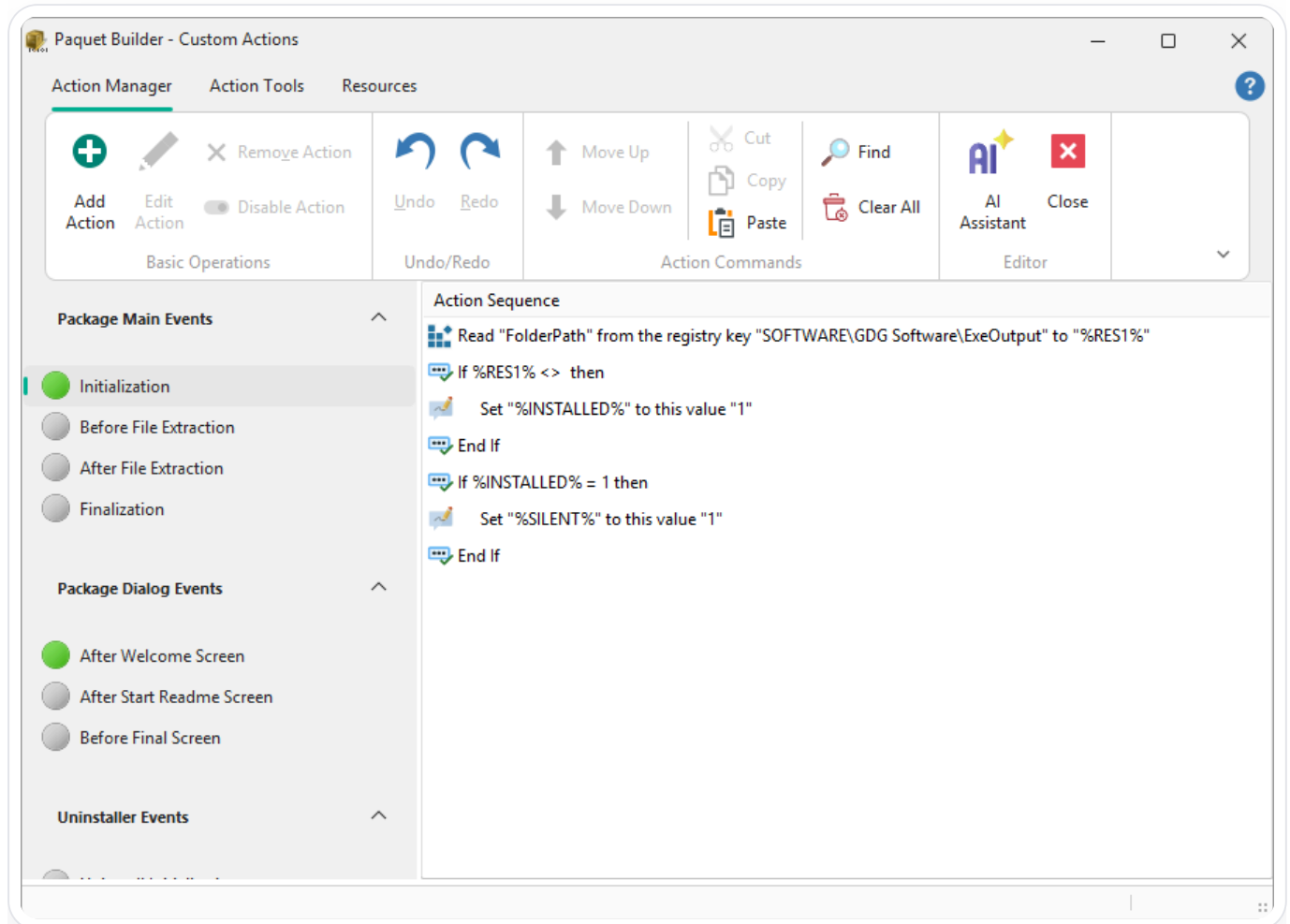


**Insert Uninstall Command**



## 62. Managing Custom Actions

The **Custom Action Manager** is the primary interface for managing logic and operations in your project.



### 62.1 UI Features



#### Add / Remove

Click **Add Action** (Ctrl+A) to insert a new operation. Select an action and press Del to remove it.



#### Edit

Double-click any action to open the [Action Properties Editor](#).



#### Reorder

Use drag-and-drop or the **Move Up/Down** buttons. Logic executes top to bottom.



## Enable / Disable

Toggle the checkbox to bypass an action during compilation. Disabled actions appear crossed out.



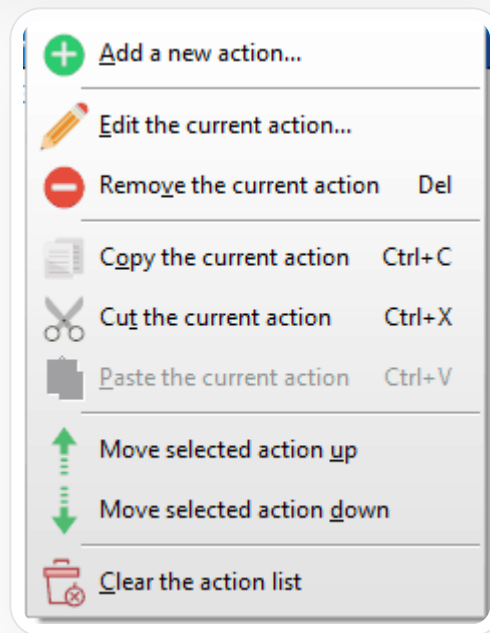
## Search

Use **Find** to locate specific text within your action list. Matches are highlighted.



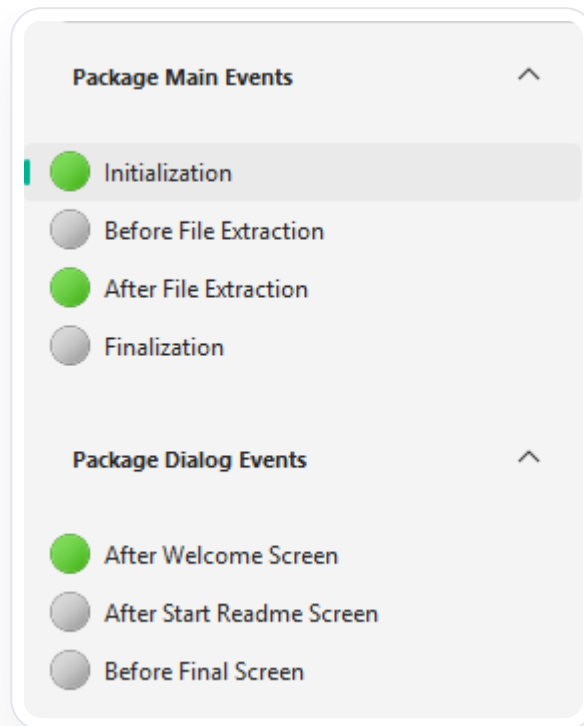
## Context menu

Right-click for cut, copy, paste, and additional commands.



## 62.2 Event Indicators

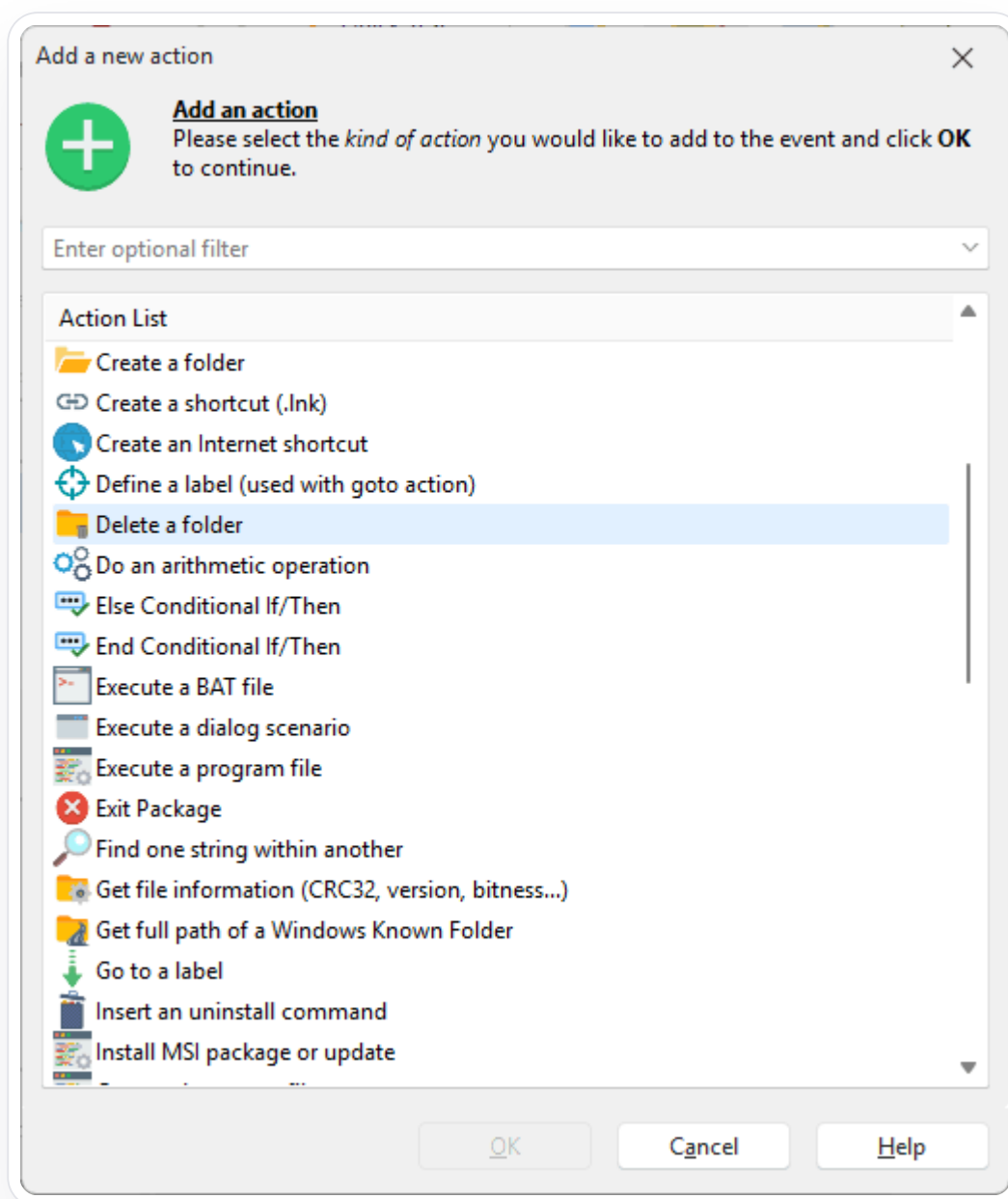
Events containing active logic are marked with a **green LED**. If the LED is off, the event has no custom actions.



## 62.3 Working with Actions

- 1 **Select an Event** — choose the trigger point where the logic should execute.

- 2 **Add the Action** — click **Add Action**, select the desired operation from the list, and click **OK**.



- 3 **Configure Properties** — use the **Properties Editor** to set parameters.

- 4 **Save changes** — click **OK** to return to the manager.

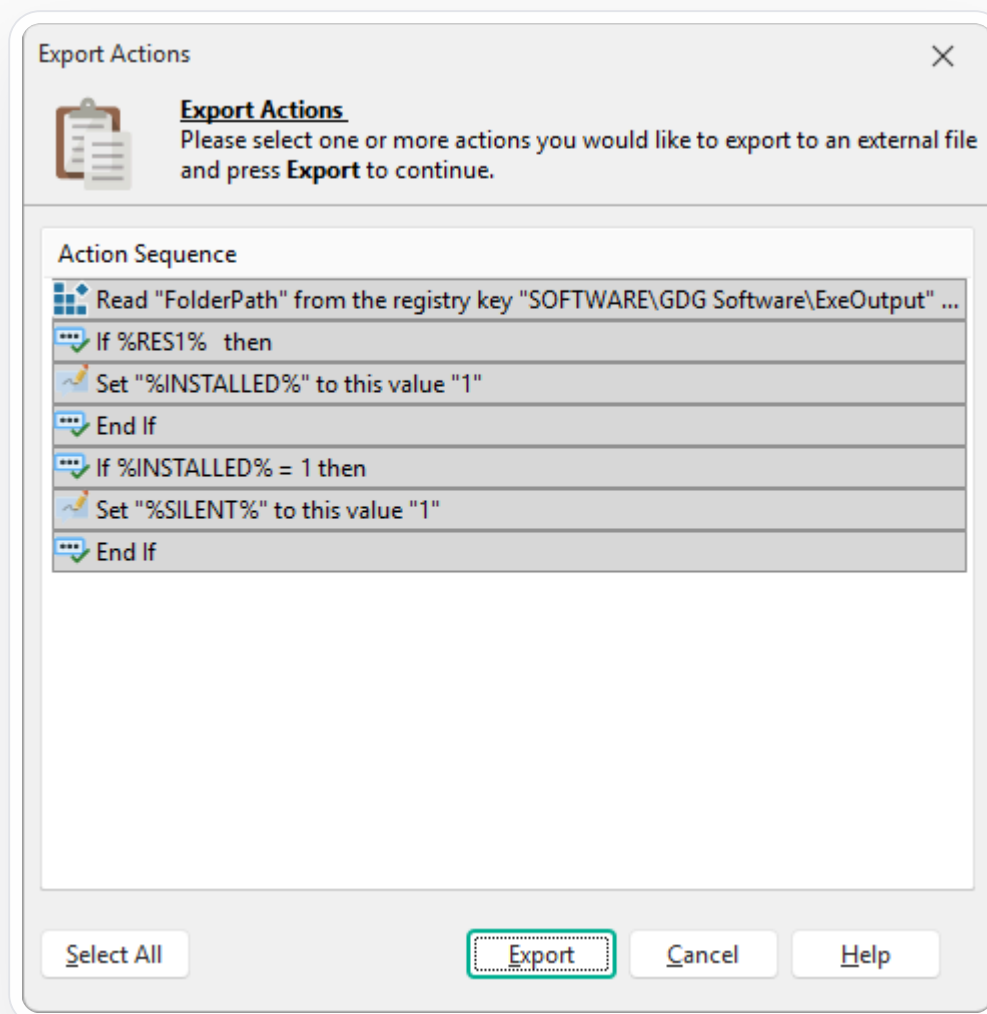
### Tip

Use the search box in the Add Action dialog to quickly find actions.

## 62.4 Import and Export

## Export

Highlight actions and click **Export Actions** to save them as an XML file for reuse across projects.



## Import

Click **Import Actions** and select an XML file. Choose which actions to import — they are added without overwriting existing ones.

## Copy / Paste blocks

Use **Action Tools > Copy Block** and **Paste Block** to duplicate multiple actions simultaneously.

## 62.5 Undo and Redo

The manager supports up to **30 levels** of undo/redo (Ctrl+Z / Ctrl+Y), including reordering, deletions, and changes made by the **AI Assistant**.

## 62.6 Keyboard Shortcuts

| Shortcut  | Action                 |
|-----------|------------------------|
| Ctrl+A    | Add a new action       |
| Ctrl+C    | Copy selected action   |
| Ctrl+Z    | Undo last change       |
| Ctrl+Y    | Redo                   |
| Del       | Remove selected action |
| Page Down | Jump to next event     |

## 62.7 Related Pages

### Custom Actions Overview →

Understand the event-based execution model.

### Action Properties Editor →

Configure parameters for each action.

### Action Catalog →

Full list of all built-in actions.

### Action Templates →

Reuse ready-made action sequences.

## 63. Editing Action Properties

The **Action Properties Editor** lets you configure how a specific custom action behaves by adjusting its parameters.

Paquet Builder - Action Properties
✕

**Ask user for one or more choices (boolean type)**  
 Use the editor below to modify the properties of the selected action. Press **Help** to display the help topic related to this action.

|                 |  |
|-----------------|--|
| Caption         | #WizInputTitle   |
| Choice1Title    | Subscribe to our newsletter  |
| Choice1Variable | %NEWS1%  |
| Choice1Visible  | <input checked="" type="checkbox"/>                                      |
| Choice2Title    | Add a shortcut to the desktop  |
| Choice2Variable | %VSHORTCUT%  |
| Choice2Visible  | <input checked="" type="checkbox"/>                                      |
| Choice3Title    |  |
| Choice3Variable |  |
| Choice3Visible  | <input type="checkbox"/>   |
| LabelText       | Choose the options you want:   |
| PromptText      | Please check the following choices: <span style="float: right;">⋮</span> |

Insert /S
OK
Cancel
Help

### 63.1 Editor Layout

The editor organizes properties into two columns:

| Column                | Description   |
|-----------------------|---|
| <b>Property Name</b>  | Some properties include sub-properties indicated by a + sign. Expand/collapse with the icons or keyboard.                               |
| <b>Property Value</b> | Depending on the type: text field, drop-down menu, or ellipsis button for complex input. Booleans display a <b>True/False</b> dropdown. |

## 63.2 Editing Values



### Inline editing

Modify values directly in the **Value** column. Press **Enter** to save, **OK** to apply all changes, or **Cancel** to discard.



### Expanded editor

For complex strings (file paths, long text), click the **Ellipsis (...)** button to open the expanded editor.

Paquet Builder - Action Properties

**Perform operation on a variable**  
Use the editor below to modify the properties of the selected action. Press **Help** to display the help topic related to this action.

|           |                 |
|-----------|-----------------|
| Operation | varAssign       |
| Separator |                 |
| Value     | 1               |
| Variable  | %INSTALLED% ... |

Insert /\$      OK      Cancel      Help

## 63.3 Inserting Dynamic Content

In the expanded editor, insert dynamic content using the toolbar:



### Variables

Insert system or project **variables** (e.g. %DESTPATH%, %PBLANGID%).



## Resource Strings

Use **localized text** for multilingual support.



## File References

Reference files that will be unpacked during installation.



## Line Breaks

Use `/s` to insert a carriage-return-line-feed (e.g. `Line1/$Line2`).



## Caution

Clicking **Cancel** reverts the action to its original state before the editor was opened. There is no undo for individual property edits once the entire dialog is cancelled.

## 63.4 Related Pages

### Custom Action Manager [→](#)

Organize and reorder your actions.

### Custom Actions Overview [→](#)

How actions and events work.

### Action Catalog [→](#)

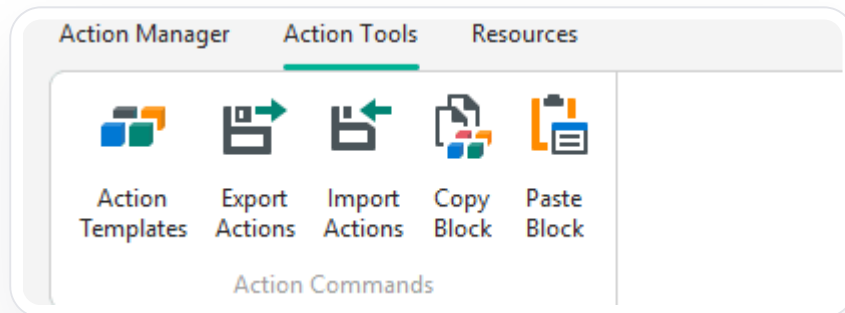
Full list of all available actions.

# 64. Custom Action Templates

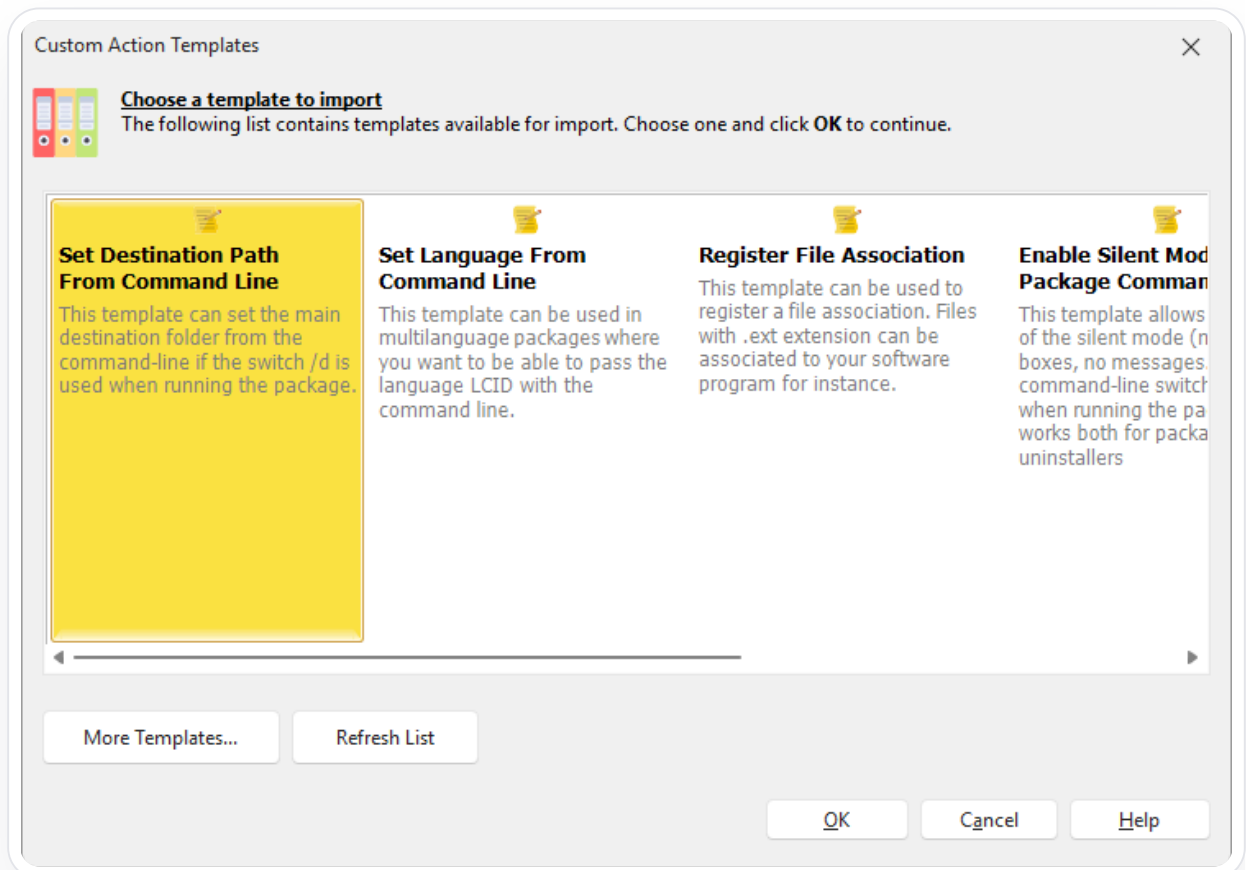
Templates let you **reuse predefined groups of actions** across projects. They simplify repetitive workflows by offering ready-made solutions you can import in one click.

## 64.1 Importing a Template

- 1 Open the **Custom Action Manager**.
- 2 Navigate to **Action Tools > Action Templates**.



- 3 Select a template from the list — each includes a title and description.



- 4 Click **OK** to import its actions into the selected event.

## 64.2 Built-in Templates



### Enable Silent Mode

Activates silent mode (no dialogs) using the `/s` command-line switch.



### Set Destination from CLI

Sets the **main destination folder** using the `/d` switch.



### Set Language from CLI

Configures the **language** using the LCID value.



### Register File Association

Associates a file extension with your software.



### Retrieve Windows Version

Gets the version and build number of Windows 10/11.

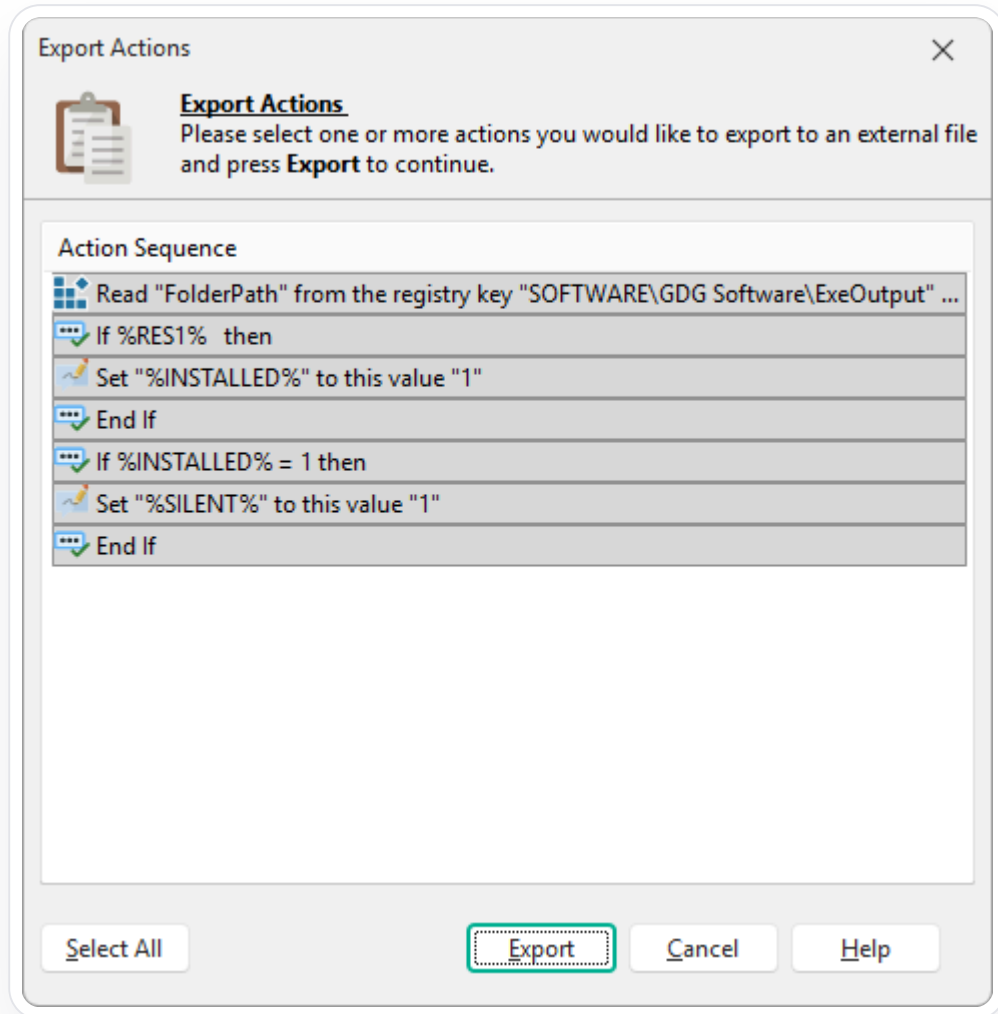


### Check Minimum Windows

Verifies the end user's system meets a required Windows version.

## 64.3 Creating Custom Templates

- 1 **Export actions** — in the Custom Action Editor, use **Action Tools > Export Actions** and save as XML.



- 2 **Add metadata** — open the XML file and add template nodes:

```
<pbgroup>
  <templatename>Register File Association</templatename>
  <templatedesc>Registers a file association for .ext files.
</templatedesc>
  ...
```

- 3 **Place the file** — move the XML to one of the template directories below, then click **Refresh List**.

## 64.4 Template Directories

| Location            | Path  |  |
|---------------------|---|--|
| <b>Application</b>  | Paquet Builder\Actions                        |  |
| <b>User profile</b> | %APPDATA%\GDG Software\Paquet Builder\Actions |  |

## 64.5 Sharing Templates

Share your templates with the community on the [GDG Software Forum](#).

## 64.6 Related Pages

### Custom Action Manager



Organize and manage actions.

### Custom Actions Overview



How the action system works.

### Action Catalog



Full list of all built-in actions.

## 65. Ask User (Text Input)

This action displays a dialog box or a **wizard step** that prompts the user to enter a text string, such as an email address, user name, password, or serial number.

### 65.1 Properties

| Property             | Data Type      | Description  |
|----------------------|----------------|--|
| <b>Caption</b>       | String         | The title of the dialog box or the wizard's <b>instruction panel</b> . |
| <b>DefaultAnswer</b> | String         | The initial value displayed in the text field.                         |
| <b>LabelText</b>     | String         | Text displayed immediately above or next to the input field.           |
| <b>PromptText</b>    | String         | The question or instruction text displayed to the user.                |
| <b>Variable</b>      | <b>VarName</b> | The variable that will store the user's input (e.g., %USERNAME%).      |

### 65.2 Behavior

- The package displays the **PromptText** and an input field for the user's response.
- When the user clicks **OK** or **Next**, the input string is stored in the specified **Variable**.
- If the user clicks **Cancel**, the package will exit by default.

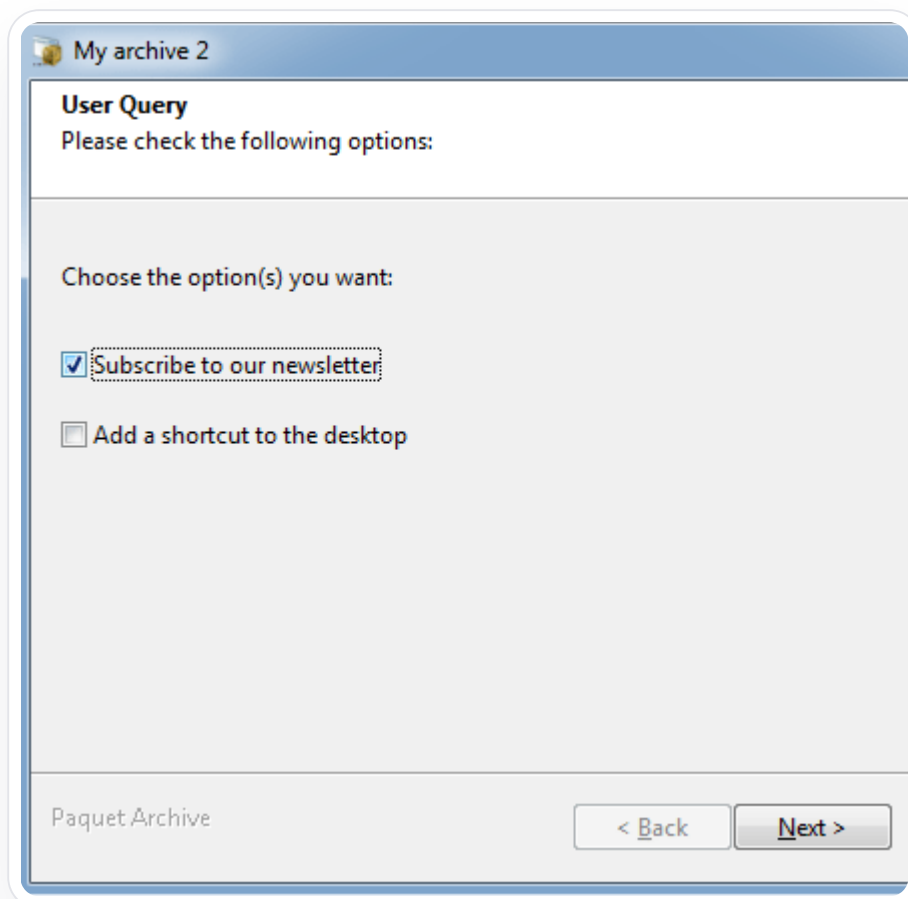
#### Tip

You can use a **goto label statement** to check whether end users have entered correct information: click [here](#) for an example.

See also: [All available actions](#)

## 66. Ask User for One or More Choices

This custom action displays a dialog box allowing end users to select one or more options (check boxes). It is useful for scenarios like deciding whether to install specific components ([see components](#)) or configuring conditional workflows using [if/then/else actions](#).



### 66.1 Properties

| Property Name       | Data Type | Description  |
|---------------------|-----------|--|
| <b>Caption</b>      | String    | The title displayed on the dialog box or in the wizard's <a href="#">instruction panel</a> . |
| <b>PromptText</b>   | String    | The question displayed as the dialog's title, asking users for input.                        |
| <b>LabelText</b>    | String    | Additional text displayed near the check boxes.  |
| <b>Choice1Title</b> | String    | The text displayed next to the first check box.  |

| Property Name                         | Data Type      | Description   |
|---------------------------------------|----------------|---|
| <b>Choice1Visible</b>                 | Boolean        | Determines whether the first check box is visible.                  |
| <b>Choice1Variable</b>                | <b>VarName</b> | Variable storing the result of choice 1, e.g., <i>%INSTALLIT%</i> . |
| <i>(Repeats for Choice 2 &amp; 3)</i> | ...            | Same properties are available for choices 2 and 3.                  |


## 66.2 Behavior

- The dialog displays all check boxes with the **Visible** property set to `True`.
- Check boxes are initially checked if their corresponding variables (e.g., *Choice1Variable*) are set to **1**; otherwise, they are unchecked.
- When users press **OK** or **Next**, the package updates each variable with the user's selection:
  - Checked: Variable value is set to **1**.
  - Unchecked: Variable value is set to **0**.
- Pressing **Cancel** closes the package.

### 66.2.1 Example

Properties for the screenshot below:

Paquet Builder - Action Properties ✕

 **Ask user for one or more choices (boolean type)**  
Use the editor below to modify the properties of the selected action. Press **Help** to display the help topic related to this action.

|                 |  |
|-----------------|--|
| Caption         | #WizInputTitle   |
| Choice1Title    | Subscribe to our newsletter  |
| Choice1Variable | %NEWS1%  |
| Choice1Visible  | <input checked="" type="checkbox"/>                                      |
| Choice2Title    | Add a shortcut to the desktop  |
| Choice2Variable | %VSHORTCUT%  |
| Choice2Visible  | <input checked="" type="checkbox"/>                                      |
| Choice3Title    |  |
| Choice3Variable |  |
| Choice3Visible  | <input type="checkbox"/>   |
| LabelText       | Choose the options you want:   |
| PromptText      | Please check the following choices: <span style="float: right;">⋮</span> |

## 66.3 Additional Notes



### Tip

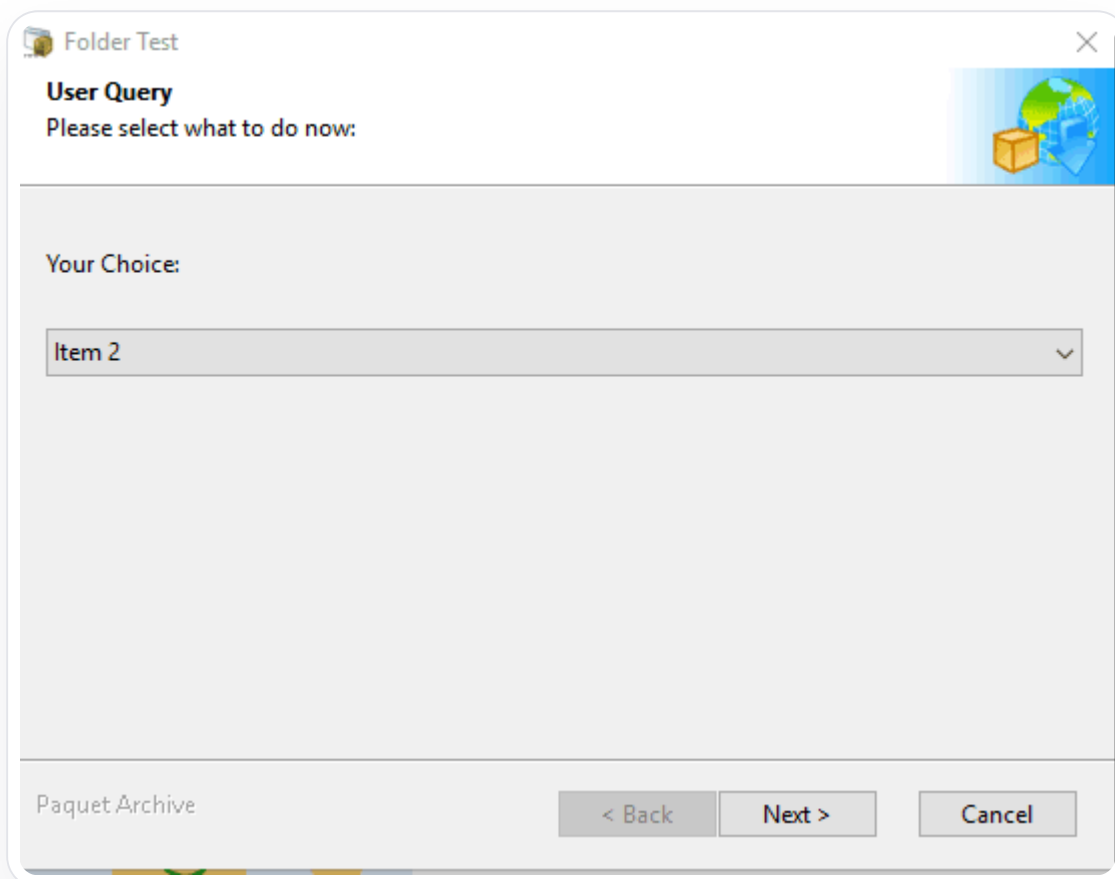
You can use a [goto label statement](#) to validate user input and branch workflows accordingly. See [this example](#).

## Action Catalog →

Browse all available custom actions.

## 67. Ask User for a Choice (Multiple Options in Drop Down)

Displays a dialog box that lets you ask end users to select one value from a list of pre-defined items (drop-down list). For instance, you could ask for a country or a language. You can use the result of the question for a [conditional if/then/else](#) action.



### 67.1 Properties

| Property Name     | Data Type | Description   |
|-------------------|-----------|---|
| <b>Caption</b>    | String    | The title that appears on the dialog box or in the wizard's <a href="#">instruction panel</a> . |
| <b>PromptText</b> | String    | The text that appears in the title of the question window and asks end users.                   |
| <b>LabelText</b>  | String    | This text will be displayed near the dropdown.  |

| Property Name        | Data Type      | Description   |
|----------------------|----------------|---|
| <b>ComboItems</b>    | String         | Defines the list of values to be displayed in the dropdown. <i>(See syntax rules below).</i>                        |
| <b>DefaultAnswer</b> | String         | Allows you to define which value in the list will be selected by default. If left empty, no value will be selected. |
| <b>Variable</b>      | <b>VarName</b> | The variable that will store the result of the choice, e.g. <i>%MYCHOICE%</i> .                                     |

## 67.2 Syntax Rules for ComboItems

Values are separated by spaces. If a value contains spaces, enclose it in double quotes. **Variables** and resource strings are allowed.

**Example:** "Item 1" "Item 2" #MyChoice1 will insert 3 items in the dropdown: Item 1, Item 2, and the text associated with the #MyChoice1 resource string.

### Action Catalog



Browse all available custom actions.

## 68. Ask User for a Folder

This action displays a directory selection dialog, allowing users to browse for or manually enter a folder path. It works exactly like the destination folder option in the [File Properties](#) page.

### 68.1 Properties

| Property                | Data Type      | Description   |
|-------------------------|----------------|---|
| <b>Caption</b>          | String         | The title of the dialog box or the wizard's <a href="#">instruction panel</a> .                                   |
| <b>DefaultDirectory</b> | String         | The initial folder path displayed when the dialog opens.  |
| <b>LabelText</b>        | String         | Text displayed near the input field. Default: <code>#AskBrowseFolders</code> ( <a href="#">Resource String</a> ). |
| <b>PromptDirText</b>    | String         | The instruction text shown within the folder browser window.  |
| <b>Variable</b>         | <b>VarName</b> | The variable that will store the selected path (e.g., <code>%MYPATH%</code> ).                                    |

### 68.2 Behavior

- The package prompts the user with the specified text and provides a “Browse” button.
- When the user clicks **OK** or **Next**, the chosen path is stored in the specified **Variable**.
- If the user clicks **Cancel**, the package will exit by default.

#### Tip

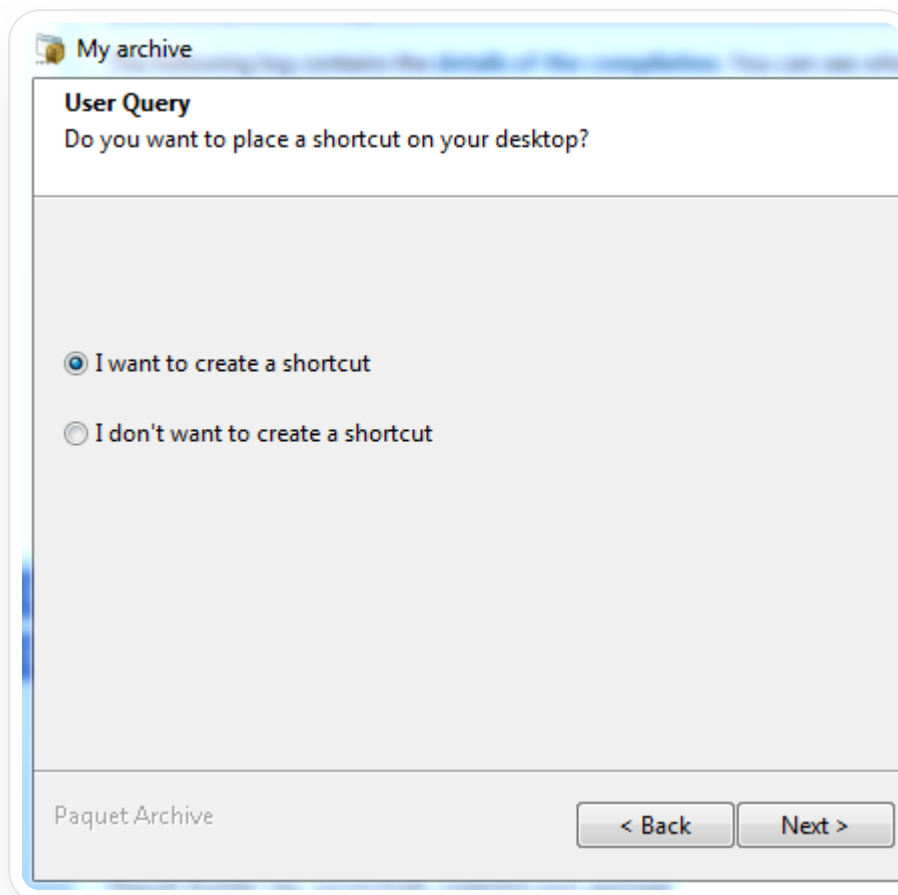
You can use a [goto label statement](#) to check whether end users have entered correct information: click [here](#) for an example.

#### Action Catalog →

Browse all available custom actions.

## 69. Ask User for a Choice

Displays a dialog box that lets you ask end users to select between two options (radio boxes). For instance, if they want to install a shortcut on the desktop. You can also use the result of the question for a **conditional if/then/else** action.



### 69.1 Properties

| Property Name     | Data Type | Description  |
|-------------------|-----------|--|
| <b>Caption</b>    | String    | The title that appears on the dialog box or in the wizard's <b>instruction panel</b> . |
| <b>PromptText</b> | String    | The text that appears in the title of the question window and asks end users.          |
| <b>LabelText</b>  | String    | This text will be displayed near the radio boxes.                                      |
| <b>YesText</b>    | String    | The text that appears on the first radio box (for instance, positive answer).          |

| Property Name         | Data Type      | Description  |
|-----------------------|----------------|--|
| <b>NoText</b>         | String         | The text that appears on the second radio box (for instance, negative answer).   |
| <b>ChoiceVariable</b> | <b>VarName</b> | The variable that will store the result of the choice, e.g. <i>%INSTALLIT%</i> . |

## 69.2 Behavior

The package shows a dialog box displaying the value of "PromptText" (the question) with two radio boxes.

Initial states of the two radio boxes are set according to the value of ChoiceVariable. If the variable mentioned in ChoiceVariable is set to "1", the first radio box will be checked. Otherwise, it is unchecked and the second one will be checked.

If the end user selects the Cancel button, the package closes.

### Note

If you want the first radio box to be checked by default, use a [Perform Operation on a Variable](#) custom action before this one to set your variable to 1.

You can use a [goto label statement](#) to check whether end users have entered correct information: click [here](#) for an example.

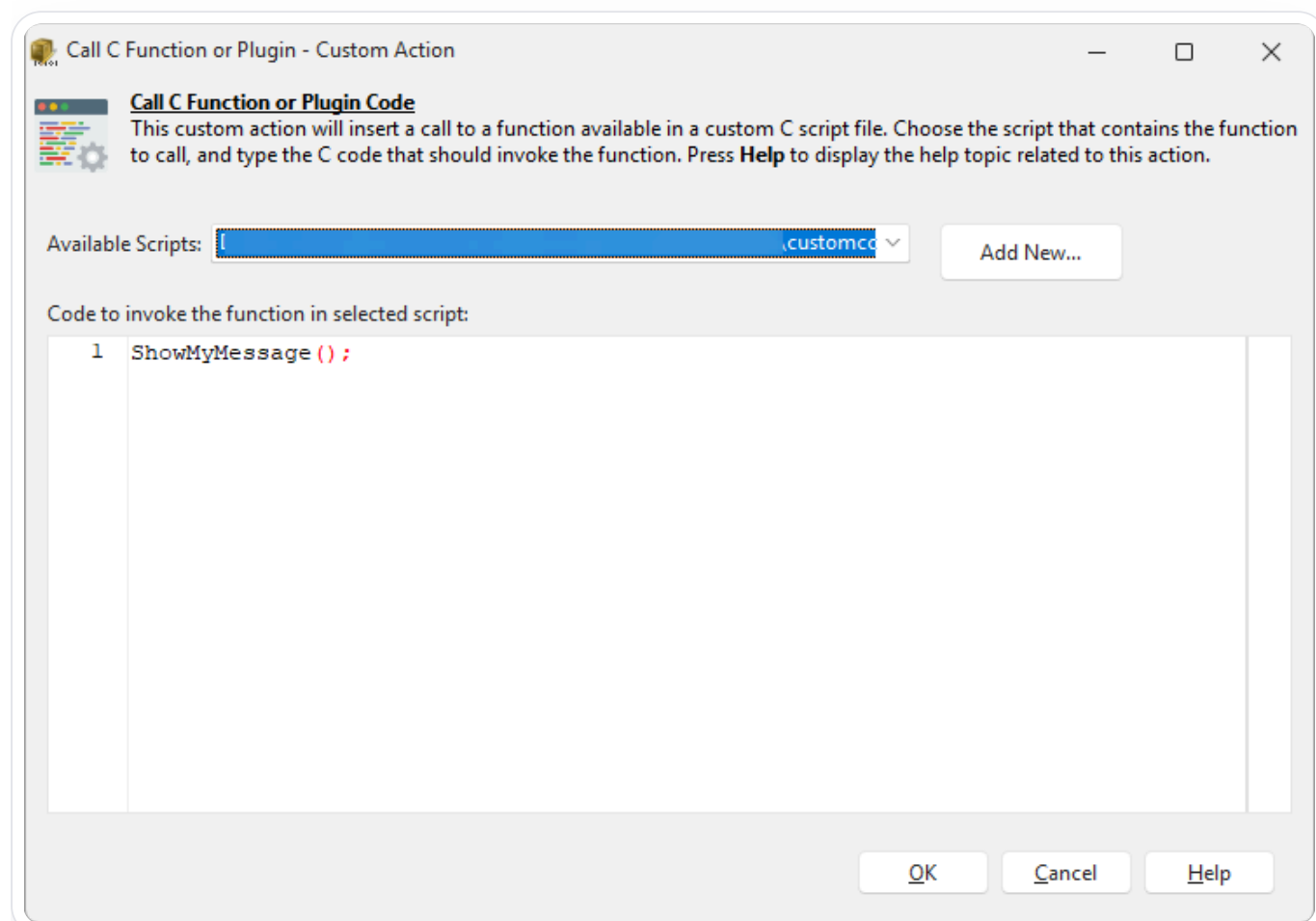
### Action Catalog

Browse all available custom actions.

## 70. Call C Function

This action lets you invoke a function available in one of the [custom C source code files](#) you added to the project.

When you edit the properties of this custom action, the following dialog box is displayed:



The “Available Scripts” list lets you choose the C source code file that contains the function you want to invoke. If it is not available in the list, you can select an existing C source file by clicking **Add New**.

In the text field, you can type the C code that will invoke the function. For instance, if the function has no parameter, it should be:

```
myfunction();
```

If it has two parameters, enter:

```
myfunction2(0, 2);
```

Paquet Builder will insert this code in the main script of the package/uninstaller. To avoid possible compilation problems since you don't have access to the main script, we recommend you enter minimal C code to be inserted: only function calls should be used. Your C code should remain in other C source files.

### Caution

A missing semicolon (;) or undeclared variable will cause compilation errors. Keep the code block limited to calling functions, and place all logic inside the `.c` file itself.

Please refer to the [Developer Options page](#) for further information and hints about this custom action.

## Action Catalog

Browse all available custom actions.

# 71. Check if a File/Folder Exists

This action will check whether the specified file or folder exists and then store the result in the specified variable.

## 71.1 Properties

| Property Name         | Data Type | Description   |
|-----------------------|-----------|---|
| <b>CheckFolder</b>    | Boolean   | If you want the custom action to verify the existence a folder, then set this property to True. Otherwise leave it to False.  |
| <b>Filename</b>       | String    | This is the full path to the file the function has to check the existence if CheckFolder is set to false. For example: %SYS%\GTtools.d11 This is the full path to the folder the function has to check the existence if CheckFolder is set to true. For example: %WIN%\ |
| <b>ReturnVariable</b> | VarName   | This is the variable that will contain the result. See below for results.   |

### Note

**About ReturnVariable values** (Boolean type): If the file (or folder) is found, then the variable specified in "ReturnVariable" will receive the value **1**. Otherwise it will be set to **0**. You can then evaluate the result using a [Conditional If/Then block](#) for instance.

### Action Catalog

Browse all available custom actions.

## 72. Check if a Process Is Running

This action will check whether the specified process is running and then store the result in the specified variable.

### 72.1 Properties

| Property Name  | Data Type | Description   |
|----------------|-----------|---|
| EXEFilename    | String    | This is the name of the process. For example: <code>PBuilder.exe</code> Do not specify a full path. |
| ReturnVariable | VarName   | This is the variable that will contain the result. See below for results.                           |

#### Note

**About ReturnVariable values** (Boolean type): If the process is found running, then the variable specified in "ReturnVariable" will receive the value **1**. Otherwise it will be set to **0**. You can then evaluate the result using a [Conditional If/Then block](#) for instance.

#### Action Catalog

Browse all available custom actions.

## 73. Check if a Registry Key Exists

This action checks whether a registry key exists in the Windows Registry. You can use it to check whether an application is installed, for example.

### 73.1 Properties

| Property Name         | Data Type | Description   |
|-----------------------|-----------|---|
| <b>Key</b>            | String    | The key name in the registry to be checked. For example: SOFTWARE\My App  |
| <b>ReturnVariable</b> | VarName   | The name of the variable whose value will be set to the result. If the registry key exists, it is set to <b>1</b> . Otherwise it is set to <b>0</b> (Boolean type). |
| <b>RootKey</b>        | hkKeys    | The base key of your registration key. There is a defined set of Base Keys from which you can choose in the list box.   |

#### Note

This action only checks for registry **keys**, not named entries. If you need to check whether a specific entry exists, use a [Read from registry](#) action and enter a default value (for example "0" or "-1" — something the registry entry should never be set to). If the entry is not found, the action will return the default value. You can then use a [Conditional If/Then](#) block to determine whether the entry exists.

#### Action Catalog

Browse all available custom actions.

## 74. Conditional If/Then/Else

The Conditional If/Then/Else action allows you to create decision-making logic within your package. It lets you execute a set of actions conditionally based on specific criteria. If the condition in the **Begin Conditional If/Then** action is true, the actions within the block execute; otherwise, they are skipped.

For more details, see this [Conditional Wikipedia topic](#).

### Caution

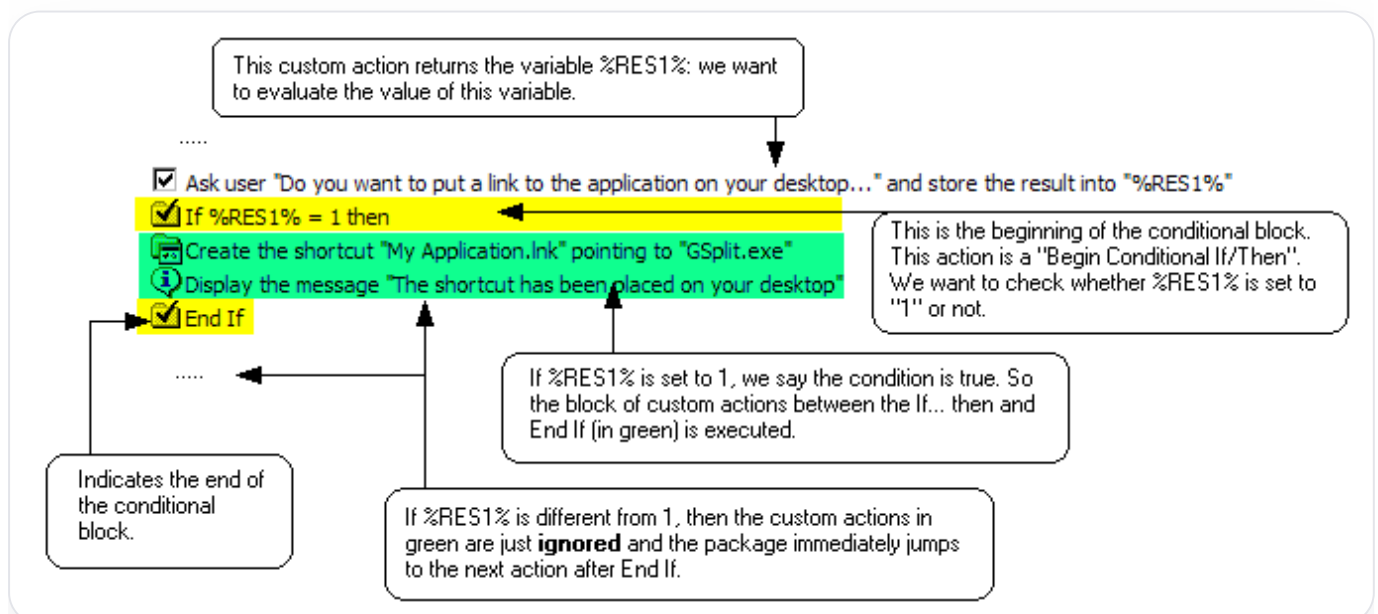
Every If block must end with an "End Conditional If/Then" action. Optionally, an "Else Conditional If/Then" action can be included for alternate execution paths.

### 74.1 How to Use

- 1 Start an If block by adding a **"Begin Conditional If/Then"** action.
- 2 Add all actions to execute if the condition is true.
- 3 Close the If block with an **"End Conditional If/Then"** action.
- 4 Optionally, insert an **"Else Conditional If/Then"** action between the Begin and End for an alternate path.

#### 74.1.1 Example

This setup evaluates the value of a variable to decide the execution path:



## 74.2 Properties of "Begin Conditional If/Then"

This action starts an If block.

| Property Name        | Data Type | Description  |
|----------------------|-----------|--|
| Operator             | TOperator | Determines how <b>Variable1</b> is compared to <b>Variable2</b> (see operators below). |
| Variable1            | VarName   | Left operand: the variable or value to be compared.                                    |
| Variable2<br>(Value) | String    | Right operand: the value or variable to compare against <b>Variable1</b> .             |

### 74.2.2 Operators Available

- **Equals:** Checks if **Variable1** equals **Variable2**.
- **NotEqual:** Checks if **Variable1** does not equal **Variable2**.
- **GreaterThan:** Checks if **Variable1** is greater than **Variable2**.
- **LessThan:** Checks if **Variable1** is less than **Variable2**.

## 74.3 Properties of "End Conditional If/Then"

This action closes an If block. It has no properties.

## 74.4 Properties of "Else Conditional If/Then"

- If the condition is false, actions within the **Else** branch execute.
- It must be placed between **Begin Conditional If/Then** and **End Conditional If/Then** actions.
- It has no properties.

## 74.5 Error Detection for Missing "End If/Then"

When you compile your project, Paquet Builder automatically checks for missing **End Conditional If/Then** actions to ensure your logic is complete.

## Action Catalog



Browse all available custom actions.

## 75. Copy a File/Folder

This action offers the possibility to copy a single file or an entire group of files (using a wildcard) to another destination folder.

To copy an entire folder, just specify the folder name followed by `\*.*`, like `%SRC%\*.*` or `c:\my documents\*.*.pas`.

### 75.1 Properties

| Property Name            | Data Type | Description   |
|--------------------------|-----------|---|
| <b>Destination</b>       | String    | This property indicates the full path + filename of the destination file. For example: <code>%DESTPATH%\Hello.exe</code> If you use wildcards in the Source property, this property indicates the folder where all source files should be copied. For example <code>%DESTPATH%\My Folder\</code> ( <b>warning: it should always end by a directory slash \</b> ). |
| <b>MoveFiles</b>         | Boolean   | Indicates that the source files should be deleted once they have been copied.   |
| <b>OverwriteIfExists</b> | Boolean   | If the destination file exists, indicates whether the package should overwrite the existing file or skip it.  |
| <b>ShowProgress</b>      | Boolean   | Indicates whether a progress monitor is displayed or not (Windows shell dialog box).  |
| <b>Source</b>            | String    | Path to the source file(s) to be copied to the Destination file. It can be a path to a single source file, or a path to a folder followed by a wildcard. For instance, <code>%SYS%\My shared\*.*.dll</code>   |

#### Caution

This function does not add any reference of the new file in the uninstaller log file. You should add it manually [using this action](#).

#### Note

The destination path must exist before copying your file! The function does not support creating folders. Use a [Create a folder](#) custom action.

## Action Catalog



Browse all available custom actions.

## 76. Create a Folder

This action lets you create a folder.

### 76.1 Properties

| Property Name       | Data Type | Description   |
|---------------------|-----------|---|
| <b>AddUninstRef</b> | Boolean   | Indicates whether the package should reference the newly-created folder to the uninstaller .log file. |
| <b>FolderName</b>   | String    | The folder to create; for example <code>%DESKTOPDIR%\My group</code>                                  |

#### Tip

This function is useful for creating folders in the Start menu before [adding new shortcuts](#), [copying files](#) or when you [work with file properties](#).

#### Action Catalog →

Browse all available custom actions.

## 77. Create an Internet Shortcut

This action will create an Internet shortcut. These shortcuts point to a URL: when such a shortcut is executed, the default Web browser is opened to the URL specified by the latter.

### 77.1 Properties

| Property Name           | Data Type | Description  |
|-------------------------|-----------|--|
| <b>DestinationURL</b>   | String    | The URL the shortcut should point to. The Web browser will be opened to this URL.  |
| <b>ShortcutFilename</b> | String    | Defines the path+filename of your shortcut (the text which will appear near the icon on the desktop for example) and where it will be located. Be sure to give the .url extension. See below for examples. |

#### 77.1.1 Example for ShortcutFilename

To create an Internet shortcut on the desktop, for instance you can enter: `%DESKTOPDIR%|My homepage.url`. This will create an icon with the text "My Homepage" on the Windows desktop.

#### Note

Internet shortcuts are removed when the **package is uninstalled**.

#### Action Catalog

Browse all available custom actions.

## 78. Create a Shortcut

This action creates a new Windows shortcut (or link), as the ones you find on your desktop. If you would like Paquet Builder to make your shortcuts automatically, you can use the [Shortcut Creation](#) page, although this action offers more options.

### 78.1 Properties

| Property Name       | Data Type     | Description   |
|---------------------|---------------|---|
| <b>Description</b>  | String        | A description for your new shortcut. It will appear in the Tip window when you move the mouse's cursor on the shortcut's icon. Optional.  |
| <b>IconFilename</b> | String        | The path to the file which should contain the icon resource for your shortcut. Optional.  |
| <b>IconIndex</b>    | Integer       | If the icon resource file specified by <i>IconFilename</i> contains several icons, you can indicate which icon should be used (0, 1, 2,..., n-1). Optional (0 by default).  |
| <b>LinkFilename</b> | String        | Defines the name of your shortcut (the text which will appear near the icon) and where it will be located. Be sure to give the <b>.lnk</b> extension. See below for examples.   |
| <b>LinkTarget</b>   | String        | This is the file that will be executed by Windows when you double-click the shortcut (document or executable file). Example:<br><i>%DESTPATH% MyDoc.hlp</i> You can select a reference to a file in the package by pressing the "..." button. |
| <b>Parameters</b>   | String        | Any command line options that should be passed to the target program file. Optional.  |
| <b>ShowWindow</b>   | dwWindowState | Defines the state of the target program's window. Optional.   |
| <b>WorkingDir</b>   | String        | Sets the default working directory of the shortcut. When the shortcut's associated target file is run, Windows will set the current directory to this folder. Optional.   |

#### 78.1.1 Examples for LinkFilename

- To create a shortcut on the desktop, enter: `%DESKTOPDIR%\My Document.Ink`. This will create an icon with the text "My Document" on the Windows desktop.
- To create a shortcut in a new program menu group, enter this: `%PROGMENUDIR%\Hello World\My Document.Ink`. This will create a new icon named My Document in the folder Hello World.

### Tip

If your end users are [prompted for selecting a Start Menu folder for your application](#), then you can also use the `%SHORTCUTPATH%` variable to place your shortcuts in it. Example: enter `%SHORTCUTPATH%\My Document.Ink` in the LinkFilename field.

### Caution

This action does **not** support creating folders! Ensure that the destination folder for your shortcuts was created before placing them, otherwise your shortcuts will not be created at all. You can use a [Create a folder](#) custom action.

### Note

Shortcuts made with this custom action are **NOT** automatically removed when the [package is uninstalled](#). You have to use a [Remove single file](#) custom action for the uninstaller.

## Action Catalog

Browse all available custom actions.

## 79. Define a Label

This custom action inserts a script label which is used in conjunction with the [Go To A Label](#) custom action. The **Go To A Label** action transfers the package execution to the statement marked by the specified “Define a label” custom action. To mark a statement, you must **first declare the label** using this custom action.

### 79.1 Properties

| Property Name    | Data Type           | Description   |
|------------------|---------------------|---|
| <b>LabelName</b> | TLabelName (String) | Identifies the label to insert. A label name can be any valid identifier or any numeral between 0 and 9999, like LABEL1, RETURN25, etc. |

#### Note

All labels must be declared with a “Define a label” custom action before they can be used with a Go To Label custom action.

To define a new label, just enter its name in the Label manager and press OK.

#### Caution

Label names are **unique**: you cannot have the same label name twice.

Remember that labels are defined by the “Define a label” custom action you add. If you remove this custom action later, you will also remove the label: in this case, ensure that no “Go to a label” custom action that could point to the old label remains. Otherwise, the C code compiler will raise an exception.

### 79.2 See Also

#### **Go To A Label** →

Transfer package execution to a defined label.

#### **All Available Actions** →

Browse the full catalog of custom actions.

## 80. Delete a Folder

This action lets you remove an existing folder, optionally including all of its sub-directories and files.

### Caution

This action cannot be undone. Make sure you target the correct folder.

### 80.1 Properties

| Property          | Data Type | Description  |
|-------------------|-----------|--|
| <b>DelTree</b>    | Boolean   | Determines whether the package should also first remove all sub-directories and files inside them. If you choose false, the folder must be empty in order to be removed. |
| <b>FolderName</b> | String    | Full path to the folder to be removed. Variables can be used of course.  |

### Action Catalog [→](#)

Browse all available custom actions.

# 81. Do an Arithmetic Operation

This action takes two variables (that should contain integer values only) and performs an arithmetic operation (such as addition, subtraction...) with their values.

## 81.1 Properties

| Property             | Data Type       | Description   |
|----------------------|-----------------|---|
| <b>LeftVariable</b>  | VarName         | Indicates the name of the left variable: its value will be read and “put before” the operator. See below. May be directly an integer and not a variable name. |
| <b>Operation</b>     | TArithmOperator | Tells the package which arithmetic operation it should execute. A list of possible operations to perform is available by pressing the Down button. See below. |
| <b>ResultVar</b>     | VarName         | The variable that will store the result of the operation. For example %MYRESULT%.   |
| <b>RightVariable</b> | VarName         | Indicates the name of the right variable: its value will be read and “put after” the operator. See below. May be directly an integer and not a variable name. |

## 81.2 Left/Right Variables

The package will evaluate the operation like this: `Left Variable Operator Right Variable`

For example, consider the left variable %LEFT% set to “6”, the right variable %RIGHT% set to “4” and the operation is a multiplication: the resulting expression will be  $6 \times 4$ . The package will then store “24” in the ResultVar.

## 81.3 Available operations

To simplify we will use %LEFT% and %RIGHT% to represent the values of the variables specified by LeftVariable and RightVariable.

| Name                           | Description  | Required  | Example  |
|--------------------------------|--|---|--|
| cDoAbsolute                    | Returns an absolute value  | Only LeftVariable is used.                                    | If %LEFT% is set to "-4", the result will be "4".  |
| cDoAddition                    | Standard addition  | Two variables required.                                       | If %LEFT% is set to "8" and %RIGHT% to "4" the result will be $8+4 = "12"$ .             |
| cDoIntDivision (div in Pascal) | Integer division: this is the value of %LEFT% / %RIGHT% rounded in the direction of zero to the nearest integer.           | Two variables required. RightVariable should not be set to 0! | If %LEFT% is set to "8" and %RIGHT% to "4" the result will be $8/4 = "2"$ .              |
| cDoMultiplication              | Standard multiplication  | Two variables required.                                       | If %LEFT% is set to "8" and %RIGHT% to "4" the result will be $8 \times 4 = "32"$ .      |
| cDoRemainder (mod in Pascal)   | Returns the remainder obtained by dividing its operands. In other words, $x \text{ mod } y = x - (x \text{ div } y) * y$ . | Two variables required.                                       | If %LEFT% is set to "8" and %RIGHT% to "3" the result will be $8 \text{ mod } 3 = "2"$ . |
| cDoSubstraction                | Standard subtraction   | Two variables required.                                       | If %LEFT% is set to "2" and %RIGHT% to "3" the result will be $2-3 = "-1"$ .             |

You may also directly enter an integer as the value for LeftVariable and RightVariable without using a real variable. For example, you can directly enter "3" for RightVariable.

### Note

Operations are done using 64-bit integers (numbers in  $[-2^{63}; 2^{63}[$ ). For instance you could use this custom action to translate the value returned by the [Get Free Disk Space](#) custom action from bytes to MB or GB.

## Action Catalog

Browse all available custom actions.

## 82. Execute a BAT File

This action lets you silently execute a batch (.bat) file. This command is interesting if you would like to register an ActiveX control using REGSVR32.EXE for example. If you need to run external programs, consider the "Execute a program" action.

### 82.1 Properties

| Property            | Data Type | Description   |
|---------------------|-----------|---|
| <b>BatFilename</b>  | String    | Full path to your .BAT file. Example: <i>%DESTPATH%\Temp\RegOCX.Bat</i>                         |
| <b>HideProgram</b>  | Boolean   | Indicates whether the command line window is hidden or not.                                     |
| <b>WaitUntilEnd</b> | Boolean   | Defines whether the package has to wait until the end of execution or continue without waiting. |
| <b>WorkingDir</b>   | String    | Full path to the working directory.   |

#### Action Catalog



Browse all available custom actions.

## 83. Execute a Dialog Scenario

### Caution

This action is provided for internal use only. **You should not use it** because it can perturb your system if not correctly configured.

The package will display the specified custom dialog and execute all internal actions associated to the dialog's controls.

### 83.1 Properties

| Property            | Data Type | Description   |
|---------------------|-----------|---|
| <b>DialogID</b>     | String    | A unique identifier used by Paquet Builder to compile the dialog resource into the package.   |
| <b>IgnoreSilent</b> | Boolean   | If true, the dialog is displayed even if the <b>silent mode is active</b> .   |
| <b>Scenario</b>     | Filename  | Defines the real path (no variable accepted) to the source dialog scenario filename. The file must be in Paquet Builder's dialog scenario format. |

### Action Catalog

Browse all available custom actions.

## 84. Execute a Program

This action lets you execute a program file (executable file .EXE). This function is useful if you want to launch additional applications or run several Setup programs.

### 84.1 Properties

| Property                   | Data Type | Description  |
|----------------------------|-----------|--|
| <b>Filename</b>            | String    | Full path to the file you would like to execute. Example: %DESPATH%\Setup.exe  |
| <b>HideProgram</b>         | Boolean   | If True, the program is run hidden: its window is not displayed.   |
| <b>Parameters</b>          | String    | If you would like to pass any command line options to your program. Optional.  |
| <b>VariableForExitCode</b> | VarName   | Variable that will receive the exit code returned by the process. If you know the possible exit codes for the program you run, you can perform some additional actions according to the variable's value. Note that this variable is not initialized before the "Execute a program" is executed. It's your job to initialize it if you use it. |
| <b>WaitUntilEnd</b>        | Boolean   | Defines whether the package has to wait until the end of execution or continue without waiting.  |
| <b>WorkingDir</b>          | String    | Path to the working directory (can be a variable)  |

#### Caution

If your package runs with **administrative rights (defined by the manifest)**, any executable started by the package will also be run with administrative rights. To avoid this, you can use the **Open a file custom action** to run the executable as un-elevated process.

#### Note

This feature is also available in the **Program/Setup Execution** panel.

#### Action Catalog

Browse all available custom actions.

## 85. Exit Package

---

This action will immediately terminate the package's execution. It is useful inside Conditional If/Then statements, if an unexpected condition occurs.

No parameters.

### Action Catalog



Browse all available custom actions.

## 86. Find One String Within Another

This action will find one string within the given text. It stores the position of the first occurrence in the specified variable or 0 if there is no occurrence.

Useful for [command-line switches](#) for instance.

### 86.1 Properties

| Property             | Data Type               | Description   |
|----------------------|-------------------------|---|
| <b>CaseSensitive</b> | Boolean                 | If true, it will be a case sensitive search.                                  |
| <b>ResultVar</b>     | <a href="#">VarName</a> | This is the variable that will contain the result. See below for results.     |
| <b>StartPosition</b> | Integer                 | The start position to begin the search. The first valid character index is 1. |
| <b>StringToFind</b>  | String                  | The string to find. Variables are accepted.                                   |
| <b>Text</b>          | String                  | The string to look in. Variables are accepted.                                |

#### Tip

**About ResultVar value:** The variable will be set to the position (in character) of the string to find, or 0 if the string is not found.

For a sample, see [Action Templates](#).

#### Action Catalog [→](#)

Browse all available custom actions.

## 87. Get File Information

This action retrieves a specific property you want to know about a given file and stores it into a variable.

### 87.1 Properties

| Property              | Data Type   | Description  |
|-----------------------|-------------|--|
| <b>Filename</b>       | String      | Indicates the full path to the file you want to get information about. For instance <code>%SYS%\shell32.dll</code> |
| <b>InfoToGet</b>      | TfiFileInfo | Tells the package which information to retrieve. See below.  |
| <b>ReturnVariable</b> | VarName     | The variable that will store the result of the operation. For example <code>%MYRESULT%</code> .                    |

### 87.2 Available information

| Name                    | Description   |
|-------------------------|---|
| fiGetFileCRC32          | Computes the 32-bit cyclic redundancy checksum (CRC-32) of the file. This command is useful if you want to check whether a file is not altered. May require time for large files (several GB) depending on the computer configuration. Use the "Tools   Compute CRC-32 of a file" menu command in Paquet Builder to get the CRC-32 of an original file, then use this action to calculate the one of the same file (once extracted). Using a <a href="#">Conditional If block</a> , you can check whether the file was successfully extracted or not. |
| fiGetFileMajorVersion   | Returns the Major Version number of an executable file (from Version Info resource block)   |
| fiGetFileMinorVersion   | Returns the Minor Version number of an executable file (from Version Info resource block)   |
| fiGetFileRevisionNumber | Returns the File Revision number of an executable file (from Version Info resource block)   |
| fiGetFileBuildNumber    | Returns the File Build number of an executable file (from Version Info resource block)  |

| Name          | Description  |
|---------------|--|
| fiGetFileSize | Obtains the size in bytes of the specified file.   |
| fiTestEXEx86  | Returns whether the tested file is a 32-bit executable (value set to 1 = true). If the EXE file is 64-bit or ARM64, value is set to 0. If the tested file is not a valid EXE, the value is set to blank. |

### Tip

The first 4 options are used to compare file version information: you can determine whether a file already present on the destination computer is more recent or not (and consequently whether it should be overwritten or not). This has to be used with the **conditional variables** and a **Conditional If block** for instance.

### Tip

“fiTestEXEx86” lets you test whether an executable program file is 32-bit or not (64-bit and ARM64 executables both return 0). It works on any Windows version. Use the %OSARCH% variable to detect the system architecture.

## Action Catalog →

Browse all available custom actions.

## 88. Get Full Path of a Windows Known Folder

This action retrieves the full path of a known folder identified by the folder's KNOWNFOLDERID and stores it into a variable. See [MSDN article about Known Folders](#)

### 88.1 Properties

| Property              | Data Type      | Description   |
|-----------------------|----------------|---|
| <b>KnownFolderID</b>  | String         | Indicates the KNOWNFOLDERID constant that identifies the Known Folder. See below. |
| <b>ReturnVariable</b> | <b>VarName</b> | The variable that will store the full path. For example %MYRESULT%.               |

#### Caution

This custom action will return a blank value if an error occurs or if the package is run on Windows XP. In fact, Known Folders is a feature only available in Windows Vista and later.

### 88.2 KNOWNFOLDERID constants

See the corresponding [MSDN article for all KNOWNFOLDERID constants](#)

Some examples:

- `FOLDERID_LocalAppDataLow` returns `%USERPROFILE%\AppData\LocalLow`.
- `FOLDERID_InternetCache` returns `%LOCALAPPDATA%\Microsoft\Windows\Temporary Internet Files`.

All paths are returned without a trailing backslash. For example, `"C:\Users"` is returned rather than `"C:\Users\"`.

#### Caution

Use the plain KNOWNFOLDERID constant name `FOLDERID_XXXX`, not the GUID. This name is case-sensitive: a typo error will raise a syntax error of the C compiler.

#### Note

These constants are defined in the `knownfolders.h` header file available at `Compiler\Include\Win`.

## Action Catalog



Browse all available custom actions.

## 89. Go to a Label

This custom action transfers the package execution to the statement marked by the specified label. To mark a statement, you must first declare the label using the “Define a label” custom action. See an example below.

### 89.1 Properties

| Property         | Data Type | Description   |
|------------------|-----------|---|
| <b>LabelName</b> | String    | Identifies the label the script will jump to. A label name can be any valid identifier or any numeral between 0 and 9999, like LABEL1, RETURN25, etc. |

All labels must be declared with a “Define a label” custom action before they can be used with a Go To Label action.

### 89.2 Example of using Go to label

On this screenshot you can see an example of the use of “Define a label” / “Go to a label” custom actions. We ask end users for their name and create a loop: they will be prompted to enter their name until they provide a non-blank value.

The screenshot displays the 'Package Main Events' sidebar on the left and the 'Action Sequence' editor on the right. The 'Action Sequence' contains the following steps:

- `-- Label LABEL1--` (Annotated: 1) We defined a label called LABEL1 using the « Define a label » custom action.
- `Ask user "Please enter your name" and store the result into "%GET1%"` (Annotated: 2) We ask the user for his/her name and check whether he left the field blank.
- `If %GET1% = "" then` (Annotated: 3) If no answer was provided, we show a warning message...
- `Display the message "You must enter your name, please!"` (Annotated: 3) If no answer was provided, we show a warning message...
- `Go to the label "LABEL1"` (Annotated: ...and we return to the LABEL1 thanks to the « Go to a label » custom action. So the user will have the possibility to enter his/her name again.
- `End If` (Annotated: ...and we return to the LABEL1 thanks to the « Go to a label » custom action. So the user will have the possibility to enter his/her name again.)
- `Display the message "Thank you, %GET1%, for installing my package."` (Annotated: 4) If he/she provided his/her name, then we show « Thank you » and can continue...
- `Show wait message "Please wait while unpacking files..."` (Annotated: 4) If he/she provided his/her name, then we show « Thank you » and can continue...

This can be used to check values provided by users (folders, etc.), to jump to specific blocks of instructions when a condition is not met, etc.

### 89.3 Restrictions

## Caution

- **“Go to a label” actions should be used sparingly** — only for simple cases like the example above.
- **Do not jump across events.** Although you are allowed to do this, you should not jump to a label that is not in the same event as the “Go to” action. Paquet Builder automatically adds internal actions between events: incorrect jumps can have unpredictable effects.
- **Compiler errors for undefined labels.** If a Go To Label points to a label name that does not exist, the C code compiler will raise an error when compiling the script generated from custom actions.

## Action Catalog

Browse all available custom actions.

## 90. Insert an Uninstall Command

This action should be used only if you decide to include an **uninstaller** with the package. It allows you to add a custom action to the uninstaller log file; this command will be then executed during the uninstall process. This feature is interesting to remove additional folders like program groups and other items that are not automatically referenced. Alternatively, you can use custom actions associated to events of the uninstaller to perform additional cleaning tasks.

### 90.1 Properties

| Property          | Data Type | Description   |
|-------------------|-----------|---|
| <b>Command</b>    | ucComs    | The uninstall command that should be performed. There is a defined set of commands from which you can choose in the list box. Please see below for information. |
| <b>Parameters</b> | String    | Depending on the selected command, you must provide parameters like filenames, folder names...  |

#### Caution

Parameters should sometimes be enclosed by quotation marks, especially when paths contain spaces.

### 90.2 Available uninstall commands

| Command Name   | Parameters       | Description   |
|----------------|------------------|---|
| ucDelFile      | "Path to file"   | The Uninstaller will remove the file specified by Parameters. Example: "file.ext" (quotation marks mandatory)   |
| ucDelFolder    | "Path to folder" | Removes the folder specified by Parameters. Example: "%DESTPAT" (quotation marks mandatory). Only empty directories are supported                           |
| ucRemoveRegKey | "Full Key Path"  | Causes the uninstaller to remove a registry key optionally including "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVer" (quotation marks mandatory) |

| Command Name            | Parameters  | Description   |
|-------------------------|---|---|
| ucRemoveRegValue        | "Full Key Path" "Key Name"                          | Removes a registry entry whose name is "Key Name" and path is "Full Key Path". Example: "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\DisplayVersion" - the two parameters must be enclosed in quotation marks and separated by a space |
| ucRemoveSharedFile      | "Path to file"                                      | The Uninstaller will decrement the shared counter for the file specified. It will remove it if the shared counter reaches 0. Example: "%DESTPATH%\sharename" (quotation marks mandatory)  |
| ucUnregisterSelfRegFile | "Path to file"                                      | Unregister a self-registered file (similar to REGSVR32.EXE -u command). Parameters, but does not delete it. Example: "%DESTPATH%\sharename" (quotation marks mandatory)   |
| ucRemoveFromIni         | "Path to .ini file"<br>"Section Name"<br>"Key Name" | Removes the specified key from the .ini. See below for example.   |
| ucAddComment            | "Comment"   | Add a personal comment to the uninstall log. Useful for support purposes. To add a timestamp, use the %CURDATETIME% variable. For instance: %CURDATETIME% version %ARCPRODVER% installed.   |

### Note

ActiveX (.OCX) and DLL files which were registered using the **"Self-register a file"** custom action are automatically unregistered by the uninstaller if necessary. Have a look at the help topic of this **"Self-register a file"** custom action.

## 90.3 Example for ucRemoveFromIni

You want to remove from "C:\windows\myapp.ini" the following key:

```
[Settings] Path=C:\Program Files\My Application.
```

The Parameters property should be set to:

```
"C:\windows\myapp.ini" "Settings" "Path"
```

## 90.4 Upgrading from Paquet Builder 2.9

This custom action has been completely redesigned for recent versions of Paquet Builder. Some old uninstaller commands were superseded because the uninstaller now accepts custom actions. Here is a guide to replace old commands with new actions:

- ucDelWildFiles ⇒ replace it with [Delete a folder](#) or [Remove single file or using wildcard](#).
- ucRunBat ⇒ replace it with [Execute a BAT file](#).
- ucRunFile ⇒ replace it with [Execute a program](#).
- ucShowMsg ⇒ replace it with [Show a message box](#).

## 90.5 Tips



### Tip

[Execute a program](#) is useful for example if you would like to unregister ActiveX controls manually, launch additional cleaning utilities.

### Action Catalog



Browse all available custom actions.

# 91. Install MSI Package or Update

This action lets you install a Windows Installer MSI package or MSP update.

## Note

Please see [Windows Installer with Paquet Builder](#) topic.

## 91.1 Properties

| Property                    | Data Type | Description  |
|-----------------------------|-----------|--|
| <b>AdditionalParameters</b> | String    | Additional parameters or switches that you want to pass to the Windows Installer runtime (msiexec: see <a href="http://technet.microsoft.com/en-us/library/cc759262%28v=ws.10%29.aspx">http://technet.microsoft.com/en-us/library/cc759262%28v=ws.10%29.aspx</a> )                                 |
| <b>ApplyUpdateMSP</b>       | Boolean   | Set this property to True if you want to run a MSP (patch/update) package and not a MSI installation   |
| <b>OtherAction</b>          | String    | If you want to use another MSI action than install or patch/update. For advanced users. See below.   |
| <b>PackagePath</b>          | String    | The full path to the .MSI or .MSP installation file you want to run. For example <code>%DESTPATH%\productsetup.msi</code>  |
| <b>VariableForExitCode</b>  | VarName   | Variable that will receive the exit code returned by Windows Installer. For the list of exit codes, please see <a href="http://msdn.microsoft.com/en-us/library/windows/desktop/aa372835%28v=vs.85%29.aspx">http://msdn.microsoft.com/en-us/library/windows/desktop/aa372835%28v=vs.85%29.aspx</a> |

## Tip

You can package and run several Windows Installer MSI installations back-to-back with Paquet Builder and this custom action: just add such a custom action for each MSI you want to run. Each time, the custom action waits for Windows Installer to finish its job before returning. You can check the result of the installation thanks to "VariableForExitCode": if the exit code is incorrect, you can interrupt the execution of next installations and display an error message for instance.

## 91.2 Other MSI actions

If you want to run other actions of Windows Installer such as administrative installation option, advertising a product... then you must use the **OtherAction** field to specify the action you want to run.

For instance, you would like to invoke the MSI installer with the /A option. If you were to run from a command prompt in Windows, you would type: `msiexec /A MyInstall.msi`

To get the same behavior in Paquet Builder, enter /A in the `otherAction` field and the full path to the MSI package in `PackagePath`.

## Action Catalog



Browse all available custom actions.

## 92. Perform Operation on a Variable

This action is important as it lets you manipulate the values of **variables**. You can initialize variables, modify the value of an existing one and perform additional operations on strings.

### 92.1 Properties

| Property         | Data Type | Description  |
|------------------|-----------|--|
| <b>Operation</b> | VarOps    | Determines which action should be performed on the variable. See below for more information. There is a defined set of operations from which you can choose in the list box. |
| <b>Separator</b> | String    | Defines the (separator) character for some operations. Optional.   |
| <b>Value</b>     | String    | The new value to assign to the variable specified in "Variable". Required. This can be a "hard-coded" text or number, another variable or even a <b>resource string</b> .    |
| <b>Variable</b>  | VarName   | Indicates the name of the variable to be changed. All variable names must be like this: %VARIABLE%. Be sure to read the help topic regarding <b>variables</b> .              |

#### Note

All operations are **Unicode-enabled**. Remember that variables are limited in length to 4096 wide characters.

You can perform some operations on the value of the new or existing variable. Below is the description of each operation:

### 92.2 Available operations

| Operation Name     | Description   |
|--------------------|---|
| varAssign          | The variable will receive the value specified by the Value property. It may also be another variable.                               |
| varExtractFilename | Removes the path from the filename specified in Value. If you specified C:\TEMP\FILES.ZIP, then the variable will contain FILES.ZIP |

| Operation Name            | Description  |
|---------------------------|--|
| varExtractFilePath        | Like the operation above, the variable will receive the path from the filename, such as C:\TEMP  |
| varExtractFileExt         | The new variable will receive the extension extracted from the filename specified in Value. If you specified MYAPP.EXE, then the variable will contain .EXE.   |
| varExtractFileDrive       | Returns a string containing the drive portion of a fully qualified path name for the file passed in "Value". For file names with drive letters, the result is in the form <drive>. For file names with a UNC path the result is in the form \\ <servername>\<sharename>. If the given path contains neither style of path prefix, the result is an empty string. |
| varRemoveFileExt          | Removes the extension of the filename. Example: MYZIP.ZIP will become MYZIP  |
| varDirectoryWOBS          | To be used with variables that contain a path. This action removes the directory backslash \ from the end of the path if present. For example C:\My Documents\ will become C:\My Documents. Important: remember that all path variables in Paquet Builder do not have the directory backslash. For instance, %WIN% points to C:\Windows, not C:\Windows.         |
| varDirectoryWBS           | To be used with variables that contain a path. This action adds the directory backslash \ to the end of the path if missing. For example %SYS% will be evaluated to %SYS%.   |
| varRemoveBeforeSep        | Removes all characters before the separator specified in Separator.  |
| varRemoveAfterSep         | Removes all characters after the separator specified in Separator.   |
| varGetShortFilename       | The variable will receive the filename specified in Value converted into the 8.3 DOS filenames format.   |
| varGetLongFilename        | The variable will receive the filename specified in Value converted into the full Windows filename format (including the full path)  |
| varReadValueFromClipboard | The package will retrieve the text from the Windows clipboard and set the resulting variable to it. If the contents of the clipboard is not text compatible, then the variable will be blank.  |

| Operation Name           | Description   |
|--------------------------|---|
| varStoreValueToClipboard | This action is special as it does not modify the variable itself: it actually reads the value of the Variable (the Value field is ignored) and stores it to the Windows clipboard. Imagine you want to put the text "It works!!" to the clipboard; you will need two "perform operation on a variable" custom actions: one to initialize a variable called "%VAR1%" to "It works!!" (select the varAssign operation) and then the second one to copy the text from %VAR1% to the clipboard (just enter %VAR1% in the Variable field, leave Value blank and select varStoreValueToClipboard as the Operation). |
| varReadEnvirVar          | The resulting variable is set to the value read from the Windows environment variable specified by Value. For instance "TEMP" will return the default temporary folder.   |
| varGetFreeSpaceOnDisk    | Determines the free space in bytes available on the disk specified by Value. For example, if you enter C:, the variable will get the total free space available on the C hard disk. Note that the value can also point to an existing folder.   |
| varGetTotalSpaceOnDisk   | Determines the total space in bytes available on the disk specified by Value. For example, if you enter C:, the variable will get the total free and used space available on the C hard disk. Note that the value can also point to an existing folder.   |
| varAddQuotes             | Adds the quote character contained in Separator to the left and right of the string contained in Value. The result is then stored in the variable whose name is specified by Variable. Separator must point to a single-byte character such as " or < or '.   |
| varStringReplace         | Returns a string with occurrences of one substring (string1) replaced by another substring (string2). Variable indicated by "Variable" property is read and contains the original string with the (string1) substring. Separator should be set to string1 (the text you want to be replaced) and Value set to string2 (the new text). After replacement, Variable is updated: its value is set to the new returned string. See examples below. This function is not case-sensitive and will replace any occurrence of string1 by string2.   |
| varNOTBool               | To be used only with variables that contain a Boolean value (1 or 0). The variable will be set to the contrary of the specified value.  |
| varANDBool               | Applies the AND operator (conjunction) to a list of Boolean variables (2 or more required). Specify a list of variables in the "Value" field, each variable must be separated by a <b>semicolon</b> ; Example:<br>%BOOLVAR1%;%BOOLVAR2%;%BOOLVAR3% will perform the following operation:<br>%BOOLVAR1% AND %BOOLVAR2% AND %BOOLVAR3% AND: an expression of the form X and Y is True if and only if both X and Y are True. The result is also a Boolean variable (remember that, in Paquet Builder, Boolean variables are represented by <b>1</b> for true and <b>0</b> for false.                             |

| Operation Name | Description  |
|----------------|--|
| varORBool      | Applies the OR operator (disjunction) to a list of Boolean variables (2 or more required). Specify a list of variables in the "Value" field, each variable must be separated by a <b>semicolon</b> ; Example: %BOOLVAR1%;%BOOLVAR2%;%BOOLVAR3% will perform the following operation: %BOOLVAR1% OR %BOOLVAR2% OR %BOOLVAR3%. OR: an expression of the form X or Y is True if X is True or Y is True or if both are True. The result is also a Boolean variable (remember that, in Paquet Builder, Boolean variables are represented by <b>1</b> for true and <b>0</b> for false. |
| varTrimSpaces  | Removes all spaces located at the beginning and at the end of the contents in Value. Value can contain variables, resources strings and plain text. Result is stored into the variable indicated by "Variable".  |
| varLength      | Get the character length of the string in "Value" and store it into the variable indicated by "Variable".  |
| varLeftString  | Extract N characters from the left side of the string in "Value" and store it into the variable indicated by "Variable". N must be an integer and specified in "Separator".  |
| varRightString | Extract N characters from the right side of the string in "Value" and store it into the variable indicated by "Variable". N must be an integer and specified in "Separator".   |

The default operation is varAssign.

## 92.3 Examples for varStringReplace

**Example 1:** You want to replace "the" by "a" in the following sentence: "The mouse is eaten by the cat".

Create a variable named %MYSTRING% for instance and assign its value to "The mouse is eaten by the cat".

Then add a "Perform operation on a variable" custom action with the following parameters:

```
Operation=varStringReplace
Separator=the
Value=a
Variable=%MYSTRING%
```

You can then display the value of %MYSTRING% using a **Show a message box** custom action. You will get: a mouse is eaten by a cat

**Example 2:** Replacing the \ character by \\ (in C++ code for instance)

Enter the following parameters:

```
Operation=varStringReplace  
Separator=\  
Value=\  
Variable=%MYRES2%
```

If %MYRES2% is set to C:\Program Files\Paquet Builder\ for instance, you will get: C:\\Program Files\\Paquet Builder\\

## Action Catalog →

Browse all available custom actions.

## 93. Open a Document File

This action lets you execute a program or open a document file. This function is useful for opening single or multiple document files at the end of the extraction process, or to execute standard applications.

In order to open a document file, the package automatically asks Windows to run the associated application. For example: to open a bitmap file (.BMP ), the default bitmap editor (generally Microsoft Paint) will be run.

| Property Name | Data Type     | Description  |
|---------------|---------------|--|
| Display       | dwWindowState | Defines the state of the program's window. There is a defined set of window states from which you can choose in the list box.  |
| Filename      | String        | Full path to the file you would like to execute. Example: %DESTPATH%\Readme.txt  |
| Operation     | String        | Verb that specifies the action to be performed. The set of available verbs depends on the particular file or folder. The following verbs are commonly used: open, edit, explorer, print  |
| Parameters    | String        | If you would like to pass any command line options to the executed program. Optional.  |
| Unelevated    | Boolean       | If your package runs with <b>administrative rights (defined by the manifest)</b> , any executable started by the package will also be run with administrative rights. To avoid this, you can turn this property to True. Then, the document file or application opened will be run with standard user rights (unelevated). |

### Note

This feature is also available in the [Program/Setup Execution](#) panel.

### Tip

In order to open your Web page (or Web site) in the user's default Web browser, just do this: enter the full URL (Uniform Resource Locator) of your Web page in the Filename property. At runtime, packages will open the default Web browser to the specified URL. For example: assign "<https://www.gdgsoft.com>" to the Filename property in order to go to G.D.G. Software site.

### Action Catalog

Browse all available custom actions.

## 94. Read Data (or Variables) from a File

This action lets you read information from a text file; for example, you can read parameters or other information useful for the package's execution and initialize some variables.

| Property Name  | Data Type | Description   |
|----------------|-----------|---|
| Filename       | String    | Path to the file (text file) that the package should read (the filename does not necessarily require the .txt extension). |
| ReturnVariable | VarName   | The variable that will store the data read from the file. For example %MYRESULT%.   |

- The action will read data from the specified file until an **'End Of Line'** character is found, so it is generally the first line of the text file.
- The action expects text files to be encoded in **UTF-8 format**.
- The variable will contain an empty string if the read has failed.

### Caution

Do not use this custom action to read large amount of data. Variables in Paquet Builder have limited content to 4096 wide characters.

### Action Catalog

Browse all available custom actions.

## 95. Read an Entry from the Registry

Lets you retrieve information from the Windows Registry. You can for example read information such as the user's name or the destination directory for your files.

| Property Name | Data Type            | Description   |
|---------------|----------------------|---|
| Access        | regaccess            | Request access to the 32-bit or 64-bit Windows Registry view. Please refer to the paragraph entitled <a href="#">Accessing an Alternate Registry View</a> . Default is <code>regaDefault</code> |
| Ident         | String               | The named entry within the registry key. Leave this field blank to read the default value associated to the key.  |
| Key           | String               | The key name of the registry entry. Like <code>SOFTWARE\My App</code>   |
| RootKey       | hkKeys               | The base key of your registration entry. There is a defined set of Base Keys from which you can choose in the list box.   |
| Variable      | <code>VarName</code> | The name of the variable whose value will be set to what has been read. If the entry is not found, a blank value is returned.   |

Paquet Builder can read the following entry types:

- REG\_SZ
- REG\_EXPAND\_SZ
- REG\_DWORD
- REG\_QWORD

It automatically converts them to string values when storing them in variables.

### 95.1 Reading destination path from registry

You can use the feature to read the default destination path from the Windows Registry. However do not assign the path to the `%DESTPATH%` variable which is read-only. First read the path to a specific variable like `%MYPATH%` and then in the default destination folder field, enter `%MYPATH%`. For more information about setting the destination folder, please [click here](#).

## Caution

If you want to read registry data from `HKEY_LOCAL_MACHINE`, please note that 32-bit packages can't access `HKEY_LOCAL_MACHINE\SOFTWARE` directly, but the `Wow6432Node` key instead. See [this article](#) for more details.

The solution is to use the Access property and choose `regWow64_64KEY`. Or better you can consider [making a 64-bit package with Paquet Builder](#).

## Action Catalog

Browse all available custom actions.

## 96. Read a Key from an .ini File

This custom action retrieves a string from the specified section in an initialization file. Some applications may still work with .ini files although Win32-based applications should store initialization information in the [registry](#).

Format of an INI file:

```
[AppName]
Key=Value
```

| Property Name | Data Type      | Description   |
|---------------|----------------|---|
| AppName       | String         | Specifies the section containing the key name. This section name is typically the name of the calling application. Do not add the enclosing brackets [ ] yourself.  |
| IniFile       | String         | Points to a string that names the initialization file. Do not forget the .ini extension and its path. Example: <i>%WIN%\Myappli.ini</i>   |
| Key           | String         | Contains the name of the key to be associated with a string.  |
| Variable      | <b>VarName</b> | This is the variable that will store the result read from the .ini file. For example <i>%MyIniValue%.\ %MyIniValue%</i> can be any value, such as a path to a file or a number. If the key is not found, a blank value is returned. |

### Action Catalog [→](#)

Browse all available custom actions.

## 97. Refresh Windows Shell Icons

---

This action will refresh the Windows shell icons. This custom action is useful if you change file associations for instance. To be used with the [Register File Association action template](#).

No parameters.

### Action Catalog



Browse all available custom actions.

## 98. Register a Shared File

This action lets you register the specified file with Windows as a shared legacy file. Shared means that this file may be used by different applications, and consequently should be removed with care.

It uses the Windows' shared file counting feature (located in the registry at `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\SharedDLLs`). Each time the file is registered, the reference count for the file is incremented. When an application using the file is removed, the reference count is decremented. If the count reaches zero, the file is deleted.

| Property Name      | Data Type | Description   |
|--------------------|-----------|---|
| AddUninstReference | Boolean   | If true, the package will add a reference to this file in the <b>uninstaller log file</b> . The file will be then removed only if no more used. |
| SharedFilename     | String    | Full path to the file you want to register as shared. For example <code>%SYS%\Myd11.d11</code>  |

### Action Catalog [→](#)

Browse all available custom actions.

## 99. Remove a Registry Key or Entry

Lets you delete a registry entry or a registry key from Windows Registry.

| Property Name       | Data Type | Description   |
|---------------------|-----------|---|
| Access              | regaccess | Request access to the 32-bit or 64-bit Windows Registry view. Please refer to the paragraph entitled <a href="#">Accessing an Alternate Registry View</a> . Default is <code>regaDefault</code> |
| Ident               | String    | The named entry within the registry key. For example <code>AppPath</code> . Leave this field blank to remove a registry key.  |
| Key                 | String    | The key name of the registry entry. Like <code>SOFTWARE\My App</code> .   |
| RemoveSubKeysIdents | Boolean   | If true, all subkeys will be removed too.   |
| RootKey             | hkKeys    | The base key of your registration entry. There is a defined set of Base Keys from which you can choose in the list box.   |

### Tip

To remove a registry key, just leave the Ident field blank.

### Tip

You can use this custom action for the uninstaller in order to remove additional registry entries or keys.

### Caution

Be careful when using this custom action, especially if `RemoveSubKeysIdents` is set to `True`.

### Caution

`RemoveSubKeysIdents` is not available if `Access` is different from `regaDefault`.

### Action Catalog

Browse all available custom actions.

## 100. Remove Single File or Using Wildcard

This action is used to remove a single file or an entire group of files (using a wildcard).

| Property Name   | Data Type | Description  |
|-----------------|-----------|--|
| Filename        | String    | If UseWildcard is set to false, then it should point to the full path to the file to delete. Example: %DESTPATH%\Temp\RegOCX.Bat If UseWildcard is set to true, then it should be the path + the source wildcard. For instance %DESTPATH\Templates\*.* |
| RemoveAtStartup | Boolean   | This will register the file to be removed by Windows on next reboot.   |
| UseWildcard     | Boolean   | To remove a single file, leave this property to false. If you want to delete an entire group of files based on a source wildcard, turn this property on.   |

### Tip

Also have a look at the [Delete a folder](#) custom action.

### Tip

You can use this custom action for the uninstaller in order to remove additional files.

### Action Catalog →

Browse all available custom actions.

# 101. Rename a File

You can use this action to rename a file.

| Property Name | Data Type | Description                       |
|---------------|-----------|-----------------------------------|
| NewFilename   | String    | This is the file's new name.      |
| OrigFilename  | String    | This is the file's original name. |

## Note

This function does not add any reference of the renamed file to the uninstaller log file.

## Action Catalog

Browse all available custom actions.

## 102. Replace Variables in File

This action lets you replace all variable holders in the contents of an existing text file. This file is then saved to another file.

This custom action is useful if you want to use templates for configuration files like INI, REG, TXT, XML... and store values of variables, to set initial options for your program or store information about the package's execution, such as the user's name or the destination directory. See example below.

| Property Name | Data Type | Description  |
|---------------|-----------|--|
| DestFile      | String    | Path to the destination text file that the package will create (the filename does not necessarily require the .txt extension). Variables are possible. If the file exists, it will be overwritten. |
| SourceFile    | String    | Path to the existing source text file that the package should use as a template (the filename does not necessarily require the .txt extension). Variables are possible.                            |

### 102.1 Notes

- If the destination text file exists, it is overwritten.
- The **uninstaller** will not remove this file unless you **specify it**.
- All existing variables and resource strings are replaced.

### 102.2 Example

You want to store the destination path in an INI file for your application.

Here is an INI sample:

```
[Parameters]
Destination=%DESTPATH%
IsWin64=%IS64OS%
```

- 1 Store this content in a template file named `template.ini`. Then add it to the package's overhead thanks to the Loader Temporary Resources in **Advanced Build Options**. This will ensure the `TEMPLATE.INI` file will be available when the package runs, in the temporary folder. Since this is a template file only, it is only required temporarily. The package will erase it when it closes.

- 2 Add the custom action "Replace variables in file" to the "After File Extraction" event, with these properties:

```
SourceFile=%PBSFXPATH%\Template.ini  
DestFile=%DESTPATH%\Settings.ini
```

- 3 %DESTPATH%\Settings.ini tells the package to store the new INI file with variables correctly replaced in the destination folder of our application, with the filename "settings.ini", so that our application can find it.

## Action Catalog



Browse all available custom actions.

## 103. Restart Computer

---

This action instructs the package to **restart the system when it closes**. The user is **not prompted**, so it falls to you to inform users that their computer will be restarted.

No parameters.

### Caution

Be sure to call this custom action at the end when you are sure that the system needs to be restarted. This custom action is useful if some **extracted files could not be replaced** and **%NEEDREBOOT%** is set to "1".

### Action Catalog [→](#)

Browse all available custom actions.

## 104. Self-Register a File (ActiveX, OLE)

This action registers .dll and ActiveX control files as command components in Windows registry. It is similar to Windows REGSVR32.EXE utility except that you do not need to ship this tool with your package if you use this custom action.

Finally, if the package includes an uninstaller, this custom action may also put a reference into the [uninstaller log](#), so the registered DLL or OCX file is unregistered when the package is uninstalled by end users.

| Property Name      | Data Type | Description   |
|--------------------|-----------|---|
| AddUninstReference | Boolean   | If true, the package will add a reference to this file in the <a href="#">uninstaller log file</a> . The file will be first unregistered and then safely removed if no other application uses it. |
| ServerFile         | String    | Full path to the file you want to self-register. The file must exist when this custom action is executed. For example <code>%SYS% Control.ocx</code>  |

### Tip

This custom action should always be put in the "After File Extraction" event, especially if AddUninstReference is set to True.

### Tip

You can also use [Register a shared file](#) custom action to mark the file as shared.

### Action Catalog [→](#)

Browse all available custom actions.

## 105. Set File Attributes

---

This action lets you modify the attributes (archive, read-only...) of a file.

| Property Name | Data Type | Description  |
|---------------|-----------|--|
| Attributes    | atAttrs   | Defines the new attributes to set up. There is a defined set of attributes from which you can choose in the list box |
| Filename      | String    | Full path to the requested file. Example: %DESTPATH%\MyExt.dll   |

### Action Catalog [→](#)

Browse all available custom actions.

## 106. Show an Information Panel

This custom action creates a dialog box displaying rich text to your end users. It is similar to the [Show a Readme dialog](#) custom action except that **all variables** found in the rich text can be optionally replaced by their values.

| Property Name       | Data Type | Description  |
|---------------------|-----------|--|
| Caption             | String    | The title that appears on the dialog box or in the wizard's <a href="#">instruction panel</a> .  |
| EvalForVar          | Boolean   | Indicates whether the rich text should be scanned for <b>variables</b> before it is displayed. If true, all variables found will be replaced by their values.                      |
| NextCaption         | String    | Holds the text of the Next button or Continue button. You can insert your own text. Otherwise, leave it to #Next or #Continue resource strings.                                    |
| ForceMarkdownViewer | Boolean   | When True, forces the legacy Markdown viewer even if WebView2 is available on the end user's system. Default: False. Use this if you need consistent rendering across all systems. |
| Text                | RichLines | The text that should be displayed in the text box in Rich Text Format (RTF). Click the '...' button to edit it with <a href="#">Rich Content Editor</a> .                          |
| WizardDescription   | String    | An additional description for the information panel that will appear in the instruction bar (only used if you enable the Wizard <a href="#">interface theme</a> ).                 |

### Tip

This dialog can be used to replace the Welcome/End screens. You can also use it to show a “setting summary” before file extraction for instance.

### Action Catalog [→](#)

Browse all available custom actions.

## 107. Show a License Agreement

You can use this action to show your own license agreement text. Users must agree with it before continuing (they can choose between Yes or No). Clicking No will immediately close the package.

| Property Name       | Data Type | Description  |
|---------------------|-----------|--|
| Caption             | String    | The title that appears on the dialog box or in the wizard's <a href="#">instruction panel</a> .  |
| EvalForVar          | Boolean   | Indicates whether the rich text should be scanned for variables before it is displayed. If true, all variables found will be replaced by their values.                             |
| LicEnd              | String    | The text which appears after the License Agreement and asks end users to choose between Yes or No. You can use the standard resource string <code>#LicEnd</code>                   |
| LicIntro            | String    | The text which invites end users to read your License Agreement. You can use the standard resource string <code>#LicIntro</code>   |
| ForceMarkdownViewer | Boolean   | When True, forces the legacy Markdown viewer even if WebView2 is available on the end user's system. Default: False. Use this if you need consistent rendering across all systems. |
| Text                | RichLines | Your License Agreement in Rich Text Format (RTF). Click the '...' button to edit it with <a href="#">Rich Content Editor</a> .   |

### Note

Displaying a License Agreement is also available in the [License Agreement page](#).

### Action Catalog

Browse all available custom actions.

## 108. Show a Message Box

This action lets you display a message box (or prompt message). This message box can show a standard message (it returns no value) or a question (the message box returns a value). Paquet Builder uses this feature to display its [default prompt messages](#).

### Note

Paquet Builder determines whether the message box will be a standard text or a question based on the *VariableReturn* property. If you leave this property blank, then Paquet Builder will show a standard text (even if you set the Buttons property to pbOkCancel). Otherwise, the message box will be defined as a question and its return value will be stored in the variable specified by *VariableReturn*.

| Property Name  | Data Type      | Description   |
|----------------|----------------|---|
| Caption        | String         | The title that appears on the message box. For example: <i>Confirmation</i>   |
| MessageBtn     | pbButtons      | Defines the button(s) that will appear. There is a defined set of buttons from which you can choose in the list box.  |
| MessageType    | pslcons        | Defines the icon that will appear on the message box. There is a defined set of icons from which you can choose in the list box.  |
| Text           | RichLines      | The text that will appear on the message box, as "This will install MyApp./\$Do you want to continue?". Tip: to put a carriage-return-line-feed between "Line 1" and "Line 2", use /\$, like "Line1/\$Line2". |
| VariableReturn | <b>VarName</b> | Optional: if the prompt is a question, enter the variable where the return value will be stored. Like %MYANSWER%. See below about possible return values.   |

### Tip

If the VariableReturn is not blank, then the variable specified by this property will receive the result of the end user's choice when the latter select one of the buttons (OK, Cancel, Yes or No). If the value is positive (Yes, OK), then the variable will receive the value **1** (true). Otherwise, if the value is negative (No, Cancel), then it will contain **0** (false).

### Action Catalog

Browse all available custom actions.

## 109. Show a Readme Dialog

You can use this action to show a readme text (additional instructions, etc.).

| Property Name         | Data Type | Description  |
|-----------------------|-----------|--|
| Caption               | String    | The title that appears on the dialog box or in the wizard's <a href="#">instruction panel</a> .  |
| EvalForVar            | Boolean   | Indicates whether the rich text should be scanned for variables before it is displayed. If true, all variables found will be replaced by their values. Ignored if <code>ReadFromFileAtRuntime</code> is used.  |
| ReadFromFileAtRuntime | String    | Optional: instead of specifying the text to be displayed using the <code>Text</code> property below, you can cause the package to read the text from an external RTF file. In this case you will need to specify the full path to this RTF file: generally you can use this option after the file extraction to display a final readme for example <code>(%DESTPATH%\Readme.rtf)</code> . Note that only true rich text files may be loaded. If this option is used, the <code>Text</code> property will be ignored. |
| ForceMarkdownViewer   | Boolean   | When True, forces the legacy Markdown viewer even if <code>WebView2</code> is available on the end user's system. Default: False. Use this if you need consistent rendering across all systems.  |
| Text                  | RichLines | Your Readme Dialog in Rich Text Format (RTF). Click the '...' button to edit it with <a href="#">Rich Content Editor</a> .   |
| WizardDescription     | String    | An additional description for the readme that will appear in the instruction bar (only used if you enable the Wizard <a href="#">interface theme</a> ).  |

### Note

Also available in the [Readme's page](#).

### Action Catalog

Browse all available custom actions.

## 110. Show/Hide a Wait Message Dialog

This action lets your package display or hide a wait message dialog: this dialog does not contain any button, it works as an indicator to inform end users what the package is doing. You can see similar messages in standard Setup programs (“Please wait while creating shortcuts...”).

The message dialog is non-modal: that means the package will continue its execution during the display of this message.

| Property Name | Data Type  | Description  |
|---------------|------------|--|
| TextToDisplay | String     | The text that should appear on the dialog box. Something to inform your end users as Please wait while .... Please use a short sentence!   |
| WhatToDo      | TswmAction | Two possible values: <code>swmShowMessage</code> causes the package to display the wait message dialog. <code>swmHideMessage</code> causes the package to destroy the wait message dialog. |

### Tip

You should always start with an `swmShowMessage` action to display the wait dialog first. Then if you just want to change the dialog’s text without closing the dialog itself, use the `swmShowMessage` action again: only the text will be changed.

### Caution

Always close the wait dialog using the `swmHideMessage` action before the package ends.

### Note

The wait message dialog should not be used with modal dialogs (all other dialogs like message boxes, wizard pages...). Use it when the package performs non visual operations like shortcut creations, file copy...

### Action Catalog

Browse all available custom actions.

# 111. Sleep (Pause Program Execution)

---

This action suspends the package execution until the number of seconds specified by **SecondsToSleep** has elapsed.

This custom action can be used in combination with [Show/Hide a wait message dialog](#).

| Property Name  | Data Type | Description  |
|----------------|-----------|--|
| SecondsToSleep | Integer   | Number of seconds to wait for. The value is limited to a maximum of 65535 seconds. |

## Action Catalog



Browse all available custom actions.

## 112. User Comment

---

This action has no functionality! It just lets you insert a comment for yourself in the action list.

| Property Name | Data Type | Description                     |  |
|---------------|-----------|---------------------------------|--|
| Comment       | String    | The comment you wish to insert. |  |

### Action Catalog [→](#)

Browse all available custom actions.

## 113. Write an Environment Variable

This action lets you set a persistent Windows environment variable. It writes the value to the Windows registry and notifies running processes of the change via `WM_SETTINGCHANGE`, so applications can pick up the new value without a reboot.

You can choose to write to the **user** environment (available only to the current user) or the **system** environment (available to all users on the machine). Writing to the system environment requires administrator privileges.

| Property Name    | Data Type    | Description  |
|------------------|--------------|--|
| VarName          | String       | The name of the environment variable to set (e.g., <code>JAVA_HOME</code> , <code>MY_APP_DIR</code> ). You can use <a href="#">variables</a> .   |
| Value            | String       | The value to assign to the environment variable. You can use <a href="#">variables</a> such as <code>%DESTPATH%</code> .   |
| Scope            | TEnvVarScope | <code>evsUser</code> writes to the current user's environment ( <code>HKEY_CURRENT_USER\Environment</code> ). <code>evsSystem</code> writes to the system environment ( <code>HKEY_LOCAL_MACHINE\...\Environment</code> ) and requires admin rights. Default is <code>evsUser</code> .   |
| ExpandableString | Boolean      | When true, the value is stored as <code>REG_EXPAND_SZ</code> (expandable string), allowing references to other environment variables like <code>%SystemRoot%</code> . When false (default), the value is stored as <code>REG_SZ</code> (regular string). Use <code>REG_EXPAND_SZ</code> when appending to variables like <code>PATH</code> . |
| AddUninstRef     | Boolean      | When true (default), the environment variable will be removed by the <a href="#">uninstaller</a> — but <b>only if the variable did not already exist</b> before the installer ran. If the variable already existed (e.g., <code>PATH</code> ), the uninstaller will leave it untouched to avoid breaking the system.                         |

### Caution

Writing to the system environment (`evsSystem`) requires the installer to run with administrator privileges. Make sure your package has the appropriate [UAC manifest](#) settings.

### Caution

Be careful when overwriting existing environment variables. This action replaces the entire value of the variable. If you need to **append** to an existing variable like `PATH`, first read its current value using the [Modify a variable](#) action with the `Read environment variable` operation, then concatenate your new path, and finally write the result back with this action.

## 113.1 Example 1: Set a new application directory variable

To create a user environment variable `MY_APP_HOME` pointing to the installation directory:

- 1 Set `VarName` to `MY_APP_HOME`
- 2 Set `Value` to `%DESTPATH%`
- 3 Set `Scope` to `evsUser`
- 4 Leave `ExpandableString` as `False`
- 5 Leave `AddUninstRef` as `True` — the variable will be removed on uninstall since it did not exist before

## 113.2 Example 2: Add to system PATH

To add your application's bin directory to the system PATH:

- 1 First, use a **Modify a variable** action to read the current `PATH` value into a variable (e.g., `%OLDPATH%`)
- 2 Then use this action:
  - `VarName` = `Path`
  - `Value` = `%OLDPATH%;%DESTPATH%\bin`
  - `Scope` = `evsSystem`
  - `ExpandableString` = `True` (PATH uses expandable strings)
  - `AddUninstRef` = `False` (since PATH already exists, set this to False)

### Action Catalog



Browse all available custom actions.

## 114. Write a Registry Entry

This action lets you write information directly into Windows registry; for example to set initial options for your program or store information about the package's execution, such as the user's name or the destination directory. Storing this kind of information in Windows registration database makes it easy for your program to access it later.

| Property Name | Data Type | Description  |
|---------------|-----------|--|
| Access        | regaccess | Request access to the 32-bit or 64-bit Windows Registry view. Please refer to the paragraph entitled <a href="#">Accessing an Alternate Registry View</a> . Default is regaDefault       |
| AddUninstRef  | Boolean   | Indicates whether the registry entry should be removed by the uninstaller or not. By default it is true.   |
| Ident         | String    | The named entry within the registry key. Leave this field blank to read the default value associated to the key.   |
| Key           | String    | The key name of the registry entry. Like SOFTWARE\My App   |
| RootKey       | hkKeys    | The base key of your registration entry. There is a defined set of Base Keys from which you can choose in the list box.  |
| Value         | String    | This is the data contained by the named entry. You can use variables such as %DESTPATH%  |
| ValueType     | regType   | Paquet Builder supports working with four types of registry entries: string (REG_SZ), dword (REG_DWORD and REG_QWORD) and expanded strings (REG_EXPAND_SZ). Choose the type in the list. |
| Volatile      | Boolean   | Marks the registry key as volatile (default, false)  |

### Caution

All values are automatically overwritten. Be sure that you won't overwrite important information.

### 114.1 Example

To create the key `HKEY_CURRENT_USER\Software\MyCompany\MyWordPad` and set its `AppPath` entry to the location where your software was installed:

- 1 Select `hkCurrentUser` for the `Rootkey` and type `Software\MyCompany\MyWordPad` for the `Key`.
- 2 Add `AppPath` to the `Ident` entry
- 3 Set `ValueType` to `REG_SZ`
- 4 Finally enter `%DESTPATH%` (the `variable` which contains the final destination folder) in the `value` field.

## Action Catalog



Browse all available custom actions.

# 115. Write Text (or Variables) to a File

This action lets you write information to a text file; for example to set initial options for your program or store information about the package's execution, such as the user's name or the destination directory. All variables can be stored into the text file (additional settings for your application for example).

Another alternative: [replace variables in file](#)

| Property Name   | Data Type | Description   |
|-----------------|-----------|---|
| AppendCRLFAtEnd | Boolean   | If true, write the string to the file and add an 'end of line' character.   |
| Contents        | String    | String that should be added to the file. Can accept variables.  |
| Filename        | String    | Path to the file (text file) that the package should create or modify (the filename does not necessarily require the .txt extension). |

## Caution

Do not use this custom action for writing large amounts of data. The Contents property is limited to 4096 wide characters. You can repeat this action several times for the same file and with different content, since contents is added to the end of the existing file.

## Notes

- If the text file exists, the string is added at the end.
- The [uninstaller](#) will not remove this file unless you [specify it](#).
- All existing variables and resource strings are replaced.

## Action Catalog

Browse all available custom actions.

## 116. Write a Value to an .ini File

This custom action replaces the keys and values under the specified section in an initialization file. Although you should use [registry](#) to store application settings, some applications may still work with .ini files.

Format of an INI file:

```
[AppName]
Key=Value
```

| Property Name | Data Type | Description  |
|---------------|-----------|--|
| AddUninstRef  | Boolean   | Indicates whether the <a href="#">uninstaller</a> will remove the entry added to the .ini file by this custom action.  |
| AppName       | String    | Specifies the section containing the key name. This section name is typically the name of the calling application. Do not add the enclosing brackets [ ] yourself. |
| IniFile       | String    | Points to a string that names the initialization file. Do not forget the .ini extension and its path. Example: <i>%WIN% Myappli.ini</i>                            |
| Key           | String    | Contains the name of the key to be associated with a string.   |
| Value         | String    | Points to a string to be written to the .ini file. You can of course use <a href="#">variables</a> .   |

### Note

If your application uses a non-common .ini file (common .ini are win.ini for example), then you should [add a reference](#) to the uninstaller manually; so the uninstaller will remove the entire .ini file and not only its entries.

### Tip

If you want to create a new .ini with some pre-defined entries, you can have a look at the custom action titled "[Write text \(or variables\) to a file](#)".

### Action Catalog

Browse all available custom actions.

# 117. XML - Begin Editing File or String

This custom action lets you start manipulating XML data. It can read XML data from existing XML file or from a string.

All XML-related custom actions require an XML ID that is created with this custom action "Begin editing file or string": thus, it must be the first custom action executed when you want to edit XML.

| Property Name | Data Type | Description   |
|---------------|-----------|---|
| XMLLoadWay    | TXMLLoad  | If set to <code>xmlLoadFromFile</code> , the XML file whose path is specified with <code>Filename</code> is opened (the file must exist). If set to <code>xmlLoadFromString</code> , the XML resource is loaded from the string directly entered in the <code>Filename</code> property. |
| Filename      | String    | Full path to the XML file that should be loaded, or a string that contains an entire XML document or a well-formed fragment (can use <b>PB variables</b> ).   |
| IDXML         | TXMLID    | A unique string identifier that will be associated to the opened XML resource.  |

## 117.1 Examples

### Tip

When using `xmlLoadFromFile`, the XML file must exist. Variables are accepted: `%DESTPATH%\my file.xml`

### Tip

If you want to create an XML file from scratch, use `xmlLoadFromString` and provide the contents of the XML document in the `Filename` property. For instance:

```
JoeDoe
```

## 117.2 Best practices

## Always include an encoding declaration

When creating or editing XML files that will be read by Paquet Builder custom actions, always include an explicit encoding declaration at the top of the file:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Omitting the encoding attribute (e.g., `<?xml version="1.0"?>`) can cause parsing failures on non-English Windows systems due to codepage detection differences in the MSXML parser.

## Self-closing elements

XML allows self-closing (empty) elements: `<System />` is equivalent to `<System></System>`. If your XML file may contain self-closing elements and your custom actions read child nodes inside them (e.g., reading `//System/Backuppath` when `<System />` has no children), the read/write actions will safely skip the operation and leave the target variable untouched. See the documentation for each XML action for details.

## Action Catalog

Browse all available custom actions.

## 118. XML - End Editing File (and Save)

This custom action lets you close an XML resource being edited, optionally save XML data to an XML file and free memory.

### Caution

All XML custom actions must terminate with this “End editing file” action in order to free memory and resources.

| Property Name | Data Type | Description  |
|---------------|-----------|--|
| Filename      | String    | Full path to the XML file to be saved (can use <a href="#">PB variables</a> ).   |
| IDXML         | TXMLID    | A unique string identifier associated to the opened XML resource.  |
| SaveFile      | Boolean   | If set to True, the XML file is saved. If set to False, no XML data is saved: all changes are lost. Generally, you'll use False if you were just reading data from existing XML files. |

### Action Catalog

Browse all available custom actions.

# 119. XML - Read Attribute Value from Node

This custom action lets you read the value of an attribute associated to an XML node.

## Caution

This custom action requires an XML ID that is first created with "Begin editing file or string".

| Property Name | Data Type | Description   |
|---------------|-----------|---|
| AttributeName | string    | A string specifying the name of the attribute to return.  |
| IDXML         | TXMLID    | A unique string identifier associated to the opened XML resource you want to read data from.  |
| PathToNode    | XPath     | XPath expression to the node that contains the attribute.   |
| Variable      | VarName   | The name of the variable whose value will be set to what has been read. If the node attribute can't be found or read, the variable is left untouched. |

## 119.1 Behavior when the node does not exist

If the XPath expression does not match any node in the XML document (for example, if the parent element is self-closing like `<Config />` instead of `<Config key="value">`), the read operation is silently skipped and the target variable is left untouched.

## Tip

If you need to handle both cases (node present and node absent), you can initialize the variable to a default value before the read action. If the node and attribute exist, the variable will be overwritten; if either is missing, it will keep the default value.

## 119.2 PathToNode explanation

Use of XPath is explained at <http://msdn.microsoft.com/en-us/library/ms256086.aspx>

### Action Catalog



Browse all available custom actions.

## 120. XML - Read Value from Node

This custom action lets you read the text content of an XML node.

### Caution

This custom action requires an XML ID that is first created with **"Begin editing file or string"**.

| Property Name | Data Type | Description   |
|---------------|-----------|---|
| IDXML         | TXMLID    | A unique string identifier associated to the opened XML resource you want to read data from.  |
| PathToNode    | XPath     | XPath expression to the node.   |
| Variable      | VarName   | The name of the variable whose value will be set to what has been read. If the node can't be found or read, the variable is left untouched. |

### 120.1 Behavior when the node does not exist

If the XPath expression does not match any node in the XML document (for example, if the parent element is self-closing like `<System />` instead of `<System><Backuppath>value</Backuppath></System>`), the read operation is silently skipped and the target variable is left untouched.

### Tip

If you need to handle both cases (node present and node absent), you can initialize the variable to a default value before the read action. If the node exists, the variable will be overwritten with the node's value; if the node is missing, it will keep the default value.

### 120.2 PathToNode explanation

Use of XPath is explained at <http://msdn.microsoft.com/en-us/library/ms256086.aspx>

#### Action Catalog

Browse all available custom actions.

# 121. XML - Write Node Attribute

This custom action lets you write the value of an attribute associated to an XML node.

## Caution

This custom action requires an XML ID that is first created with "[Begin editing file or string](#)".

| Property Name  | Data Type | Description   |
|----------------|-----------|---|
| AttributeName  | string    | A string specifying the name of the attribute to create or change.                          |
| AttributeValue | string    | Value of the attribute you want to set. You can use variables such as %DESTPATH%.           |
| IDXML          | TXMLID    | A unique string identifier associated to the opened XML resource you want to write data to. |
| PathToNode     | XPath     | XPath expression to the node that contains the attribute.                                   |

## 121.1 PathToNode explanation

Use of XPath is explained at <http://msdn.microsoft.com/en-us/library/ms256086.aspx>

### Action Catalog

Browse all available custom actions.

## 122. XML - Write Node

This custom action lets you set the text content of an existing XML node, or create a new node.

### Caution

This custom action requires an XML ID that is first created with **"Begin editing file or string"**.

This custom action has two different behaviors, depending on whether Name is set or not.

- If you leave Name blank, the node whose XPath is defined by PathToNode will have its text set to the contents specified with "Value". The node must exist.
- If you set Name, a new node with this name is created. The node parent is defined by PathToNode. The node text is also set to the contents specified with "Value".

| Property Name | Data Type | Description  |
|---------------|-----------|--|
| IDXML         | TXMLID    | A unique string identifier associated to the opened XML resource you want to write data to.  |
| Name          | String    | Name of the node. Optional: see above  |
| PathToNode    | XPath     | XPath expression to the parent node (if you create a new XML node) or to the existing node (if you want to change its text content). |
| Value         | String    | This is the text you want to assign to the node. You can use <b>variables</b> such as %DESTPATH%.                                    |

### 122.1 Behavior when the node does not exist

- **Update mode** (Name is blank): If the XPath expression does not match any existing node (for example, if the parent element is self-closing like `<System />` instead of `<System><Backuppath>value</Backuppath></System>`), the write operation is silently skipped — no error occurs and no node is created.
- **Create mode** (Name is set): If the parent node specified by PathToNode does not exist, the new child node is not created.

 **Tip**

If you are reading values from a third-party XML file that may have self-closing empty elements (`<Element />`), make sure to handle the case where child nodes may not exist. You can use the **Create mode** (set Name) to ensure the node is created before writing to it.

## 122.2 PathToNode explanation

Use of XPath is explained at <http://msdn.microsoft.com/en-us/library/ms256086.aspx>

### Action Catalog →

Browse all available custom actions.

## 123. Automation and Command Line

This section covers the automation features of Paquet Builder: command-line builds, directive files, exit codes, and related tools.

### When to use these topics

Use this section when you want repeatable builds, batch workflows, or integration with external automation systems. Directive files are plain-text INI files — you can create them with any text editor, store them in version control, and generate them from scripts or CI/CD pipelines.

#### Command-line Builder [→](#)

Launch Paquet Builder with command-line switches for scripted workflows.

#### Console Compiler [→](#)

Compile packages from a portable console tool without opening the full UI.

#### Directive Files [→](#)

Simple plain-text INI files (.pbd) that describe build settings — easy to create, version, and share.

#### Execute Directives [→](#)

Run directive files to automate project changes or builds without opening the GUI.

#### Builder Exit Codes [→](#)

Exit codes returned by Paquet Builder during compilation.

#### Package Installer Exit Codes [→](#)

Exit codes returned by packages and installers at runtime.

#### Package Command-line Switches [→](#)

Define custom command-line switches for your packages.

### 123.1 Automation via MCP Server

Beyond command-line tools and directives, Paquet Builder also exposes an **MCP (Model Context Protocol) server** that lets AI agents and external tools create, modify, and compile projects programmatically. This is ideal for AI-driven workflows, CI/CD pipelines, or any scenario where you want to control Paquet Builder from another application.

## MCP Server Overview →

Learn how the MCP server works and what it can automate.

## MCP Server Setup →

Install and configure the MCP server for your environment.

## MCP Server Tools →

Browse all available MCP tools for project management and builds.

# 124. Command-line options and switches for Paquet Builder

Paquet Builder supports command-line options to automate tasks such as managing project files and compiling packages. These options allow you to bypass the graphical interface for efficiency or integration into automation workflows.

## Tip

For advanced automation, consider using the [console command-line compiler](#), which is specifically designed for CI/CD and headless environments.

## 124.1 What is a switch?

A switch is a command-line parameter, often accompanied by a value, used to configure how Paquet Builder runs. Switches are specified with a forward slash /.

For example, `/c` is the switch that triggers compilation.

## Handling spaces in paths

Use quotation marks (") for paths with spaces:

```
/log:"C:\Output Folder\Log.html"
```

Do **not** use quotes if the path has no spaces:

```
/log:C:\OutputFolder\Log.html
```

## 124.2 Launching Paquet Builder from the command line

You can specify a project file when launching Paquet Builder:

```
PBUILDER.EXE "C:\Projects\MyProject.pbp"
```

This opens the project file. Pressing the **Compile** button creates the package.

When a command-line switch is included, the splash screen is hidden and a taskbar window displays progress.

## 124.3 Supported switches

| Switch                         | Description  |
|--------------------------------|--|
| <code>/c</code>                | Compile the specified project or directive file.   |
| <code>/q</code>                | Exit Paquet Builder after compilation, regardless of success or failure (error messages are displayed).  |
| <code>/s</code>                | Non-interactive mode: suppresses progress dialogs. Errors still prompt interaction. For true silent mode, use the <a href="#">console compiler</a> . |
| <code>/log:"path"</code>       | Save the compilation log to an HTML file at the specified path.  |
| <code>/outfolder:"path"</code> | Override the output directory for the compiled package.  |
| <code>/outfname:"name"</code>  | Specify a custom filename for the compiled package.  |
| <code>/exetype:N</code>        | Set the target architecture: 0 = 32-bit, 1 = 64-bit, 2 = ARM64.  |
| <code>/arcfile:"path"</code>   | Change the path of the external 7z archive file.   |
| <code>/filelistupt</code>      | Force a live update of all file lists and components before compilation.   |

## 124.4 Examples

Compile the project silently and exit:

```
PBUILDER.EXE "C:\Projects\MyProject.pbp" /c /q /s
```

Change the output directory and save the log:

```
PBUILDER.EXE "C:\Projects\TestProject.pbp" /c  
/log:"C:\Logs\BuildLog.html" /outfolder:"C:\Output"
```

Build a 64-bit package, refresh file lists, and compile silently:

```
PBUILDER.EXE "C:\Projects\TestProject.pbp" /exetype:1 /c /q /s  
/filelistupt
```

Build an ARM64 package, refresh file lists, and compile silently:

```
PBUILDER.EXE "C:\Projects\TestProject.pbp" /exetype:2 /c /q /s  
/filelistupt
```

## 124.5 See also

### Console compiler →

Headless compiler for CI/CD pipelines

### Directives →

Override project settings without the GUI

### Exit codes →

Return codes after compilation

# 125. About Directive Files

## 125.1 Introduction to Directive Files

Paquet Builder introduces a specific file type called **"directive file"**, using the `.pbd` extension. These **text-based** files contain instructions for **Paquet Builder** to create or modify a package/installer.

### 125.1.1 Why Use Directive Files?

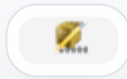
- **Automate Package Creation** — Useful for external applications needing to generate packages dynamically.
- **Modify Existing Projects** — Enables automated transformations of existing projects.
- **Shell Extension Support** — The Paquet Builder shell extension works with directive files.

**Recommended Format:** Save directive files in **UTF-8** (with or without BOM).

**Learn More:** [How to Open and Execute Directives](#).

#### Tip

You can **export a full directive file** from your current project with the **"Export Directives"** menu accessible from Paquet Builder's application button



. The export includes all project settings, components with their file lists, and optionally screen templates as separate HTML files. This is the easiest way to create a directive file that faithfully reproduces your project configuration.

## 125.2 How Directive Files Work

Directive files follow a format similar to **Windows INI configuration files**.

### 125.2.2 Example of a Directive File (`.pbd`)

```
; Paquet Builder Directive File
; Version 1.1

[DirectiveVariables]
MYVERSION=1.0

[General]
Title=My archive {$MYVERSION$}
ProjectTemplate=C:\MyWork\MyProject\baseproj.pbp
OutputDir=C:\MyWork\MyProject\output
OutputName=MyArchive.exe
Company=G.D.G. Software
EXEType=0 ; 32-bit package (0=32-bit, 1=64-bit, 2=ARM64)
IsWizard=1
StorePaths=1
Folder=C:\MyWork\Source\
SubFolders=1
Icon=C:\MyWork\MyProject\media6.ico

[Source]
0=C:\MyWork\MyProject\Source\*.
1={$PBDIRECTPATH$}\Data Source Files\*.xml
```

### 125.2.3 Mandatory Sections

- `[General]` — Configures the package settings (always required).
- `[Source]` — Lists files to include (unless `NoSourceSection=1` is set).

#### Caution

If `NoSourceSection` is **not** set to 1, files from a **project template** will be ignored.

## 125.3 [General] Section Parameters

This section defines the package configuration.

- Parameters can be specified in any order.
- **Boolean values:** Use 1 (true) or 0 (false).

## Note

If a field is missing, the default value from the **template project** is used.

| Parameter                    | Description   |
|------------------------------|---|
| <b>BinFolder</b>             | Defines the root folder for temporary output files.                                 |
| <b>CheckRights</b>           | Checks for administrative rights (1=Yes, 0=No).                                     |
| <b>Company</b>               | Defines the company name. <b>Mandatory if using the command-line compiler.</b>      |
| <b>CompressionCPUThreads</b> | Number of CPU threads used for compression (LZMA2 required).                        |
| <b>CompressionEngine</b>     | Defines the unpacking engine: 0=Auto, 1=Small engine, 2=Full-featured engine.       |
| <b>CompressionLevel</b>      | Compression level for <b>7-Zip</b> (0=No compression, 4=Ultra compression).         |
| <b>CompressionLZMA2</b>      | Enables <b>LZMA2 compression</b> instead of LZMA (1=Yes, 0=No).                     |
| <b>CompressionSolid</b>      | Enables <b>solid compression</b> (1=Yes, 0=No).                                     |
| <b>Copyright</b>             | Defines the <b>copyright</b> string in the installer.                               |
| <b>CryptFileNames</b>        | Encrypts filenames in the archive (1=Yes, 0=No).                                    |
| <b>DestFolder</b>            | Defines the destination folder for extracted files (do not use for Setup projects). |
| <b>DigitalSign</b>           | Enables <b>digital signing</b> (1=Yes, 0=No).                                       |
| <b>Email</b>                 | Specifies the company's <b>email address</b> .                                      |
| <b>EncryptFiles</b>          | Encrypts files during compression (1=Yes, 0=No).                                    |
| <b>EndPrompt</b>             | Custom message shown at the end of the package execution.                           |
| <b>EXEType</b>               | Defines <b>bitness</b> : 0=32-bit (default), 1=64-bit, 2=ARM64.                     |

| Parameter                       | Description  |
|---------------------------------|--|
| <b>ExistingArchive</b>          | Specifies the full path to an existing <b>7-Zip archive</b> instead of listing files manually.   |
| <b>FileDesc</b>                 | Sets the <b>file description</b> in version info.  |
| <b>ForcelIntegrity</b>          | Forces <b>integrity check</b> on signed executables (1=Yes, 0=No). Default: 0.   |
| <b>FileDescUninst</b>           | Sets the <b>uninstaller file description</b> in version info.  |
| <b>FileVerNum</b>               | Sets the <b>file version number</b> in version info.   |
| <b>Folder</b>                   | Defines the root folder for storing path information.  |
| <b>HideProgress</b>             | Hides the extraction progress window (1=Yes, 0=No).  |
| <b>Homepage</b>                 | Specifies the company's <b>website URL</b> .   |
| <b>Icon</b>                     | Path to the package <b>icon file</b> .   |
| <b>IconUninst</b>               | Path to the <b>uninstaller icon file</b> .   |
| <b>InstallMSI</b>               | Indicates whether the file in SetupExec is a <b>Windows Installer MSI file</b> (1=Yes, 0=No).  |
| <b>InstallScope</b>             | Defines the <b>installation scope mode</b> : -1=None (legacy), 0=Current User, 1=All Users, 2=User Choice, 3=Automatic. See <a href="#">Installation Scope</a> . |
| <b>IsWizard</b>                 | Enables the <b>wizard-style interface</b> (1=Wizard UI, 0=Standard UI).  |
| <b>KeepExistingProjectFiles</b> | If using ProjectTemplate, retains existing files (1=Yes, 0=No).  |
| <b>KeyProtect</b>               | Defines the <b>password</b> used for encrypting files.   |
| <b>LiveUpdate</b>               | Enables <b>live update</b> when using ProjectTemplate (1=Yes, 0=No).   |
| <b>MinimumElevatedRights</b>    | Required elevation level if RequestElevatedRights=1: 0=Require Admin, 1=Highest Available.   |

| Parameter                    | Description   |
|------------------------------|---|
| <b>MinimumUserRights</b>     | Required user rights if CheckRights=1: 0=Admin, 1=Power User.   |
| <b>MultiVolumeArc</b>        | Creates a <b>multi-volume archive</b> (1=Yes, 0=No).  |
| <b>NoSourceSection</b>       | Disables the [Source] section (1=Yes, 0=No).  |
| <b>OutputDir</b> (Required)  | Specifies the <b>output directory</b> for the generated package.  |
| <b>OutputName</b> (Required) | Defines the <b>package filename</b> (e.g., Setup.exe).  |
| <b>OverwriteReadOnly</b>     | Allows overwriting <b>read-only files</b> (1=Yes, 0=No).  |
| <b>PackMode</b> (Required)   | Specifies package type: 0=Standard, 1=Setup, 2=Archive.   |
| <b>PasswordToAsk</b>         | Defines the <b>password required</b> to extract or install the package.   |
| <b>PFXFile</b>               | Path to the <b>PFX file</b> for digital signing (ignored if SignCertifLocation=1).  |
| <b>PFXKey</b>                | Password for the <b>PFX file</b> (if required).   |
| <b>ProdVerNum</b>            | Defines the <b>product version number</b> in version info.  |
| <b>Project</b>               | Path to the <b>project file</b> (.pbpx) that Paquet Builder should create.  |
| <b>ProjectTemplate</b>       | Uses an <b>existing project template</b> and modifies it dynamically.   |
| <b>ReplaceStartup</b>        | Enables <b>file replacement at Windows startup</b> (1=Yes, 0=No).   |
| <b>RequestElevatedRights</b> | Requests <b>UAC elevation</b> (1=Yes, 0=No).  |
| <b>ScopeFailOnAdmin</b>      | Fail if <b>admin rights</b> cannot be obtained when scope requires them (1=Yes, 0=No). Default: 0. Only applies when InstallScope is set. |
| <b>ScopeRequestUAC</b>       | Request <b>UAC elevation</b> for All Users scope (1=Yes, 0=No). Default: 1. Only applies when InstallScope is set.                        |
| <b>SetupExec</b>             | Path to the <b>setup loader</b> for installer-type projects.  |

| Parameter                                 | Description   |
|---|---|
| <b>SetupParams</b>                        | Command-line parameters for the setup loader.   |
| <b>ShowCompSelDlg</b>                     | Displays the <b>Select Components</b> dialog (1=Yes, 0=No).   |
| <b>SignCertifLocation</b>                 | Defines where to get the <b>code signing certificate</b> or the signing method: 0=PFX, 1=Windows Store, 2=Thumbprint, 3= <b>SignTool Commands</b> , 4= <b>Azure Artifact Signing</b> , 5=JSig (see <b>JSigCmd section</b> below). |
| <b>SignDlls</b>                           | Also <b>sign embedded DLLs</b> (1=Yes, 0=No). Default: 1.   |
| <b>SignMethod</b>                         | Defines the <b>message digest method</b> : 0=SHA-256 & SHA-1, 1=SHA-256 only, 2=SHA-1 only. Ignored if SignCertifLocation > 2.  |
| <b>SignUninst</b>                         | Also <b>sign the uninstaller</b> executable (1=Yes, 0=No). Default: 1.  |
| <b>Silent</b>                             | Enables <b>silent mode</b> (1=Yes, 0=No).   |
| <b>SizeCheck</b>                          | Activates <b>size checking</b> (1=Yes, 0=No).   |
| <b>StartPrompt</b>                        | Defines a custom <b>startup prompt</b> .  |
| <b>StorePaths</b>                         | Defines whether <b>path information is stored</b> (1=Yes, 0=No).  |
| <b>SubFolders</b>                         | Includes <b>subfolders</b> when adding files (1=Yes, 0=No).   |
| <b>Title</b> (Required)                   | Sets the <b>title of the package</b> .  |
| <b>ArtifactSignEndpoint</b>               | Defines the <b>Azure Artifact Signing account endpoint</b> (must match your <b>Azure region</b> ). Old name TrustSignEndpoint is still accepted.  |
| <b>ArtifactSignCodeSigningAccountName</b> | Specifies the <b>Azure Artifact Signing account name</b> . Old name TrustSignCodeSigningAccountName is still accepted.  |
| <b>ArtifactSignCertificateProfileName</b> | Defines the <b>certificate profile name</b> for signing with Azure Artifact Signing. Old name TrustSignCertificateProfileName is still accepted.  |
| <b>UIModern</b>                           | Enables the <b>modern UI interface</b> : 0=Classic UI, 1=Modern UI. Overrides IsWizard if specified.  |

## 125.4 [Source] Section

This section lists **files to be included** in the package.

- Supports **wildcards** (e.g., \*.exe, \*.xml).
- Allows **absolute or relative paths** using directive variables.

### 125.4.4 Example Source Section

```
[Source]
0=C:\MyWork\Project\Source\*. *
1=C:\MyWork\Project\Readme.txt
2=C:\Windows\System\cabinet.dll
3={$PBDIRECTPATH$}\Data\*.xml
```

Use directive variables like {\$PBDIRECTPATH\$} to avoid hardcoded paths.

## 125.5 [Components] Section

Defines **optional components** that users can select during installation.

### 125.5.5 Example Component Structure

```

[Components]
0=Documentation
1=Pictures

[Documentation.Source]
0=C:\Project\Help\*.chm

[Documentation.Properties]
Description=Application Documentation
Title=Documentation
StoringPath=1

[Pictures.Source]
0=C:\Project\Media\*.png
1=C:\Project\Media\*.jpg

[Pictures.Properties]
Title=Pictures
Description=Sample pictures
Required=0
InstallDef=1
SourceLinked=1
SourceFolder=C:\Project\Media
IncSubFolders=1

```

- **Each component** requires a `[ComponentName.Source]` and `[ComponentName.Properties]` section.
- **The Main component always exists** and is defined in `[Main.Properties]`.

## 125.5.6 Component Properties Reference

| Property           | Description   |
|--------------------|---|
| <b>Title</b>       | Display name of the component in the selection dialog.  |
| <b>Description</b> | Description shown to the user during installation.  |
| <b>Variable</b>    | Runtime variable name for this component (e.g., <code>%VCOMPDOCUMENTATION%</code> ). Auto-generated if not specified. |

| Property                       | Description  |
|--------------------------------|--|
| <b>Required</b>                | Component is <b>required</b> and cannot be unchecked (1=Yes, 0=No). Default: 0.  |
| <b>InstallDef</b>              | Component is <b>checked by default</b> (1=Yes, 0=No). Default: 1.                |
| <b>NoDisplay</b>               | <b>Hide</b> the component from the selection dialog (1=Yes, 0=No). Default: 0.   |
| <b>SourceLinked</b>            | Link to a <b>source folder</b> for live updates (1=Yes, 0=No). Default: 0.       |
| <b>SourceFolder</b>            | Path to the <b>source folder</b> when SourceLinked=1.                            |
| <b>CustomDestinationFolder</b> | Override the <b>destination folder</b> for this component's files.               |
| <b>IncSubFolders</b>           | Include <b>subfolders</b> when scanning source folder (1=Yes, 0=No). Default: 0. |
| <b>ExcludeComponent</b>        | <b>Exclude</b> this component from the build (1=Yes, 0=No). Default: 0.          |
| <b>FileMaskInclude</b>         | Wildcard mask for <b>including</b> files (e.g., *.dll;*.exe).                    |
| <b>FileMaskExclude</b>         | Wildcard mask for <b>excluding</b> files (e.g., *.pdb;*.log).                    |
| <b>StoringPath</b>             | How file paths are stored: 0=Default, 1=Absolute, 2=No path (flat), 3=Relative.  |
| <b>RootFolder</b>              | Root folder for <b>relative path</b> storage (StoringPath=3).                    |

Learn more: [Component Management](#).

## 125.6 Directive Variables

Directive variables allow **dynamic path replacement** in directive files.

### 125.6.7 Built-in Variables

| Variable               | Meaning                                      |
|------------------------|--|
| { \$PBDIRECTPATH\$ }   | Path to the <b>directive file folder</b> .   |
| { \$PBTEMPLATEPATH\$ } | Path to the <b>template project folder</b> . |

| Variable        | Meaning                                      |
|-----------------|--|
| {\$PBDOCPATH\$} | Path to the <b>user's Documents folder</b> . |

## 125.6.8 Example Usage

```
[Source]
0={$PBDIRECTPATH$}\Source Files\*. *
1={$PBDIRECTPATH$}\Docs\*.pdf
```

This way, you can move the directive file without breaking file paths.

## 125.7 [Templates] Section — Screen Content

This section allows you to load **HTML content** into the installer's screens (Welcome, License Agreement, Readme, Final screen).

```
[Templates]
Welcome=C:\MyProject\screens\welcome.html
LicAg=C:\MyProject\screens\license.html
StartReadme=C:\MyProject\screens\readme_start.html
EndReadme=C:\MyProject\screens\readme_end.html
End=C:\MyProject\screens\final.html
```

| Template Key | Screen                                  |
|--------------|---|
| Welcome      | Welcome screen                          |
| LicAg        | License / End-User Agreement            |
| StartReadme  | Readme shown <b>before</b> installation |
| EndReadme    | Readme shown <b>after</b> installation  |
| End          | Final screen                            |

- Values can be **absolute paths** or **relative filenames** (looked up in the `Templates\` folder next to `PBuilder.exe`).
- Files can be **HTML** or legacy **RTF** (automatically converted to HTML).
- Template variables like `[ARCHIVETITLE]`, `[COMPANYNAME]`, `[COMPANYEMAIL]`, `[HOMEPAGE]`, `[COPYRIGHT]`, `[DESCRIPTION]`, `[PRODNUM]`, `[FILENUM]` are replaced with project values.

### Tip

When using “**Export Directives**”, checking “Export screen templates as HTML files” will automatically save the screen content as `.html` files alongside the `.pbd` file and reference them in the `[Templates]` section.

## 125.8 [SignToolCmd] Section — Custom SignTool Commands

This section allows **multiple digital signatures** to be applied sequentially. Only used when **SignCertifLocation** is set to 3.

Example:

```
[SignToolCmd]
0=sign /fd SHA256 /tr "http://timestamp.digicert.com" /td SHA256 "
  {$PBOUTPUTFILE$}"
1=sign /fd SHA1 /tr "http://timestamp.digicert.com" /td SHA1 "
  {$PBOUTPUTFILE$}"
```

- Each line represents one signing command.
- Commands are executed in order (e.g., first SHA-256, then SHA-1).

## 125.9 [JSignCmd] Section — JSign Commands

This section defines **JSign signing commands** when **SignCertifLocation** is set to 5.

**JSign** is a free, open-source Java-based code signing tool supporting PFX/PKCS#12, hardware tokens (ETOKEN, YUBIKEY), Azure Key Vault, AWS KMS, Google Cloud KMS, and more.

Example:

```
[JSignCmd]
0=sign --keystore "certificate.p12" --storepass "mypass" --tsurl
http://timestamp.sectigo.com --tsmode RFC3161 --digest-algorithm SHA-256 "
{$PBOUTPUTFILE$}"
```

### 125.9.9 Available Placeholders

| Placeholder          | Replaced with                                     |
|----------------------|---|
| {\$PBOUTPUTFILE\$}   | Full path to the <b>output file</b> being signed. |
| {\$PBOUTPUTFOLDER\$} | Path to the <b>output folder</b> .                |

#### Note

JSign requires **Java** (java.exe on PATH) and the **JSign JAR file** configured in Paquet Builder's Environment Options. You can install it via `choco install jsign`.

## 125.10 Define Custom Directive Variables

Create **your own variables** inside [DirectiveVariables].

```
[DirectiveVariables]
MYVERSION=1.0
OUTPUTNAME=Installer
```

Use them in any section:

```
[General]
Title=My archive {$MYVERSION$}
OutputName={$OUTPUTNAME$} {$MYVERSION$}.exe
```

## 125.11 Example: Simple Directive File Without Source

```
; Paquet Builder Directive File
[General]
Title=My archive 1
NoSourceSection=1
OutputName=Direct2.exe
OutputDir=C:\Build\Output
ProjectTemplate=C:\Build\Template.pbpX
LiveUpdate=1
```

Use `NoSourceSection=1` if your directive modifies an existing project without changing its file list.

## 125.12 Related Topics

### Execute Directives →

How to open and execute directive files.

### Working with Components →

Component management in Paquet Builder.

### MCP Server →

Manage projects programmatically via AI assistants like Claude Code, Cursor, or Windsurf.

### AI Assistant →

Configure projects interactively via natural language.

### Installation Scope →

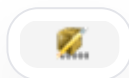
InstallScope directive parameter documentation.

### Signing with JSign →

JSignCmd section documentation.

## 126. How to execute directive files in Paquet Builder?

To execute a directive file without using the command line, click the



and select the **"Open Directive File"** menu option.

Alternatively, you can execute directive files via the **command line**. When launching Paquet Builder manually (e.g., using the "Run" command in the Windows Start menu), you can pass parameters to specify the directive file.

### Tip

Directive files can also be executed using the **console command-line compiler**.

Use the following command line to execute a directive file:

```
PBUILDER.EXE "c:\mywork\myproject\file.pbd"
```

This command opens Paquet Builder, reads the settings from the directive file, and prepares the project. You only need to press the Compile button to create the package.

Paquet Builder also supports the **same command-line switches** as for project files.

### 126.0.1 Examples

Open a directive file, compile the project without interaction, save the log to an HTML file, and exit:

```
PBUILDER.EXE "C:\My Documents\pbtests\directive1.pbd" /c /s /q /log:"C:\My Documents\pblog.html"
```

Open a directive file and set the **bitness to x64** via the command line:

```
PBUILDER.EXE "C:\My Documents\pbtests\instructions package 2.pbd"  
/exetype:1
```

You can also override settings defined in the directive file by adding command-line switches. For example, if the directive specifies a 32-bit build but you need 64-bit, simply add `/exetype:1`.

## 126.0.2 Executing directives with AI

In addition to the command line and the GUI menu, you can execute directive files using AI-powered tools:

- **MCP Server** — The `execute_directive` tool lets AI assistants like [Claude Code](#), [Cursor](#), or [Windsurf](#) run `.pbd` files programmatically. For example, you can ask: *“Execute the directive file at `C:|builds|release.pbd`”* and the AI will call the tool, compile the project, and report the result.
- **AI Assistant** — While the built-in AI Assistant does not execute directive files directly, it can accomplish the same results by modifying project settings and compiling — all through natural language conversation. For batch or scripted workflows, use directives; for interactive, conversational changes, use the AI Assistant.



### Tip

Combine directive files with the MCP server for powerful CI/CD automation: let the AI assistant generate directive files, then execute them headlessly via the MCP server or the [console compiler](#).

#### Builder Exit Codes →

Exit codes returned by Paquet Builder during compilation.

#### About Directive Files →

Directive file format and parameters.

#### Command-line Options →

All command-line switches.

#### MCP Server Setup →

Connect AI assistants to Paquet Builder.

#### MCP Server Tool Reference →

All 20+ tools including `execute_directive`.

#### AI Assistant Overview →

Built-in conversational project editing.

# 127. Console Command-line Compiler

---

The console command-line compiler is designed for advanced users who want to use Paquet Builder in automated workflows, such as process automation, distribution cycles, or scheduled tasks. This tool is a Windows console application with no graphical user interface (GUI).

Unlike the Paquet Builder GUI, the console compiler does not require a user logon, making it ideal for environments where interaction with the Windows desktop is unavailable. It is also portable, meaning it can be used simply by placing it in the same directory as `PBuilder.exe`.

## 127.1 Benefits of the Console Compiler

While the Paquet Builder GUI supports [command-line options and switches](#), the console compiler is better suited for headless environments, allowing tasks to be performed without requiring a logged-in user.

## 127.2 How to Obtain the Console Command-line Compiler

The console compiler is **only available to registered users** of the [Paquet Builder Commercial edition](#).

You can download it from "[My Account - Compiler](#)".

## 127.3 Setting Up the Console Command-line Compiler

- 1 Install Paquet Builder (GUI) on the target computer.
- 2 Download the console compiler's self-installing archive from "[My Account - Compiler](#)" and run it.
- 3 Copy your Paquet Builder license file into the same directory as the console compiler executable (`PBCMDCOMPILER.EXE`) to activate it.
- 4 The console compiler is now ready to use.

For detailed setup instructions, visit "[My Account - Compiler](#)".

## 127.4 Using the Console Command-line Compiler

### 127.4.1 Syntax

```
PBCMDCOMPILER [project filepath] [other parameters]
```

## 127.4.2 Parameters

The console compiler supports the same **command-line parameters and switches** as the Paquet Builder GUI, except `/c`, `/s`, and `/q`, which are unnecessary for the console compiler.

## 127.4.3 Exit Codes

The exit codes are identical to those used by the **Paquet Builder GUI**.

## 127.4.4 Examples

**Compile a package:**

```
PBCMDCOMPILER "c:\mywork\myproject\file.pbp"
```

**Change the **output directory**, compile the project, and save the log to an HTML file:**

```
PBCMDCOMPILER "C:\My Documents\pbttests\testdirect1.pbp" /log:"C:\my documents\pblog.html" /outfolder:"C:\my documents\pbttests\output"
```

## 127.5 Additional Notes

The console compiler also supports **directive files**, allowing even more flexibility for automation.

### Tip

For AI-driven automation, consider using the **MCP Server** instead. It exposes the same build engine via the Model Context Protocol, allowing AI assistants like Claude Code, Cursor, or Windsurf to open, modify, compile, and save projects programmatically.

## MCP Server



AI-driven headless project management.

## About Directive Files



Script-based automation.

## Command-line Options



All command-line switches.

## 128. Exit codes for Paquet Builder

This page describes possible exit codes returned by Paquet Builder when you compile packages using the [command line](#) or the [console compiler](#).

| Value | Meaning   |
|-------|---|
| 0     | No error (compilation successful).                        |
| 1     | Error in project settings.                                |
| 2     | Error while creating package scripts for compilation.     |
| 3     | Error while compiling scripts for the package stub.       |
| 4     | Error while linking the package stub.                     |
| 5     | Error while creating uninstaller scripts for compilation. |
| 6     | Error while compiling scripts for the uninstaller stub.   |
| 7     | Error while linking the uninstaller stub.                 |
| 8     | Error while compressing files.                            |
| 9     | Error while generating internal resources.                |
| 10    | Error while generating the uninstaller EXE.               |
| 11    | Error while combining the uninstaller EXE and resources.  |
| 12    | Error while combining the package EXE and resources.      |
| 13    | Error while combining the package EXE and the archive.    |
| 14    | Unexpected error.   |

### Tip

Exit code 0 indicates success. Any non-zero value means the build failed at the stage described above. Use these codes in your scripts to detect and handle build failures.

## Command-line Builder →

Launch Paquet Builder with command-line switches.

## Console Compiler →

Compile packages from a portable console tool.

## Package Installer Exit Codes →

Exit codes returned by packages at runtime.

## 129. Command-line switches for packages

Paquet Builder allows you to define custom command-line switches for your packages and uninstallers. These switches can be used to enhance the flexibility and functionality of your packages.

### Caution

Paquet Builder does not include pre-configured command-line switches. You must define them manually. However, [action templates](#) are available to help you set them up quickly.

### 129.1 Accessing the command line for packages and uninstallers

Paquet Builder provides several predefined [variables](#) that can be used in conjunction with [custom actions](#) to process command-line arguments.

| Variable name             | Description   |
|---------------------------|---|
| <code>%PARAMCOUNT%</code> | Number of command-line parameters passed to the package. Defines how many <code>%PARAMx%</code> variables are available.  |
| <code>%PARAM%</code>      | First command-line parameter passed to the package (if any); blank if none.   |
| <code>%PARAM2%</code>     | Second command-line parameter passed to the package (if any); blank if none.  |
| <code>%PARAM3%</code>     | Third command-line parameter passed to the package (if any); blank if none. Replace 3 with the desired parameter position (up to <code>%PARAMCOUNT%</code> ).     |
| <code>%PARAMx%</code>     | x-th parameter passed to the package (replace x with the position of the argument); blank if none. Valid values for x range from 2 to <code>%PARAMCOUNT%</code> . |
| <code>%PARAMS%</code>     | Entire command line passed to the package.  |

#### 129.1.1 Examples

Command:

```
MYPACK.EXE /s
```

Values:

- %PARAMCOUNT% = 1
- %PARAM% = /s
- Other %PARAMx% variables are blank.

Command:

```
MYPACK.EXE /LCID 1024 /d "C:\My Documents\Data"
```

Values:

- %PARAMCOUNT% = 4
- %PARAM% = /LCID
- %PARAM2% = 1024
- %PARAM3% = /d
- %PARAM4% = C:\My Documents\Data

### Caution

Arguments with spaces must be enclosed in **quotes**. Paquet Builder automatically removes these quotes when processing the command line.

## 129.2 Defining command-line switches

You can define custom switches using the **custom action** "Find one string within another". This allows you to detect and process specific switches included in the command line.

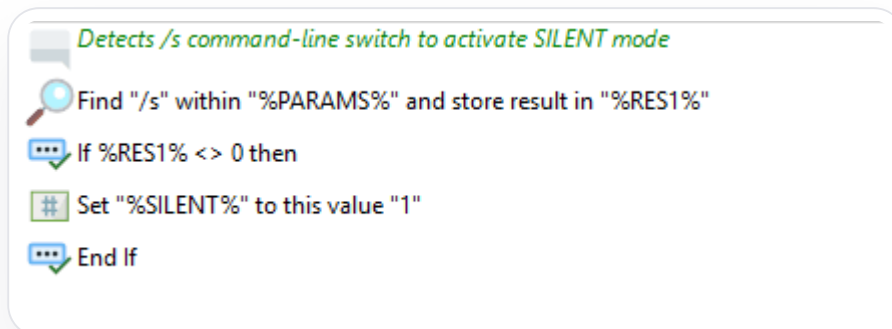
### 129.2.2 Action Templates

Paquet Builder provides **action templates** for common use cases. For example:

- **Enable Silent Mode from Package Command Line:** This template activates **silent mode**, suppressing all dialog boxes and messages when a specific switch is detected.

### 129.2.3 Example:

To detect and process a `/silent` switch, use the provided template or create a custom action to parse `%PARAMS%` for the switch.



#### Custom Actions →

Overview of custom actions in Paquet Builder.

#### Action Templates →

Pre-built action templates for common tasks.

#### Variables →

All available variables for packages and installers.

# 130. Exit code for packages and installers

## 130.1 Exit code meaning

The **exit code** is the value returned by a terminating package or installer to the invoker. This is particularly useful in batch processing or when executing commands from the command line.

Packages and installers created with Paquet Builder automatically set their exit codes based on the execution outcome (success or error). The table below lists the possible default values.



### Tip

You can **manually set the exit code** using **custom actions**. Assign a value to the **%EXITCODE%** global variable using the **Perform operation on a variable (varAssign)** custom action.

## 130.2 Default exit code values for Paquet Builder packages

| Value | Meaning   |
|-------|---|
| -3    | Corrupted data detected (error while decompressing data; initialization error from loader). |
| -2    | Cannot create runtime DLL (initialization error from loader).                               |
| -1    | Cannot create temporary folder (initialization error from loader).                          |
| 0     | No error (execution successful).  |
| 1     | Error while decompressing files.  |
| 2     | Aborted by the end user.  |
| 3     | Unexpected error - Fatal Windows error (rare).  |

### Builder Exit Codes



Exit codes returned by Paquet Builder during compilation.

### Custom Actions



Set custom exit codes using custom actions.

## Variable List



All available variables including %EXITCODE%.

# 131. About 7-Zip Format

**7z** is an open archive format developed by [Igor Pavlov](#) offering high compression ratios and modern features. Paquet Builder uses 7-Zip technology for all its compression and self-extracting operations.

## 131.1 Key Features



### High compression

Advanced algorithms like LZMA and LZMA2 deliver some of the best compression ratios available.



### AES-256 encryption

Strong encryption for secure data protection — both file content and filenames can be encrypted.



### Large file support

Files up to 16 exbibytes ( $2^{64}$  bytes).



### Solid compression

Treats multiple files as a single data block for better compression ratios.

## 131.2 Compression Methods

| Method       | Description  |
|--------------|--|
| <b>LZMA</b>  | Enhanced LZ77 algorithm — default method, high compression ratio |
| <b>LZMA2</b> | Improved LZMA with better multithreading support                 |
| <b>PPMD</b>  | PPMdH algorithm — excellent for text-heavy content               |
| <b>BCJ</b>   | Converter for 32-bit x86 executables                             |
| <b>BCJ2</b>  | Advanced converter for 32-bit x86 executables                    |

| Method         | Description                         |
|----------------|-------------------------------------|
| <b>BZip2</b>   | Burrows-Wheeler Transform algorithm |
| <b>Deflate</b> | Standard LZ77 + Huffman coding      |

### 131.3 Filters

| Filter       | Description                                    |
|--------------|--|
| <b>ARM</b>   | Converter for 32-bit ARM executables           |
| <b>ARM64</b> | Converter for ARM64 executables                |
| <b>PPC</b>   | Converter for PowerPC executables              |
| <b>SPARC</b> | Converter for SPARC executables                |
| <b>IA64</b>  | Converter for Itanium executables              |
| <b>Delta</b> | Converter for multimedia data (e.g. WAV files) |

### 131.4 LZMA Characteristics



#### Variable dictionary

Dictionary sizes up to 4 GB for maximum compression.



#### Fast decompression

10–20 MB/s on a 2 GHz CPU, with low memory requirements.



#### Small decoder

Decompression code is only ~5 KB — ideal for embedded applications.



#### Multithreading

Full support for multithreaded and hyper-threaded CPUs.

## Note

Paquet Builder Self-Extractors support **LZMA2, LZMA, and BCJ/BCJ2** methods. BZip2 and Deflate are not supported in self-extracting packages, as the other methods provide sufficient performance.

## 131.5 Technical Notes

The 7-Zip compression system in Paquet Builder is based on **7-Zip and LZMA SDK 26.0**.

Developers interested in the LZMA algorithm can access the **LZMA SDK** with C++ source code for both encoder and decoder.

## 131.6 Related Pages

### Compression Options

Configure compression level and 7-Zip parameters.

### File Extraction Settings

Choose the decompression engine for your packages.

### Multi-Volume Archive

Split large archives into smaller volumes.

# 132. Working with Microsoft Windows Installer Packages

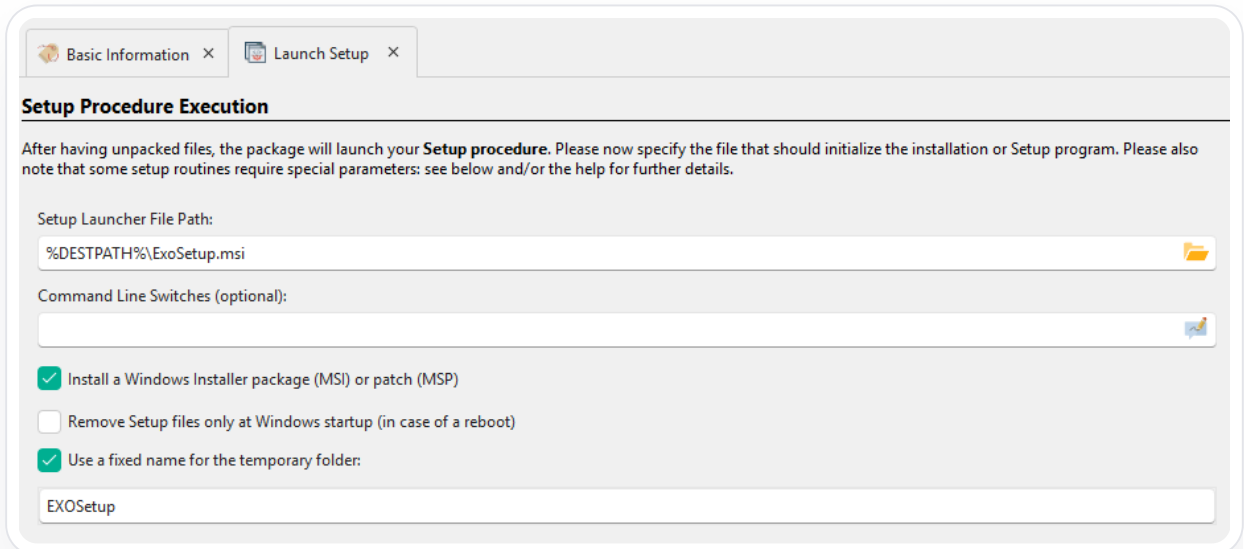
**Windows Installer** (MSI) is an operating system service for installing and uninstalling software. Paquet Builder lets you wrap MSI setups and MSP patches into a single self-extracting `.exe` — ideal for distribution over the internet, email, or removable media.

## Tip

Use Paquet Builder to create an MSI bootstrapper/loader, combine multiple MSI packages into one delivery, or add a branded wizard on top of an existing MSI setup.

## 132.1 Creating an MSI Setup Package

- 1 **Create a Package for Setup** — choose the Setup Routine type.
- 2 In the “Choose Setup Program” panel, select **“Install a Windows Installer package (MSI) or patch (MSP)”** and pick your file:



Basic Information × Launch Setup ×

### Setup Procedure Execution

After having unpacked files, the package will launch your **Setup procedure**. Please now specify the file that should initialize the installation or Setup program. Please also note that some setup routines require special parameters: see below and/or the help for further details.

Setup Launcher File Path:  
%DESPATH%\ExoSetup.msi

Command Line Switches (optional):

Install a Windows Installer package (MSI) or patch (MSP)  
 Remove Setup files only at Windows startup (in case of a reboot)  
 Use a fixed name for the temporary folder:

EXOSetup

- 3 Enable **“Use a fixed name for the temporary folder”** — Windows Installer remembers the source folder for repairs and updates, so a consistent name ensures it can locate files.
- 4 For multiple MSI setups or prerequisites, add **custom actions**.

## 132.2 Advanced MSI Actions

Administrative installations, product advertising, and other advanced operations are available through the [Install MSI Package custom action](#).

## 132.3 Compressing MSI Files with 7-Zip

You can significantly reduce MSI/CAB file sizes using Paquet Builder's LZMA2 and BCJ2 compression:

- 1 **Disable CAB compression** — set `CompressionLevel` to "none" in your MSI's Media settings.
- 2 **Keep CAB files external** — do not embed them inside the MSI.
- 3 **Add MSI + CAB files to Paquet Builder** — 7-Zip compression delivers much better ratios than the LZX used in Cabinet format.

## 132.4 Additional Resources

### Install MSI Custom Action →

Run MSI packages and patches from custom actions.

### Setup Execution →

Configure the setup launcher and options.

### Microsoft MSI Documentation →

Official Windows Installer reference.

## 133. About Some Setup Software

---

When creating a [Setup Routine package](#), the package extracts files to a temporary folder, launches the setup, and removes the temporary files when the setup finishes. Some third-party setup programs cause issues because they exit before the installation is actually complete.

### 133.1 The Problem

Certain setup programs use a small **bootstrap loader** that initializes temporary files, launches the real installation, then exits immediately — even though the installation is still running. Paquet Builder sees the loader exit and removes the temporary files prematurely.

### 133.2 Solutions

---

[InstallShield](#) setups use a loader that closes before the installation finishes.

**Fix:** Add the `/SMS` parameter to the [Command Line Options](#) field. This instructs the loader to wait until the installation routine completes before exiting.

---

For other setup programs (e.g. Visual Studio Installer, MSI bootstrappers) or setups that require a **system reboot**:

Enable **“Remove Setup files only at Windows startup (in case of a reboot)”** in the [Setup Execution](#) panel.

- 1 The package extracts files to a temporary folder.
- 2 The setup procedure is launched.
- 3 Instead of immediate cleanup, an internal uninstaller is created and scheduled to run at the next Windows startup.
- 4 At startup, the uninstaller silently removes the temporary files and folder.

#### Tip

The delayed cleanup option is also useful for setups requiring a reboot. It ensures temporary files remain available during the reboot cycle and are cleaned up automatically afterward — without affecting your installed application files.

### 133.3 Related Pages

## Setup Execution →

Configure the setup launcher and cleanup options.

## MSI Packages →

Wrap Microsoft Windows Installer setups.

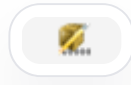
## Create a Project →

Choose the right package type for your scenario.

# 134. Environment Options

---

The **Environment Options** dialog configures global settings for Paquet Builder. Access it via the application button



then **Environment Options**.

## 134.1 Settings

| Option                                      | Description  |
|---|--|
| <b>Display compilation log immediately</b>  | Shows the log during the build process   |
| <b>Suppress live-update notifications</b>   | Hides messages when <b>folder-linked components</b> are auto-updated             |
| <b>Include sub-folders when adding</b>      | Includes subdirectories when adding folders or wildcards to the <b>file list</b> |
| <b>Remove obsolete files during updates</b> | Auto-removes missing files from lists during <b>live-update</b>                  |
| <b>Reduce bitmap color to 8bpp</b>          | Converts auto-generated bitmaps to 256 colors (smaller size, lower quality)      |
| <b>Hide "Getting Started" tip</b>           | Disables the tutorial shown at project creation                                  |
| <b>Hide build notifications</b>             | Suppresses Windows notifications after a build completes                         |
| <b>Hamburger menu navigation</b>            | Switches the sidebar from accordion to compact hamburger style                   |

Choose the Paquet Builder interface theme:

| Option                | Description  |
|-----------------------|--|
| <b>System-defined</b> | Follows your Windows light/dark preference (default) |
| <b>Light</b>          | Forces light theme                                   |
| <b>Dark</b>           | Forces dark theme                                    |

Pre-filled values for new projects:



## Company Name

Your name or company name — appears in **About Your Package**.



## E-Mail Address

Contact email for end users.



## Web Homepage

Your website URL.



## Default Copyright

Legal copyright text for new packages.

| Option                       | Description  |
|------------------------------|--|
| <b>Project Format</b>        | <b>Project Folder</b> (directory-based, easier to version-control) or <b>Single File</b> (.pbpx ZIP archive)   |
| <b>Content Viewer</b>        | <b>WebView2</b> (recommended, full HTML/CSS) or <b>Markdown viewer</b> (legacy, for Windows 7/8.1). Also configurable per-project in <a href="#">Developer Options</a> |
| <b>Default Language File</b> | <a href="#">Language file</a> for new projects   |
| <b>Default Icon File</b>     | <a href="#">Icon</a> for new projects  |

Configure defaults for [digital signing](#):

| Option                            | Description  |
|-----------------------------------|--|
| <b>SHA-1 TSA URL</b>              | Authenticode-compatible timestamp server   |
| <b>SHA-256 TSA URL</b>            | RFC 3161 timestamp server  |
| <b>SignTool.exe Path</b>          | Full path to <code>SignTool.exe</code> if not on system PATH                                     |
| <b>SignTool Default Command</b>   | Custom command-line template for SignTool  |
| <b>Artifact Signing dlib Path</b> | Path to the <a href="#">Azure Trusted Signing</a> dlib (auto-detected)                           |
| <b>JSigntool JAR Path</b>         | Path to <code>jsigntool.jar</code> for <a href="#">JSigntool signing</a> (requires Java on PATH) |

| Option                          | Description  |
|---------------------------------|--|
| <b>Working Folder</b>           | Temporary folder for compilation — useful for systems with limited temp space      |
| <b>Use default folders</b>      | Sets initial directories to Paquet Builder defaults for resource selection dialogs |
| <b>Set project path on load</b> | Saves the project path as default for related resource open dialogs                |

## 134.2 AI Assistant

The **AI Assistant** has its own settings dialog accessible from the AI Assistant window.



**Tip**

See [Setting Up the AI Assistant](#) for configuration details.

## 134.3 Related Pages

### Digital Signature →

Configure code signing for your packages.

### JSign Code Signing →

Sign packages with JSign (Java-based).

### AI Assistant Setup →

Configure the built-in AI assistant.

### Package Output →

Output settings including compilation log.

# 135. Technical Documentation

Technical notes about Paquet Builder and the packages it creates.

## 135.1 System Requirements

| Requirement         | Value  |
|---------------------|--|
| <b>RAM</b>          | 256 MB minimum (1 GB recommended for <b>high compression</b> ) |
| <b>OS</b>           | Windows 11, 10, 8.1, 8, 7 and Server editions                  |
| <b>Architecture</b> | 32-bit, x64, and ARM64   |

| Requirement         | Value   |
|---------------------|---|
| <b>RAM</b>          | 128 MB or higher                              |
| <b>OS</b>           | Windows 11, 10, 8.1, 8, 7 and Server editions |
| <b>Architecture</b> | 32-bit, x64, and ARM64                        |

## 135.2 Package Size Limit



### Self-Extracting packages

Up to **4 GB** — this is the EXE file size limit on Windows.



### Archive files

Non-SFX `.7z` archives can exceed 4 GB.



### Multi-volume

Use **multi-volume archives** to bypass the 4 GB limit for self-extracting packages.

## 135.3 AI and Automation

Paquet Builder 2026 includes advanced AI and automation capabilities:

### AI Assistant →

Built-in chatbot for configuring projects through natural language.

### MCP Server →

Headless automation with AI tools like Claude Code, Cursor, or Windsurf.

### Console Compiler →

Headless batch compilation from the command line.

### Directive Files →

Plain-text INI files for scripted project creation and builds.

## 136. Automatic Updates

Paquet Builder includes a built-in **auto-update system** that checks for new versions and additional packages directly within the application — no separate update tool required.

### 136.1 Automatic Update Checks

On startup, Paquet Builder can automatically check for available updates in the background:



#### Once per day

Runs at most once per day to avoid unnecessary network activity.



#### Non-blocking

Uses a background thread with a short timeout — never slows down application startup.



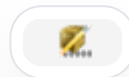
#### Opt-in

Paquet Builder asks on the second launch whether you want to enable automatic checks.

### 136.2 Manual Update Check

1

Click the Paquet Builder application button



2

Select **"Help & Web Resources"**, then **"Check for Updates"**

### 136.3 Update Dialog

When an update is found, the dialog shows **release notes** and three options:

| Option                   | Description  |
|--------------------------|--|
| <b>Install Update</b>    | Downloads and launches the installer               |
| <b>Skip This Version</b> | Suppresses notifications for this specific version |

| Option                 | Description   |
|------------------------|---|
| <b>Remind Me Later</b> | Dismisses the dialog — you'll be notified again on the next check |

## 136.4 Additional Packages

The update system can also offer **additional packages** — standalone installers for components not included in the main distribution (e.g. documentation, language packs, or supplementary tools).

## 136.5 Security

### Note

Downloaded update files are verified using **Authenticode signature verification** before execution. If the digital signature is invalid or missing, the download is rejected and the temporary file is deleted.

## 136.6 Settings

Update preferences are stored locally:

- Whether automatic checks are enabled
- The date of the last check
- The version you chose to skip (if any)

### Tip

To disable automatic notifications, uncheck the option in [Environment Options](#). You can still check manually at any time.

## 136.7 Related Pages

### Environment Options

Configure global preferences including update settings.

### Digital Signature

Learn about Authenticode verification used for update security.