UNIVERSAL APP INSTALLER - DOCUMENTATION & CONFIGURATION GUIDE

Table of Contents

Overview

The Universal App Installer is a flexible, parameter-driven PowerShell script designed to install most standard Windows applications without requiring custom installation scripts. It integrates seamlessly with the orchestration framework and provides comprehensive logging, error handling, and validation.

When to Use Universal Installer vs Custom Scripts

✅ **Use Universal Installer For:**

- Simple applications with standard silent install switches
- Apps that only need file or registry detection
- Apps without complex pre/post configuration
- Standard MSI, EXE, MSIX, or APPX installers
- **Approximately 80% of your application portfolio**

❌ **Use Custom Scripts For:**

- Microsoft Office (complex XML configuration)
- SQL Server (multi-component installation)

- Adobe Creative Cloud (licensing complexity)
- Apps requiring enterprise policy configuration (Chrome, Firefox)
- Apps with complex pre/post installation steps
- **Approximately 20% of your application portfolio**

Features

Supported Features

- ✅ **Multiple Installer Types**: MSI, EXE, MSIX, APPX
- ✅ **Auto-Detection**: Automatically detects installer type from extension
- ✅ **Multiple Detection Methods**: Registry, File, AppX, Package, Custom
- ✅ **Version Comparison**: Ensures minimum version requirements
- ✅ **Automatic Installer Discovery**: Searches multiple paths
- ✅ **Pre/Post Install Scripts**: Execute custom logic when needed
- ✅ **Comprehensive Logging**: Detailed logs for troubleshooting
- ✅ **Timeout Protection**: Prevents hung installations
- ✅ **Exit Code Standards**: Integrates with orchestration reporting
- ✅ **Validation**: Confirms successful installation

Exit Codes

The Universal App Installer returns standard exit codes for orchestration integration:

Exit Code	Meaning	Orchestration Action
0	Success - App installed	Continue to next task
1	General failure	Retry or fail task
2	Installer not found	Fail task (file missing)
3	Detection method failed	Fail task (config error)
4	Installation failed	Retry or fail task
5	Validation failed	Retry or fail task
10	Already installed (success)	Continue to next task

Configuration Examples

Example 1: Simple MSI Application (7-Zip)

```powershell
@{
    TaskID = "APP-010"
    TaskName = "Install 7-Zip"
    ScriptPath = "Scripts\Universal-AppInstaller.ps1"
    Enabled = $true
    Timeout = 600
    RunAs = "SYSTEM"
    RequiresReboot = $false
    AllowRetry = $true
    Critical = $false
    Description = "Installs 7-Zip file archiver"
    Parameters = @{
        AppName = "7-Zip"
        InstallerFileName = "7z2408-x64.msi"
        InstallerType = "MSI"
        InstallArguments = "/quiet /norestart"
        DetectionMethod = "Registry"
        DetectionPath = "HKLM:\SOFTWARE\7-Zip"
        DetectionValue = "Path"
    }
}
```

Example 2: EXE with File Detection (Notepad++)

```powershell
@{
    TaskID = "APP-011"
    TaskName = "Install Notepad++"
    ScriptPath = "Scripts\Universal-AppInstaller.ps1"
    Enabled = $true
    Timeout = 600
    RunAs = "SYSTEM"
    RequiresReboot = $false
    AllowRetry = $true
    Critical = $false
    Description = "Installs Notepad++ text editor"
    Parameters = @{
        AppName = "Notepad++"
        InstallerFileName = "npp.8.6.9.Installer.x64.exe"
        InstallerType = "EXE"
        InstallArguments = "/S"
        DetectionMethod = "File"
        DetectionPath = "C:\Program Files\Notepad++\notepad++.exe"
    }
}
```

Example 3: Version-Specific Installation (VLC)

```powershell
```

```
@{
    TaskID = "APP-012"
    TaskName = "Install VLC Media Player"
    ScriptPath = "Scripts\Universal-AppInstaller.ps1"
    Enabled = $true
    Timeout = 600
    RunAs = "SYSTEM"
    RequiresReboot = $false
    AllowRetry = $true
    Critical = $false
    Description = "Installs VLC Media Player"
    Parameters = @{
        AppName = "VLC Media Player"
        InstallerFileName = "vlc-3.0.20-win64.exe"
        InstallerType = "EXE"
        InstallArguments = "/L=1033 /S"
        DetectionMethod = "File"
        DetectionPath = "C:\Program Files\VideoLAN\VLC\vlc.exe"
        RequiredVersion = "3.0.20.0"
    }
}
```

Example 4: Adobe Reader with Registry Detection

```powershell
```

```
@{
    TaskID = "APP-013"
    TaskName = "Install Adobe Acrobat Reader"
    ScriptPath = "Scripts\Universal-AppInstaller.ps1"
    Enabled = $true
    Timeout = 900
    RunAs = "SYSTEM"
    RequiresReboot = $false
    AllowRetry = $true
    Critical = $false
    Description = "Installs Adobe Acrobat Reader DC"
    Parameters = @{
        AppName = "Adobe Acrobat Reader DC"
        InstallerFileName = "AcroRdrDC2400221005_en_US.exe"
        InstallerType = "EXE"
        InstallArguments = "/sAll /rs /msi EULA_ACCEPT=YES"
        DetectionMethod = "Registry"
        DetectionPath = "HKLM:\SOFTWARE\WOW6432Node\Adobe\Acrobat Reader\DC\Installer"
        DetectionValue = "Path"
    }
}
```

Example 5: PuTTY with Multiple Detection Paths

```powershell
```

```
@{
    TaskID = "APP-014"
    TaskName = "Install PuTTY"
    ScriptPath = "Scripts\Universal-AppInstaller.ps1"
    Enabled = $true
    Timeout = 300
    RunAs = "SYSTEM"
    RequiresReboot = $false
    AllowRetry = $true
    Critical = $false
    Description = "Installs PuTTY SSH client"
    Parameters = @{
        AppName = "PuTTY"
        InstallerFileName = "putty-64bit-0.81-installer.msi"
        InstallerType = "MSI"
        InstallArguments = "/quiet /norestart"
        DetectionMethod = "File"
        DetectionPath = "C:\Program Files\PuTTY\putty.exe"
    }
}
```

Example 6: WinSCP with Package Detection

powershell

```
@{
    TaskID = "APP-015"
    TaskName = "Install WinSCP"
    ScriptPath = "Scripts\Universal-AppInstaller.ps1"
    Enabled = $true
    Timeout = 600
    RunAs = "SYSTEM"
    RequiresReboot = $false
    AllowRetry = $true
    Critical = $false
    Description = "Installs WinSCP FTP/SFTP client"
    Parameters = @{
        AppName = "WinSCP"
        InstallerFileName = "WinSCP-6.3.5-Setup.exe"
        InstallerType = "EXE"
        InstallArguments = "/VERYSILENT /SUPPRESSMSGBOXES /NORESTART /SP-"
        DetectionMethod = "Package"
        DetectionPath = "WinSCP*"
    }
}
```

Example 7: Modern MSIX App (Paint.NET from Microsoft Store)

```
powershell
```

```
@{
    TaskID = "APP-016"
    TaskName = "Install Paint.NET"
    ScriptPath = "Scripts\Universal-AppInstaller.ps1"
    Enabled = $true
    Timeout = 600
    RunAs = "SYSTEM"
    RequiresReboot = $false
    AllowRetry = $true
    Critical = $false
    Description = "Installs Paint.NET image editor"
    Parameters = @{
        AppName = "Paint.NET"
        InstallerFileName = "paint.net.msix"
        InstallerType = "MSIX"
        InstallArguments = ""
        DetectionMethod = "AppX"
        DetectionPath = "dotPDN.LLC.Paint.NET"
    }
}
```

Example 8: TreeSize with Custom Post-Install Script

```
powershell
```

```
@{
    TaskID = "APP-017"
    TaskName = "Install TreeSize Free"
    ScriptPath = "Scripts\Universal-AppInstaller.ps1"
    Enabled = $true
    Timeout = 600
    RunAs = "SYSTEM"
    RequiresReboot = $false
    AllowRetry = $true
    Critical = $false
    Description = "Installs TreeSize disk space analyzer"
    Parameters = @{
        AppName = "TreeSize Free"
        InstallerFileName = "TreeSizeFree-Portable.exe"
        InstallerType = "EXE"
        InstallArguments = "/VERYSILENT /SUPPRESSMSGBOXES /NORESTART /SP-"
        DetectionMethod = "File"
        DetectionPath = "C:\Program Files\JAM Software\TreeSize Free\TreeSizeFree.exe"
        PostInstallScript = {
            # Create desktop shortcut for all users
            $ShortcutPath = "C:\Users\Public\Desktop\TreeSize Free.lnk"
            $WScriptShell = New-Object -ComObject WScript.Shell
            $Shortcut = $WScriptShell.CreateShortcut($ShortcutPath)
            $Shortcut.TargetPath = "C:\Program Files\JAM Software\TreeSize Free\TreeSizeFree.exe"
            $Shortcut.Save()
        }
    }
}
```

Example 9: FileZilla with Version Check

```
powershell
```

```
@{
    TaskID = "APP-018"
    TaskName = "Install FileZilla"
    ScriptPath = "Scripts\Universal-AppInstaller.ps1"
    Enabled = $true
    Timeout = 600
    RunAs = "SYSTEM"
    RequiresReboot = $false
    AllowRetry = $true
    Critical = $false
    Description = "Installs FileZilla FTP client"
    Parameters = @{
        AppName = "FileZilla Client"
        InstallerFileName = "FileZilla_3.67.0_win64-setup.exe"
        InstallerType = "EXE"
        InstallArguments = "/S"
        DetectionMethod = "Registry"
        DetectionPath = "HKLM:\SOFTWARE\FileZilla Client"
        DetectionValue = "Version"
        RequiredVersion = "3.67.0"
    }
}
```

Example 10: Git with Pre-Install Cleanup

```powershell
```

```
@{
    TaskID = "APP-019"
    TaskName = "Install Git for Windows"
    ScriptPath = "Scripts\Universal-AppInstaller.ps1"
    Enabled = $true
    Timeout = 900
    RunAs = "SYSTEM"
    RequiresReboot = $false
    AllowRetry = $true
    Critical = $false
    Description = "Installs Git version control system"
    Parameters = @{
        AppName = "Git for Windows"
        InstallerFileName = "Git-2.43.0-64-bit.exe"
        InstallerType = "EXE"
        InstallArguments = "/VERYSILENT /SUPPRESSMSGBOXES /NORESTART /SP- /COMPONENTS=`"icons,ext\reg\s
        DetectionMethod = "File"
        DetectionPath = "C:\Program Files\Git\bin\git.exe"
        PreInstallScript = {
            # Remove old Git installations
            $OldGit = "C:\Program Files (x86)\Git"
            if (Test-Path $OldGit) {
                Remove-Item -Path $OldGit -Recurse -Force -ErrorAction SilentlyContinue
            }
        }
    }
}
```

Detection Methods

1. Registry Detection

Best For: Applications that write to registry during installation

Parameters Required:

- `DetectionPath`: Registry path (e.g., `HKLM:\SOFTWARE\AppName`)
- `DetectionValue`: (Optional) Specific registry value to check

Example:

```powershell
DetectionMethod = "Registry"
DetectionPath = "HKLM:\SOFTWARE\7-Zip"
DetectionValue = "Path"
```

Common Registry Paths:

- `HKLM:\SOFTWARE\AppName`
- `HKLM:\SOFTWARE\WOW6432Node\AppName` (32-bit apps on 64-bit Windows)
- `HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\{GUID}`

2. File Detection

Best For: Applications with predictable installation paths

Parameters Required:

- `DetectionPath`: Full path to executable or key file

Example:

```powershell
DetectionMethod = "File"
DetectionPath = "C:\Program Files\Notepad++\notepad++.exe"
```

Tips:

- Use main executable, not DLLs
- Include full path with drive letter
- Can include version checking with `RequiredVersion` parameter

3. AppX Detection

Best For: Modern Windows Store / MSIX applications

Parameters Required:

- `DetectionPath`: Package family name (e.g., `Microsoft.WindowsCalculator`)

Example:

```powershell
```

```
DetectionMethod = "AppX"
DetectionPath = "Microsoft.WindowsCalculator"
```

Finding Package Names:

```powershell
Get-AppxPackage | Where-Object {$_.Name -like "*Calculator*"}
```

4. Package Detection

Best For: Applications registered with Windows Package Manager

Parameters Required:

- `DetectionPath`: Package name or wildcard pattern

Example:

```
DetectionMethod = "Package"
DetectionPath = "WinSCP*"
```

Finding Package Names:

```powershell
Get-Package -Name "*WinSCP*"
```

5. Custom Detection

Best For: Complex detection logic

Parameters Required:

- `DetectionScript`: PowerShell script block returning $true/$false

Example:

```powershell
```

```
DetectionMethod = "Custom"
DetectionScript = {
    # Custom logic here
    $Installed = Test-Path "C:\Program Files\MyApp\app.exe"
    $ConfigExists = Test-Path "C:\ProgramData\MyApp\config.xml"
    return ($Installed -and $ConfigExists)
}
```

6. None

Best For: Always reinstall (force install)

Example:

```
powershell

DetectionMethod = "None"
```

Common Silent Install Switches

MSI Installers

```
powershell

# Standard MSI
InstallArguments = "/quiet /norestart"

# With verbose logging
InstallArguments = "/quiet /norestart /l*v `"C:\Logs\install.log`""

# Suppress reboot
InstallArguments = "/qn REBOOT=ReallySuppress"
```

EXE Installers

NSIS Installers (Nullsoft)

```
powershell

InstallArguments = "/S"
```

Examples: Notepad++, PuTTY, WinSCP, FileZilla

Inno Setup

```
powershell

InstallArguments = "/VERYSILENT /SUPPRESSMSGBOXES /NORESTART /SP-"
```

Examples: Git, Paint.NET, TreeSize

InstallShield

```
powershell

InstallArguments = "/s /v`"/qn`""
```

Wise Installer

```
powershell

InstallArguments = "/s"
```

Advanced Installer

```
powershell

InstallArguments = "/q /norestart"
```

Application-Specific Examples

Adobe Acrobat Reader

```
powershell

InstallArguments = "/sAll /rs /msi EULA_ACCEPT=YES"
```

VLC Media Player

```
powershell

InstallArguments = "/L=1033 /S"
```

7-Zip

```powershell
InstallArguments = "/S"  # EXE
InstallArguments = "/quiet /norestart"  # MSI
```

Java JRE

```powershell
InstallArguments = "/s INSTALL_SILENT=1 STATIC=0 AUTO_UPDATE=0 WEB_ANALYTICS=0"
```

Chrome (Standalone)

```powershell
InstallArguments = "/silent /install"
```

Troubleshooting

Problem: Installer Not Found

Symptoms:

- Exit code 2
- Log shows "Installer not found in any search path"

Solutions:

1. **Verify Installer Path:**

```powershell
# Check default search paths
Test-Path ".\Installers\Apps\yourinstaller.exe"
Test-Path "C:\Deploy\Apps\yourinstaller.exe"
```

2. **Add Custom Search Path:**
   - Edit script `$InstallerSearchPaths` array
   - Add network path or local directory

3. **Copy Installer to Expected Location:**

```powershell
Copy-Item "\\server\share\installer.exe" -Destination "C:\Deploy\Apps\"
```

Problem: Detection Failing

Symptoms:

- App installs but validation fails
- Exit code 5
- Log shows "Validation failed"

Solutions:

1. **Verify Detection Path:**
   - Check actual install location
   - Verify registry key exists
   - Check for 32-bit vs 64-bit paths
2. **Test Detection Manually:**

```powershell
# Registry
Test-Path "HKLM:\SOFTWARE\7-Zip"
Get-ItemProperty "HKLM:\SOFTWARE\7-Zip"

# File
Test-Path "C:\Program Files\Notepad++\notepad++.exe"

# Package
Get-Package -Name "WinSCP*"
```

3. **Use Alternative Detection Method:**
   - If registry fails, try file detection
   - If file fails, try package detection

Problem: Installation Hangs

Symptoms:

- Script times out
- Process never completes

Solutions:

1. **Increase Timeout:**

```powershell
Timeout = 3600  # Increase to 1 hour
```

2. **Verify Silent Switches:**
   - Test installer manually with switches
   - Some installers need specific switches to be truly silent
3. **Check for User Prompts:**
   - Installer may be waiting for user input
   - Add more aggressive silent switches

Problem: Exit Code 1 (General Failure)

Symptoms:

- Installation fails with generic error
- Exit code 1

Solutions:

1. **Review Logs:**
   - Check `C:\ProgramData\OrchestrationLogs\Apps\Install-AppName_*.log`
   - Look for specific error messages
2. **Test Installer Manually:**

```powershell
# Run installer manually to see what happens
Start-Process "path\to\installer.exe" -ArgumentList "/S" -Wait
```

3. **Check Prerequisites:**
   - Ensure .NET Framework installed

- Verify sufficient disk space
- Check Windows version compatibility

Problem: App Installs But Doesn't Work

Symptoms:

- Exit code 0 (success)
- App installed but crashes or doesn't function

Solutions:

1. **Add Post-Install Configuration:**

```
powershell

  PostInstallScript = {
    # Configure app settings
    $ConfigPath = "C:\ProgramData\MyApp\config.xml"
    # ... configure settings ...
  }
```

2. **Check Dependencies:**
   - Install required runtimes (.NET, Visual C++)
   - Ensure services are running
3. **Verify Permissions:**
   - Check if app requires admin rights
   - Verify file/folder permissions

Best Practices

1. Always Test Installation Manually First

Before adding to orchestration:

```
powershell
```

```
# Test the installer manually
& "C:\Deploy\Apps\installer.exe" /S

# Verify detection works
Test-Path "C:\Program Files\App\app.exe"
```

2. Use Consistent Naming

```
powershell

# Good naming conventions
InstallerFileName = "7z2408-x64.msi"            # Clear version in name
InstallerFileName = "npp.8.6.9.Installer.x64.exe" # Version and architecture

# Poor naming
InstallerFileName = "setup.exe"                 # Too generic
InstallerFileName = "installer_final_v2_NEW.msi"  # Unclear version
```

3. Prefer MSI Over EXE When Available

MSI installers:

- Have standardized switches
- Better logging
- More reliable silent installation
- Easier to troubleshoot

4. Document Custom Switches

```
powershell

@{
    TaskName = "Install App"
    Description = "Installs App with custom switch for no desktop icon"
    Parameters = @{
        InstallArguments = "/S /NOICONS"  # /NOICONS prevents desktop shortcut
    }
}
```

5. Use Version Checking for Critical Apps

```
powershell
```

```
Parameters = @{
    RequiredVersion = "8.6.9"  # Ensures minimum version
}
```

6. Group Related Apps in Phase 4

```powershell
$Phase4_Applications = @{
    Tasks = @(
        @{ TaskID = "APP-010"; TaskName = "Install 7-Zip" }      # Utilities
        @{ TaskID = "APP-011"; TaskName = "Install Notepad++" }  # Editors
        @{ TaskID = "APP-012"; TaskName = "Install VLC" }        # Media
        # ... more apps ...
    )
}
```

7. Set Appropriate Timeouts

```powershell
Timeout = 300   # 5 minutes - Small utility
Timeout = 600   # 10 minutes - Standard app
Timeout = 1800  # 30 minutes - Large app
Timeout = 3600  # 60 minutes - Very large app (Office, Adobe CC)
```

8. Enable Retry for Network Installers

```powershell
@{
    AllowRetry = $true
    Critical = $false  # Don't stop orchestration if fails
}
```

9. Use Pre-Install Scripts for Cleanup

```powershell
```

```
PreInstallScript = {
    # Remove old version first
    $OldVersion = "C:\Program Files (x86)\OldApp"
    if (Test-Path $OldVersion) {
        Remove-Item -Path $OldVersion -Recurse -Force
    }
}
```

10. Validate After Installation

```
powershell

Parameters = @{
    ValidateInstall = $true  # Always enabled by default
}
```

Quick Reference: Task Template

Copy this template for new apps:

```
powershell
```

```
@{
    TaskID = "APP-0XX"                          # Unique ID
    TaskName = "Install [Application Name]"     # Descriptive name
    ScriptPath = "Scripts\Universal-AppInstaller.ps1"  # Universal installer
    Enabled = $true                             # Enable task
    Timeout = 600                               # 10 minutes
    RunAs = "SYSTEM"                            # Run as SYSTEM
    RequiresReboot = $false                     # Usually false
    AllowRetry = $true                          # Enable retry
    Critical = $false                           # Usually not critical
    Description = "[Brief description of application]"  # Description
    Parameters = @{
        AppName = "[Application Display Name]"          # Full app name
        InstallerFileName = "[installer-filename.exe/msi]"  # Installer file
        InstallerType = "AUTO"                  # AUTO, MSI, EXE, MSIX
        InstallArguments = "[/S or /quiet]"     # Silent switches
        DetectionMethod = "[Registry/File/Package]"  # How to detect
        DetectionPath = "[path or registry key]"     # Where to check
        DetectionValue = ""                     # Optional value
        RequiredVersion = ""                    # Optional version
    }
}
```

Support & Additional Resources

Finding Silent Install Switches

- Silent Install HQ
- WPKG Wiki
- Vendor documentation
- Trial and error with `/? /h --help` switches

Testing Tools

```
powershell
```

```
# Test MSI install
msiexec /i "installer.msi" /qn /l*v "C:\Temp\install.log"

# Monitor registry changes
Process Monitor (procmon.exe) from Sysinternals

# Check what files are written
Process Explorer (procexp.exe) from Sysinternals
```

End of Documentation