Package Development Manager 2.0 - Complete Technical Documentation

GPS Packaging Team - Optum

July 23, 2025

Table of Contents

# Package Development Manager 2.0 - Technical Documentation

## Table of Contents

1. [Application Overview](#application-overview)
2. [System Architecture](#system-architecture)
3. [Core Features](#core-features)
4. [Technical Implementation](#technical-implementation)
5. [Code Structure](#code-structure)
6. [Installation & Deployment](#installation--deployment)
7. [User Guide](#user-guide)
8. [Testing Procedures](#testing-procedures)
9. [Troubleshooting](#troubleshooting)
10. [Version History](#version-history)

## Application Overview

### Purpose

Package Development Manager 2.0 is a comprehensive Windows application designed for automated software package testing, installation, and validation. It provides enterprise-level package management capabilities with detailed reporting and logging.

### Key Capabilities

* **Multi-Source Package Support**: Network archives, completed packages, and local installers
* **Automated Installation Testing**: MSI, MSI+MST, EXE, and PSADTK package types
* **Comprehensive Scanning**: Before/after system state comparison
* **Professional Reporting**: HTML reports with detailed change analysis
* **Advanced Logging**: UHG logs integration and management
* **Validation Tools**: Post-installation verification and TSV generation

### Target Environment

* **Platform**: Windows 10/11, Windows Server 2016+
* **Architecture**: x64 systems
* **Permissions**: Administrator privileges required
* **Dependencies**: PowerShell 5.1+, .NET Framework 4.7.2+

## System Architecture

### Application Structure

Package Development Manager 2.0  
├── Core Engine (PowerShell)  
├── GUI Framework (Windows Forms)  
├── Package Management System  
├── Scanning & Comparison Engine  
├── Reporting System  
└── Integration Components

### Data Flow

1. **Package Selection** → Network/Local source identification
2. **Package Analysis** → Type detection and command building
3. **System Scanning** → Before-state capture
4. **Installation Execution** → Task Scheduler integration
5. **Post-Installation Scanning** → After-state capture
6. **Difference Analysis** → Change detection and categorization
7. **Report Generation** → HTML and TSV output
8. **Validation** → Copy-logs and verification

### File System Integration

* **Working Directory**: C:\temp\AutoTestScan\
* **Package Cache**: C:\temp\<Application>\<Version>\<DRMBuild>\
* **Log Storage**: C:\Windows\UHGLOGS\
* **Report Output**: Package-specific directories
* **Tool Output**: C:\temp\AutoTest\Tool\_Output\

## Core Features

### 1. Package Source Management

#### Network Package Archives

* **Package Archive**: Development/staging packages
* **Completed Packages**: Production-ready packages
* **Dynamic Discovery**: Vendor → Application → Version → Deployment → DRM Build

#### Local Package Support

* **File Browser Integration**: MSI/EXE file selection
* **Automatic Type Detection**: Package format identification
* **Parameter Configuration**: Custom installation arguments

### 2. Installation Management

#### Supported Package Types

* **MSI Only**: Standard Windows Installer packages
* **MSI + MST**: MSI with transform files
* **EXE Files**: Executable installers with custom parameters
* **PSADTK Wrapper**: PowerShell App Deployment Toolkit

#### Installation Methods

* **Task Scheduler Integration**: High-privilege execution
* **Command Line Generation**: Dynamic parameter building
* **Logging Integration**: UHG logs creation and management

### 3. System Scanning Engine

#### Scan Targets

* **Program Files**: C:\Program Files and C:\Program Files (x86)
* **Shortcuts**: Desktop and Start Menu locations
* **Registry**: Uninstall keys (32-bit and 64-bit)
* **UHG Logs**: Recursive log file scanning

#### Difference Detection

* **File System Changes**: New, removed, and modified files
* **Registry Changes**: Added/removed uninstall entries
* **Shortcut Changes**: Created/deleted shortcuts
* **Log File Changes**: New, removed, and modified log files

### 4. Reporting System

#### HTML Reports

* **Professional Styling**: Modern CSS with Optum branding
* **Comprehensive Sections**: Package info, system changes, timeline
* **Status Indicators**: Operation-specific labels (Added/Removed)
* **Interactive Elements**: Clickable sections and navigation

#### TSV Generation

* **Registry Data Export**: Uninstall key information
* **System Information**: OS, architecture, processor details
* **Standardized Format**: Tab-separated values for analysis

## Technical Implementation

### PowerShell Framework

* **Version**: PowerShell 5.1+ compatible
* **Execution Policy**: Bypass required for operation
* **Module Dependencies**: Windows Forms, System.Drawing
* **Error Handling**: Comprehensive try-catch blocks

### Windows Forms GUI

* **Framework**: .NET Windows Forms
* **Styling**: Modern flat design with custom colors
* **Controls**: Dropdowns, buttons, text areas, panels
* **Event Handling**: Real-time updates and validation

### File Operations

* **ROBOCOPY Integration**: Reliable file copying with fallback
* **Path Handling**: Quoted paths for special characters
* **Recursive Scanning**: Deep directory traversal
* **JSON Serialization**: Complex object storage

### Task Scheduler Integration

* **High Privilege Execution**: Administrator-level installation
* **Temporary Task Creation**: Install-specific scheduling
* **Process Monitoring**: Installation completion detection
* **Cleanup**: Automatic task removal

## Code Structure

### Main Components

#### 1. Global Variables (Lines 1-50)

# Package selection variables  
$global:selectedVendor = ""  
$global:selectedApplication = ""  
$global:selectedVersion = ""  
$global:selectedDeployment = ""  
$global:selectedDRMBuild = ""  
  
# Package file variables  
$global:filemsi = ""  
$global:filemst = ""  
$global:fileexe = ""  
$global:fileps1 = ""  
  
# Configuration variables  
$global:packageType = ""  
$global:installCommand = ""  
$global:exeArguments = ""

#### 2. GUI Creation Functions (Lines 51-500)

* Add-ModernDropdownControls: Dynamic dropdown creation
* Add-ModernExeArgumentsControl: Parameter configuration interface
* Add-ModernActionButtons: Installation and utility buttons

#### 3. Event Handlers (Lines 501-800)

* Package source selection (Archive/Completed/Local)
* Dropdown change events with dynamic population
* Button click handlers for all operations

#### 4. Package Management (Lines 801-1500)

* Load-LocalPackages: Local file browser and analysis
* Invoke-PackageDownload: Network package retrieval
* Determine-PackageType: Automatic type detection
* Build-InstallCommand: Dynamic command generation

#### 5. Installation Engine (Lines 1501-2000)

* Invoke-InstallPackage: Main installation orchestrator
* Invoke-UninstallPackage: Uninstallation management
* Task Scheduler integration functions

#### 6. Scanning System (Lines 2001-3500)

* Invoke-BaseScan: Pre-installation system state
* Invoke-SecondScan: Post-installation comparison
* File system, registry, and log scanning functions

#### 7. Reporting Engine (Lines 3501-4000)

* Generate-CreativeHTMLReport: Professional report creation
* Generate-ValidationSections: Detailed change analysis
* Generate-TSV: Registry data export

#### 8. Utility Functions (Lines 4001-4500)

* Copy-Logs: Log file management
* Refresh-Form: Interface reset
* Run-QCTool: External tool integration

## Installation & Deployment

### Prerequisites

1. **Windows PowerShell 5.1+**
2. **Administrator Privileges**
3. **Network Access** (for package archives)
4. **Disk Space**: Minimum 5GB free on C: drive

### Deployment Steps

1. **Create Directory**: C:\OPC\V2.0\
2. **Copy Files**:
   * PackageDevManager.exe (Main application)
   * QC\_Tool.EXE (Quality control tool)
   * PackageDevManager.ico (Application icon)
3. **Set Permissions**: Ensure administrator access
4. **Test Launch**: Verify application starts correctly

### Configuration

* **Network Paths**: Update archive locations in code if needed
* **Log Directories**: Verify UHG logs path accessibility
* **Tool Integration**: Ensure QC\_Tool.EXE is present

## User Guide

### Getting Started

1. **Launch Application**: Run PackageDevManager.exe as Administrator
2. **Select Package Source**: Choose Archive, Completed, or Local
3. **Browse/Select Package**: Use dropdowns or file browser
4. **Configure Parameters**: Modify installation arguments if needed
5. **Execute Installation**: Click Install button
6. **Review Results**: Check generated reports and logs

### Package Source Options

#### Network Packages

1. Click **Package Archive** or **Completed Package**
2. Select **Vendor** from dropdown
3. Select **Application Name**
4. Select **Version**
5. Select **Deployment**
6. Select **DRM Build**
7. Package downloads automatically

#### Local Packages

1. Click **LOCAL** button
2. Browse for installer file (MSI/EXE)
3. Review auto-populated parameters
4. Modify arguments if needed
5. Proceed with installation

### Advanced Features

#### Custom Arguments

* Enable “Custom Arguments” checkbox
* Modify installation parameters
* Add logging or silent switches
* Test different installation scenarios

#### Validation and Reporting

* Use **Install Validate** to open install reports
* Use **Uninstall Validate** to open uninstall reports
* Generate **TSV** files for registry analysis
* Copy logs using integrated log management

## Testing Procedures

### Pre-Installation Testing

* Verify package source accessibility
* Confirm package type detection
* Validate command generation
* Check parameter customization

### Installation Testing

* Monitor before/after scans
* Verify installation execution
* Check log file creation
* Validate system changes

### Post-Installation Validation

* Review HTML reports for accuracy
* Verify difference detection
* Check UHG logs integration
* Test uninstallation process

### Report Verification

* Confirm professional styling
* Validate data accuracy
* Check status label correctness
* Verify file generation

## Version History

### Version 2.0 (Current)

* **Enhanced LOCAL Function**: File browser integration
* **Improved UHG Logs**: Recursive scanning and filtering
* **Fixed Status Labels**: Operation-specific reporting
* **Enhanced Error Handling**: ROBOCOPY fallback and validation
* **Professional Reporting**: Modern HTML with Optum branding
* **TSV Generation**: Registry data export functionality

### Key Improvements from V1.5

* Complete LOCAL workflow implementation
* Enhanced difference detection algorithms
* Professional report styling and branding
* Comprehensive error handling and logging
* Improved user interface and experience

## Development Information

**Developed by**: Bharadwaj @ GPS Packaging Team  
**Copyright**: © 2025 Optum  
**Platform**: Windows PowerShell with .NET Windows Forms  
**Architecture**: Modular design with separation of concerns  
**Documentation**: Comprehensive inline comments and help text

## Detailed Code Explanations

### Key Algorithms and Functions

#### 1. Package Type Detection Algorithm

function Determine-PackageType {  
 # Priority-based detection system  
 if ($global:exeExist -and $global:ps1Exist) {  
 # PSADTK: ServiceUI.exe + PowerShell script  
 $global:packageType = "PSADTK Wrapper"  
 $global:installCommand = "`"$global:fileexe`" `"$global:fileps1`" -DeployMode Silent -DeploymentType Install"  
 }  
 elseif ($global:msiExist -and $global:mstExist) {  
 # MSI with Transform  
 $global:packageType = "MSI + MST"  
 $logFile = "C:\Windows\UHGLOGS\$($global:selectedApplication)\_$($global:selectedVersion)\_Install.log"  
 $global:installCommand = "msiexec.exe /i `"$global:filemsi`" TRANSFORMS=`"$global:filemst`" /qn /norestart /l\*v `"$logFile`""  
 }  
 elseif ($global:msiExist) {  
 # Standard MSI  
 $global:packageType = "MSI Only"  
 $logFile = "C:\Windows\UHGLOGS\$($global:selectedApplication)\_$($global:selectedVersion)\_Install.log"  
 $global:installCommand = "msiexec.exe /i `"$global:filemsi`" /qn /norestart /l\*v `"$logFile`""  
 }  
 elseif ($global:exeExist) {  
 # Executable installer  
 $global:packageType = "EXE File"  
 if ($global:exeArguments) {  
 $global:installCommand = "`"$global:fileexe`" $global:exeArguments"  
 } else {  
 $global:installCommand = "`"$global:fileexe`" /S /silent"  
 }  
 }  
}

#### 2. Enhanced UHG Logs Comparison

function Compare-UHGLogs {  
 param($beforeUHG, $afterUHG, $OperationType)  
  
 # Create path arrays for efficient comparison  
 $beforePaths = @($beforeUHG | ForEach-Object { $\_.FullName })  
 $afterPaths = @($afterUHG | ForEach-Object { $\_.FullName })  
  
 if ($OperationType -eq "Install") {  
 # Install: Find new and modified files  
 $newPaths = $afterPaths | Where-Object { $beforePaths -notcontains $\_ }  
 $newFiles = @()  
 if ($newPaths) {  
 $newFiles = @($afterUHG | Where-Object { $newPaths -contains $\_.FullName })  
 }  
  
 # Find modified files (same path, different timestamp/size)  
 $modifiedFiles = @()  
 foreach ($afterFile in $afterUHG) {  
 $beforeFile = $beforeUHG | Where-Object { $\_.FullName -eq $afterFile.FullName }  
 if ($beforeFile) {  
 $afterTime = [DateTime]::Parse($afterFile.LastWriteTime)  
 $beforeTime = [DateTime]::Parse($beforeFile.LastWriteTime)  
 if ($afterTime -gt $beforeTime -or $afterFile.Length -ne $beforeFile.Length) {  
 $modifiedFiles += $afterFile  
 }  
 }  
 }  
  
 # Combine results safely  
 $diffUHGLogs = @()  
 if ($newFiles) { $diffUHGLogs += $newFiles }  
 if ($modifiedFiles) { $diffUHGLogs += $modifiedFiles }  
 }  
 else {  
 # Uninstall: Find removed, new, and modified files  
 $removedPaths = $beforePaths | Where-Object { $afterPaths -notcontains $\_ }  
 $removedFiles = @()  
 if ($removedPaths) {  
 $removedFiles = @($beforeUHG | Where-Object { $removedPaths -contains $\_.FullName })  
 }  
  
 $newPaths = $afterPaths | Where-Object { $beforePaths -notcontains $\_ }  
 $newFiles = @()  
 if ($newPaths) {  
 $newFiles = @($afterUHG | Where-Object { $newPaths -contains $\_.FullName })  
 }  
  
 # Combine all changes  
 $diffUHGLogs = @()  
 if ($removedFiles) { $diffUHGLogs += $removedFiles }  
 if ($newFiles) { $diffUHGLogs += $newFiles }  
 if ($modifiedFiles) { $diffUHGLogs += $modifiedFiles }  
 }  
  
 return $diffUHGLogs  
}

#### 3. ROBOCOPY with Fallback Implementation

function Invoke-PackageDownload {  
 # Build source and destination paths  
 $sourcePath = "$basePath\$global:selectedVendor\$global:selectedApplication\$global:selectedVersion\$global:selectedDeployment\$global:selectedDRMBuild"  
 $destinationPath = "C:\temp\$global:selectedApplication\$global:selectedVersion\$global:selectedDRMBuild"  
  
 # Ensure paths are properly quoted for ROBOCOPY  
 $quotedSourcePath = "`"$sourcePath`""  
 $quotedDestinationPath = "`"$destinationPath`""  
  
 $robocopyArgs = @(  
 $quotedSourcePath,  
 $quotedDestinationPath,  
 "/E", # Copy subdirectories including empty ones  
 "/R:3", # Retry 3 times on failed copies  
 "/W:5", # Wait 5 seconds between retries  
 "/MT:8", # Multi-threaded copy (8 threads)  
 "/LOG:C:\temp\robocopy.log"  
 )  
  
 $robocopyResult = Start-Process "robocopy" -ArgumentList $robocopyArgs -Wait -PassThru -NoNewWindow  
  
 # Check robocopy result (exit codes 0-7 are success, 8+ are errors)  
 if ($robocopyResult.ExitCode -le 7) {  
 Write-Host "Package download completed successfully!" -ForegroundColor Green  
 return $true  
 }  
 else {  
 Write-Host "ROBOCOPY failed with exit code: $($robocopyResult.ExitCode)" -ForegroundColor Red  
 Write-Host "Attempting fallback copy method..." -ForegroundColor Yellow  
  
 try {  
 # Fallback: Use PowerShell Copy-Item with force  
 Copy-Item -Path "$sourcePath\\*" -Destination $destinationPath -Recurse -Force -ErrorAction Stop  
 Write-Host "Fallback copy completed successfully!" -ForegroundColor Green  
 return $true  
 }  
 catch {  
 Write-Host "Fallback copy also failed: $($\_.Exception.Message)" -ForegroundColor Red  
 return $false  
 }  
 }  
}

#### 4. Task Scheduler Integration

function Invoke-InstallPackage {  
 # Create unique task name  
 $taskName = "PackageInstall\_$([System.Guid]::NewGuid().ToString().Substring(0,8))"  
  
 # Build task scheduler command  
 $taskCommand = "schtasks.exe"  
 $taskArgs = @(  
 "/Create",  
 "/TN", "`"$taskName`"",  
 "/TR", "`"$global:installCommand`"",  
 "/SC", "ONCE",  
 "/ST", "00:00",  
 "/RL", "HIGHEST",  
 "/F"  
 )  
  
 # Create and execute task  
 $createResult = Start-Process $taskCommand -ArgumentList $taskArgs -Wait -PassThru -NoNewWindow  
  
 if ($createResult.ExitCode -eq 0) {  
 # Run the task immediately  
 $runArgs = @("/Run", "/TN", "`"$taskName`"")  
 $runResult = Start-Process $taskCommand -ArgumentList $runArgs -Wait -PassThru -NoNewWindow  
  
 # Monitor task completion  
 do {  
 Start-Sleep -Seconds 2  
 $taskStatus = schtasks.exe /Query /TN "$taskName" /FO CSV | ConvertFrom-Csv  
 $status = $taskStatus.Status  
 } while ($status -eq "Running")  
  
 # Clean up task  
 $deleteArgs = @("/Delete", "/TN", "`"$taskName`"", "/F")  
 Start-Process $taskCommand -ArgumentList $deleteArgs -Wait -NoNewWindow  
 }  
}

#### 5. HTML Report Generation Engine

function Generate-CreativeHTMLReport {  
 param($reportData, $reportPath, $operationType)  
  
 # Professional CSS styling  
 $cssStyles = @"  
 <style>  
 body { font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif; margin: 0; padding: 0; background: linear-gradient(135deg, #667eea 0%, #764ba2 100%); }  
 .container { max-width: 1200px; margin: 0 auto; background: white; box-shadow: 0 0 20px rgba(0,0,0,0.1); }  
 .header { background: linear-gradient(135deg, #FF6B35 0%, #F7931E 100%); color: white; padding: 30px; text-align: center; position: relative; }  
 .logo { position: absolute; right: 30px; top: 50%; transform: translateY(-50%); height: 60px; }  
 .section { margin: 30px; padding: 25px; border-radius: 8px; box-shadow: 0 2px 10px rgba(0,0,0,0.1); }  
 .section h2 { color: #FF6B35; border-bottom: 2px solid #FF6B35; padding-bottom: 10px; }  
 .status-added { color: #28a745; font-weight: bold; }  
 .status-removed { color: #dc3545; font-weight: bold; }  
 .status-created { color: #007bff; font-weight: bold; }  
 .status-deleted { color: #6c757d; font-weight: bold; }  
 table { width: 100%; border-collapse: collapse; margin-top: 15px; }  
 th, td { padding: 12px; text-align: left; border-bottom: 1px solid #ddd; }  
 th { background-color: #f8f9fa; font-weight: 600; }  
 .footer { background: #343a40; color: white; text-align: center; padding: 20px; }  
 </style>  
"@  
  
 # Dynamic content generation based on operation type  
 $statusLabels = if ($operationType -eq "Install") {  
 @{  
 Files = "Added"  
 Registry = "Added"  
 Shortcuts = "Created"  
 UHGLogs = "Created"  
 }  
 } else {  
 @{  
 Files = "Removed"  
 Registry = "Removed"  
 Shortcuts = "Deleted"  
 UHGLogs = "Removed"  
 }  
 }  
  
 # Build HTML sections dynamically  
 $htmlContent = Build-HTMLSections -reportData $reportData -statusLabels $statusLabels  
  
 # Combine all elements  
 $fullHTML = @"  
<!DOCTYPE html>  
<html>  
<head>  
 <title>Package Deployment Manager 2.0 - $operationType Report</title>  
 $cssStyles  
</head>  
<body>  
 $htmlContent  
</body>  
</html>  
"@  
  
 # Save and open report  
 $fullHTML | Out-File -FilePath $reportPath -Encoding UTF8  
 Start-Process $reportPath  
}

## Advanced Configuration

### Network Path Configuration

# Update these paths in the global variables section  
$global:sharedSoftwarePath = "\\nasv0718.uhc.com\packagingarchive"  
$global:completedPackagesPath = "\\nas00036pn\Cert-Staging\2\_Completed Packages"

### Logging Configuration

# UHG Logs directory (standard location)  
$global:uhgLogsPath = "C:\Windows\UHGLOGS"  
  
# Custom log naming convention  
$logFileName = "$($global:selectedApplication)\_$($global:selectedVersion)\_$operationType.log"

### Scan Exclusions

# Files and folders to exclude from UHG logs scanning  
$excludePatterns = @(  
 "\*HealthCheck\*",  
 "\*BrowserUpdateManager\*",  
 "\*.tmp",  
 "\*.temp"  
)

## Performance Optimization

### Scanning Optimization

* **Parallel Processing**: Multi-threaded file operations
* **Selective Scanning**: Target-specific directory traversal
* **Efficient Comparison**: Hash-based change detection
* **Memory Management**: Streaming large file operations

### Network Operations

* **ROBOCOPY Multi-threading**: 8-thread parallel copying
* **Retry Logic**: Automatic retry with exponential backoff
* **Fallback Methods**: PowerShell Copy-Item as backup
* **Progress Monitoring**: Real-time status updates

## Security Considerations

### Privilege Management

* **Administrator Rights**: Required for system-level operations
* **Task Scheduler**: High-privilege execution context
* **File System Access**: Full control over installation directories
* **Registry Access**: Read/write permissions for uninstall keys

### Data Protection

* **Log Sanitization**: Removal of sensitive information
* **Temporary File Cleanup**: Automatic cleanup of working directories
* **Network Security**: Secure UNC path access
* **Error Handling**: Graceful failure without data exposure

## Maintenance and Updates

### Regular Maintenance

* **Log Rotation**: Periodic cleanup of UHG logs
* **Cache Management**: Removal of old package downloads
* **Report Archival**: Organized storage of generated reports
* **Performance Monitoring**: System resource usage tracking

### Update Procedures

1. **Backup Current Version**: Preserve working configuration
2. **Test New Features**: Validate functionality in test environment
3. **Deploy Updates**: Replace executable and supporting files
4. **Verify Operation**: Confirm all features work correctly

*This comprehensive documentation provides complete technical and user guidance for Package Development Manager 2.0. For additional support or feature requests, contact the GPS Packaging Team.*