liquidware

ProfileUnity™ with FlexApp™ Technology

FlexApp™ Packaging Automation

Release 1.6.1

March 26, 2025

This guide has been authored by experts at Liquidware in order to provide information and guidance concerning the ProfileUnity with FlexApp Packaging Automation framework.

Liquidware Labs, Inc.

3600 Mansell Road
Suite 2000
Alpharetta, Georgia 30022
U.S.A.

Phone: 678-397-0450

Web: www.liquidware.com

Table of Contents

What's New for FlexApp Packaging Automation?

Version 1.6.1 - Released March 27, 2025

What's New

- `package-create.exe` has been renamed to `fpa-packager.exe`.
- `install agent` now sends FQDN instead of IP when `/PrimaryAddress` is used to register with a Primary-Service.

Version 1.6.0 - Released November 20, 2024

Issues Resolved

- Increased number of installers supported from three to ten, allowing for a single package to contain a larger amount of installations.
- Added ability to configure package job retry behavior with the default behavior configured to retry failed jobs equal to the number of agents available.
- User registry data from HKCU and HKU can now be excluded from capture with the default behavior configured to still capture said data.
- Added support for AppData content capture.

Version 1.5.1 - Released February 28, 2024

Issues Resolved

- Fixed an issue where temporary installation directory was prematurely deleted.

Version 1.5.0 - Released November 8, 2023

What's New

- Service Account and User Compatibility Improvements:
    - Service Account for Packaging: The automation framework now creates a default service account for packaging applications, moving away from using the SYSTEM account. This change enhances compatibility with various application captures, ensuring a smoother and more reliable packaging process.

    - Case-Insensitive Usernames: Usernames for primary clients and agents are now treated as case-insensitive, reducing potential issues related to user account discrepancies and improving overall system resilience.

- Version Compatibility and Package Management:
    - Component Version Matching: All installed FlexApp components, as well as those included in the FlexApp One bundler, now match the version of the core FlexApp based on version 6.8.6. This ensures consistency and reliability across the entire FlexApp suite.
    - FlexApp One Package Creation: During the packaging process, an associated FlexApp One package is now automatically created, with any FlexApp One (FA1) arguments being duly passed along.
    - Package Versioning: Users now have the ability to apply versioning to the packages created, allowing for better management and tracking of application layers.
    - Continuous Capture: The automation framework supports continuous capture, allowing for the replacement of existing packages in a ProfileUnity configuration with newly-created packages.
    - FPC "Offline Mode" Packages.xml Support: New packages can be added to a running inventory outside of the ProfileUnity Console, a "packages.xml" file which can be opened by the FlexApp Packaging Console in "Offline Mode".

- Performance and Efficiency Enhancements:
    - API and Capture Timeouts: The wait time for various API timeouts has been reduced to 10 seconds, and the timeout for the capture process has been increased to 120 minutes. This ensures a more efficient and timely packaging process.
    - High-Compatibility Capturing: A new default capture mode, High-Compatibility Capturing, has been added to enhance the success rate of application captures.
    - OS Optimizations and Runtimes: The capture agent now performs automatic OS optimizations for the cleanest capture possible and installs common runtimes needed for compatibility.

- Capture Agent Enhancements and Installation:
    - AppData Local and Roaming: Data from these locations is now captured by default and copied in during playback or at logon for each assigned user.
    - License File Handling: The capture agent installation process now supports defining the location of a FlexApp One license file, ensuring it is copied to the correct directory. Additionally, the installation process automatically checks the current working directory for the license file.

    - Script and MSI Installer Improvements: MSI installers automatically call 'msiexec /i' with appended '/qn', and script-based installers call the respective executable with appropriate silent arguments.
    - Agent Service Auto-Registration: During an "Install Agent", the /PrimaryAddress arguments can be used to automatically register the new Agent with an existing Primary Packaging Manager. In addition, the same arguments passed during an "Uninstall" will unregister the Agent with a Primary Service during uninstallation.
    - Script-based installation: Scripts can be called in place of an installer, locally downloading the working directory to the capture machine so associated files are available when called during installation. Depending on the extension of the file, scripts are automatically run by the associated interpreter and include the respective silent installation switches.

- Cloud and Storage Enhancements:
    - IAM Role-Based S3 Access: The capture process now supports IAM Role-based S3 access, eliminating the need for access keys and secrets.
    - Custom Storage Locations: The framework now supports custom storage locations using URI format for both S3 and Azure, providing additional flexibility in storage options.
    - Block Caching Enhancements: Packages destined for cloud storage are processed, post-capture, to ensure a successful initial playback even during times of poor network conditions.

- Diagnostics and Logging:
    - ProfileUnity Diagnostic Utility: The product now includes the ProfileUnity diagnostic utility, offering enhanced diagnostics capabilities.
    - Improved Logging: Logs created during package creation now have the package name prepended to the filename, making it easier to identify and troubleshoot issues.

- User Interface and Help Improvements:
    - Help Output Updates: Various areas of the 'help' output for the installer, client, and agent have been updated and corrected, providing clearer guidance and support to users.
    - Full API Documentation: Admins can enable the API documentation page on the Primary or Agent Service, as-needed, when developing workflows and automation integration points.

- Error Handling and Validation:
    - Capture Job Retries: The Primary Packaging Manager Service will retry certain types of failures as many times as there are Capture Agents. This helps keep overall success rate high in the event a Capture Agent is experiencing an issue mid-job. If there were any, retry counts would be included in the Job Summary.

    - Improved Error Pass-Through/Recording: Enhancements have been made to the error pass-through and recording process from the capture agent to the client, ensuring that issues are promptly identified and resolved.
    - Validation Checks: Additional validation checks have been implemented against numerous parameters passed during the packaging process, including files, paths, and credentials, further ensuring the integrity of the packaging process.

- Command Line Interface (CLI) and Job Management:
    - Enhanced CLI Support: The package creation process now accepts all possible packaging arguments via CLI, and various 'agent' and 'packages' commands have been enhanced to support passing primary credentials and other essential parameters.
    - Job Filtering and Retry Mechanism: Users can now filter packaging jobs based on their status, and failed captures are retried a number of times equal to the number of known agents, improving the robustness of the packaging process.

Issues Resolved

- Fixed an issue where packages could be identically named
- Fixed an issue where packages failed to be created on Windows Server OS due to system restore attempt.
- Fixed an issue where FPA-created packages were unable to be cloned in the FlexApp packaging console in certain scenarios.
- Fixed an issue with packages that do not contain shortcuts fail to capture.
- Fixed an issue where an MSI installer path that contained spaces fail to capture.
- Fixed an issue where the Service would remain after a failed installation of either primary or agent.
- Fixed an issue where lwl_proc_info system driver is not removed during uninstallation.
- Fixed an issue with Existing packages were unable to be extended.
- Fixed an issue with Creating packages directly on the client did not honor parameters present in the argument file.

Version 1.0.25 – Released August 18, 2021

The first version of FlexApp Packaging Automation is released to market. This framework allows for multiple and concurrent unattended FlexApp Package captures.

FlexApp Packaging Automation Overview

In some cases, one-off captures of applications for dynamic layering to end users can be repetitive and time-consuming. This applies to any application layering or virtualization technology—call it capturing, recording, or sequencing—packages or layers must be created and tested prior to being deployed to production.

Some situations call for a more automated packaging process. Consider scenarios like migrating from another application deployment tool or method to FlexApp, compliance-mandated scheduled application updates, or even integrating FlexApp package creation as a component of a larger DevOps system to reduce time and intervention in getting new software builds deployed to end users.

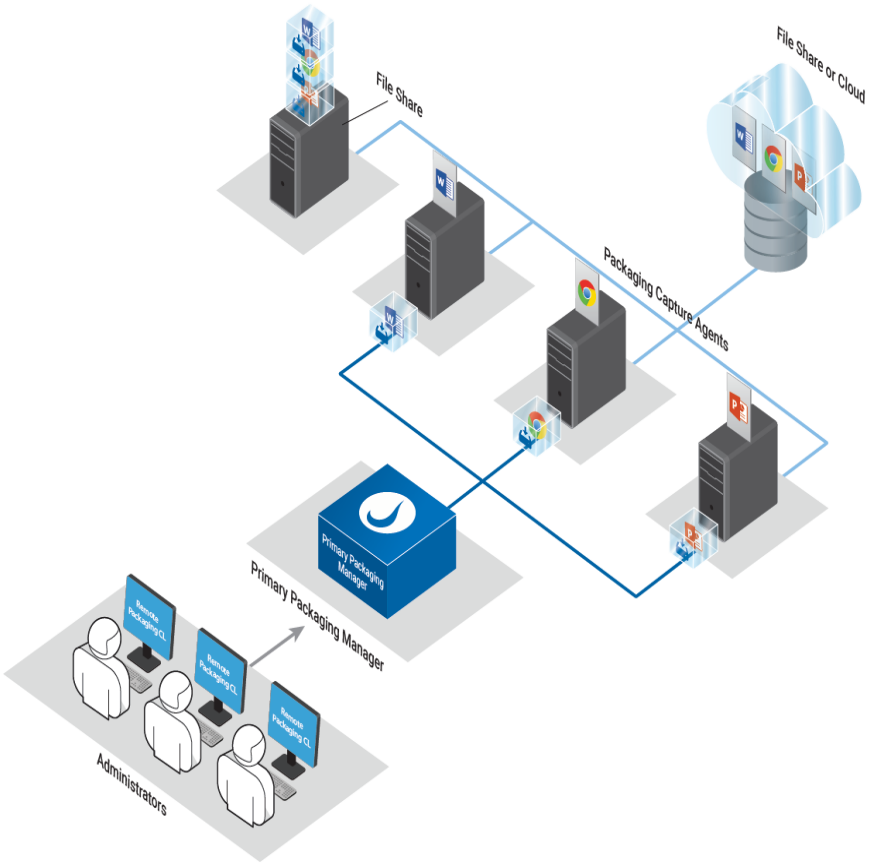
To address requests of this nature, Liquidware has created the FlexApp Packaging Automation framework. FlexApp Packaging Automation (FPA), uses a Primary Packaging Manager to delegate packaging jobs to a network of Packaging Capture Agents that execute the silent install commands, capture the software being installed, and create FlexApp packages on a network share that are immediately ready to be assigned to end users.

Going deeper into these example scenarios:

- If an organization is using a tool like SCCM, Intune, etc. to push out silent installs to end users, then migrating your application deployments to FlexApp can be accelerated using FlexApp Packaging Automation by building packaging jobs using the installers and silent install commands already contained in your current software deployment tool.
- If an internal policy requires some or all your end user software applications to be updated on a recurring schedule, maybe for security or industry-compliance reasons, then FlexApp Packaging Automation can be used to accelerate the repetitive re-packaging of the new versions to shorten the time-to-live for version updates and reduce manual IT intervention.
- If an organization relies on an application that is developed in-house, FlexApp Packaging Automation can be called by the software build server as the last step of a DevOps process to create a FlexApp package from the resultant binaries or installer. After the process is complete, that package shows in the ProfileUnity Console ready to be assigned to end users.

Architecture and Multi-Admin Usage

There are various use cases one can come up with around the use of FlexApp Packaging Automation. Here is one example of a common architecture that can help you plan your installation of FPA.



This architecture would allow packaging jobs to be submitted from the Primary Packaging Manager or remotely by one or more admins using the Remote Packaging CLI. In addition, you can conserve resources and use your existing ProfileUnity Management Console machine as the Primary Packaging Manager.

Additional Architectural Considerations and Use Cases

There are more architectural considerations and use cases around the use of FlexApp Packaging Automation than those described in Architecture and Multi-Admin Usage. Here are additional use case examples that can help you plan your installation of FPA.

Considerations

Some of the considerations are the following:

- Do you need to be able to perform multiple captures simultaneously?
  - Maybe you do not need a Primary Packaging Manager; only a single Capture Agent.
- Do you need to integrate with an existing automation workflow?
  - In addition to not needing a Primary Packaging Manager, maybe you do not use a dedicated Capture Agent and prefer to install the FPA Capture Agent on a machine to run synchronous or in-line with an existing process or script.

Automated captures as a part of DEVOPS

For automation workflows that output updated applications on a regular basis, you can incorporate a single Capture Agent machine without the need for a Primary Packaging Manager.

- The existing automation server would get the FPA Remote CLI installed, and optionally, a Defaults.JSON file created with encrypted credentials and presets.
- The automation workflow would be appended with a call to the FPA Agent Service to start a capture of a script or silent installer, asynchronously or synchronously, on a remote Capture Agent machine:

```
Agent-Client.exe Create Package /AgentAddress <https://agent-server:9074>
/AgentUsername "fpa_services" /AgentPassword "<pass>" /Name "<PackageName>"
/PackageVersion <n.n.n.n> /Path "\\fileserver\share\my-new-software-packages"
/Installer "\\fileserver\share\my-software\silent-installation-script.ps1"
/PathUsername "<domain\user>" /PathPassword "<pass>" /PuAddress
<https://ProuServerAddress:8000> /PuUsername "<domain\user>" /PuPassword
"<pass>"
```

On-Agent scripted synchronous captures

It is also possible to bypass the Agent Service of a Capture Agent and execute a capture directly on the Capture Agent machine via the command line. This can help with scenarios where you need to run scripts on the Capture Agent before a capture and then continue the script upon completion.

- The script could invoke a capture directly, in-session even, bypassing the Agent Service.

```
fpa-packager.exe Package /Name "<PackageName>" /PackageVersion <n.n.n.n> /Path
"\\fileserver\share\my-new-software-packages" /Installer "\\fileserver\share\my-
software\installation-script.bat" /NoSystemRestore
```

- The Package-Create process would end when the capture is complete and the script could continue and could then take action by moving the new package, etc.
- If executed in-session against an installer or script that is not silent, the user can interact with the installer during capture.

FlexApp Packaging Automation Requirements

The FlexApp Packaging Automation framework is made up of a Primary Packaging Manager and one or more Packaging Capture Agents that run the packaging jobs and place the resulting packages in the specified storage locations. Capture Agent VM must be 64-bit architecture.

Primary Packaging Manager

Component	Minimum Requirements
Virtual Machines	1 VM to be used as the Primary Packaging Manager. The ProfileUnity Management Console can be used as the Primary Packaging Manager or a new VM can be created.
VM OS Support	Windows 10/11 or Windows Server 2016/2019/2022/2025
VM vCPU	2 vCPU
VM Memory	4 GB
VM Storage	1 GB free on C:
Prerequisites	Microsoft® .NET 8.0 is automatically installed
Network and Firewall	Primary Packaging Manager listens on TCP/9075 and connects to TCP/9074 on Packaging Capture Agents.

Packaging Capture Agents

Component	Minimum Requirements
Virtual Machines	1+ 64-bit VM(s) to be used as Packaging Capture Agents *Liquidware recommends 3+ VMs for large batch jobs and 2+ for linear/DEVOPS/one-off processing to maintain redundancy.
VM OS Support	Windows 10/11 or Windows Server 2016/2019/2022/2025
VM vCPU	4+ vCPU
VM Memory	8+ GB
VM Storage	40 GB free on C:
VM Free Space	Free disk space is needed locally, usually C:, on each Packaging Capture Agent VM for capturing application installations. It is recommended to have at least double what you think you will ever need for your largest capture.

Component	Minimum Requirements
Prerequisites	Microsoft® .NET 8.0 is automatically installed.
Network and Firewall	Packaging Capture Agents connect to TCP/9075 on Primary Packaging Manager.

Network or Cloud Storage

A network file share or a cloud location used to store new packages is required. In addition, storage for the installer EXEs and MSIs that will be accessed by the Capture Agents will be needed.

Silent Install Requirement

Applications are required to install silently due to the headless nature of FPA. Most application installers require the use of flags on the CLI to indicate a silent install. Test each installer's silent install commands manually prior to using in a packaging job. There must be no user-interaction involved during install. Prompts will hang capture until the timeout.

Multi-Administrator Package Creation

When multiple package administrators are sending packaging jobs to the same Primary Packaging Manager it is recommended that any PathUsername or InstallerUsername be a service account created for this purpose and stored in the DefaultsJSON file. As a result, the service account will handle all file operations on the network share preventing the need for each admin to use their own credentials.

Installation: FPA-Installer.exe Command Line Arguments

The following describes the available commands for FPA-Installer.exe. Be sure to wrap all complex passwords in double-quotes like "&()[]{}^=;!'+,`~ my c0mpl3x PW". As a result of this requirement, double quotes cannot be used in passwords or AES secrets. Paths that include a space also require double quotes.

Notes about the different Username arguments:

- `/PrimaryUsername` and `/AgentUsername` are created as "internal FPA Service API users" that do not exist as local Windows or Domain accounts. Using "fpa_services" is only a suggestion.
- `/ServiceUsername` must be a local Windows or Domain account. If a non-existent local account is specified, it will be created as a non-Administrator user. Local accounts are specified by using the format `".\username"`.
- If no `/ServiceUsername` is specified during a "Install Agent" command then a local, non-Administrator Windows account named `"fpa_services"` will be created with a randomized and unsaved password. (Not to be confused with the API accounts previously-mentioned.)

Command	Description
Help	Displays the command line options Usage: `fpa-installer.exe Help`
Eula	Extracts the embedded End-User License Agreement (EULA) to disk for review Usage: `fpa-installer.exe Eula [/Path "<folder>"]`
Install Agent	Installs FPA Packaging Capture Agent Usage: `fpa-installer.exe Install Agent /AgentUsername "fpa_services" /AgentPassword "<pass>" [/Path "<folder>"] [/Temp "<folder>"] [/ServiceUsername "<domain\user>" /ServicePassword "<pass>"] [/ServiceLogPath "<folder>"] [/ServiceLogLevel <level>] [/Port <port>] [/CertificateFile "<path\cert.pfx>" [/CertPassword "<pass>"]] /AcceptEula [/PrimaryAddress <https://server:9075> /PrimaryUsername "<user>"`

Command	Description

	`/PrimaryPassword "<pass>"]`
	`[/SkipOptimizations]`
	`[/SkipRuntimes]`
	`[/Fa1LicenseFile <path\FlexApp_One.lic>]`

Install Primary	Installs FPA Primary Packaging Manager Usage: `fpa-installer.exe Install Primary /PrimaryUsername "fpa_services" /PrimaryPassword "<pass>" [/Path "<folder>"] [/Temp "<folder>"] [/ServiceUsername "<domain\user>" /ServicePassword "<pass>"] [/ServiceLogPath "<folder>"] [/ServiceLogLevel <level>] [/Port <port>] [/CertificateFile "<path\cert.pfx>" [/CertPassword "<pass>"]] /AcceptEula`
Install	Installs FPA Remote Packaging CLI Usage: `fpa-installer.exe Install [/Path "<folder>"] [/Temp "<folder>"] /AcceptEula`
Uninstall	Removes services and installed files but leaves behind .NET 5 and ProgramData Usage: `fpa-installer.exe Uninstall [/Path "<folder>"]` To cause an Agent Service to remove itself from the Primary Service, include the following options when uninstalling an Agent Service: `[/PrimaryAddress <https://server:9075> /PrimaryUsername "<user>" /PrimaryPassword "<pass>"]`

Defaults

By default, the primary-service and agent-service both run as Local System.

Command Line Option	Default Setting
`Eula [/Path]`	`"%USERPROFILE%\Desktop"`
`Install * [/Path]`	`"C:\Program Files (x86)\Liquidware Labs\FlexApp Packaging Automation"`
`Install * [/Temp]`	`"%TEMP%"` (Used for installation only)
`Install * [/ServiceLogPath]`	`"C:\Windows\Temp\fpa"`
`Install * [/Port]`	`Primary=9075, Agent=9074`
`Uninstall [/Path]`	Path used during installation

Setting Up FlexApp Packaging Automation

Download the FlexApp Packaging Automation framework from the *FlexApp Packaging Automation* page.

> **Note:** FPA requires a set of internal credentials, such as prou_services, to secure the API. These are specified during installation and consumed during package creation. They are not used to access any resources and are not linked to Active Directory.

Preparing the Primary Packaging Manager

To prepare the Primary Packaging Manager, complete the following steps:

1. From an elevated Command Prompt, install the Primary Packaging Manager:

```
fpa-installer.exe install primary /PrimaryUsername "fpa_services"
/PrimaryPassword "<pass>" /AcceptEULA
```

2. If you want, add

```
"C:\Program Files (x86) \Liquidware Labs\FlexApp Packaging Automation"
```

to your `PATH` environment variable and open a new `cmd.exe`. Otherwise `cd` there now.

Preparing the Packaging Capture Agents

Liquidware recommends that required frameworks and runtimes be natively installed on end user and Packaging Capture Agent VMs for reduced package overhead. This will happen automatically. In addition, the Capture Agent OS will be optimized for the same reason.

To prepare the Packaging Capture Agents, complete the following steps:

1. Create a new VM for use as a Packaging Capture Agent machine.

   > **Note:** An alternative method for creating Packaging Capture Agents is to clone your existing, already-optimized FlexApp Packaging Console VM and then uninstall the FPC software, reboot and that clone is ready to become an FPA Packaging Capture Agent.

2. From an elevated Command Prompt, install the Packaging Capture Agent:

   ```
   fpa-installer.exe install agent /AgentUsername "fpa_services" /AgentPassword
   "<pass>" /PrimaryAddress <https://primary:9075> /PrimaryUsername "fpa_services"
   /PrimaryPassword "<password>" /AcceptEULA
   ```

3. Ensure you have a “clean state” snapshot taken after installing the Packaging Capture Agent to which you can revert, if needed.

   > **Note:** The optional `/PrimaryAddress` arguments will auto-register this new agent with your Primary Packaging Manager, saving you the step of running Add Agent using `Primary-Client.exe` later. Agents added in this manner are registered using their FQDN rather than IP address.

4. Verify your Agent using the primary-client.exe List Agent command.

(Optional) Installing the Remote Packaging CLI

To install the Remote Packaging CLI, complete the following steps:

1. If you want to run commands from another machine instead of needing to run them from the Primary Packaging Manager, you can install the Remote Packaging CLI on another workstation:

   ```
   fpa-installer.exe install /AcceptEULA
   ```

2. (Optional) Add the following to your `PATH` environment variable:

   ```
   "C:\Program Files (x86)\Liquidware Labs\FlexApp Packaging Automation"
   ```

Testing Package Creation

Testing Scenario

In this example, you will set up files that allow a packaging admin to save their credentials and only need to provide the CryptoKey on the CLI when creating packages. This makes it more convenient to share PackagesFiles with other admins since it would not need to contain login credentials and you also would not need to specify them on the command line each time you run the Create Packages command.

Creating a Test PackagesFile and DefaultsJSON

To create a test PackagesFile and DefaultsJSON, complete the following steps:

1. From your Primary Packaging Manager or the Remote Packaging CLI, create your `PackagesFile` from one of the following:

```
primary-client.exe Add PackagesFile /PackagesFile
"%USERPROFILE%\Desktop\PackagesFile.json" /Name "Notepad++ v8.1.1" /Path
"\\fileserver\FlexApp\Packages" /Installer
"\\fileserver\FlexApp\Installers\npp.8.1.1.Installer.exe" /InstallerArgs "/S"
```

```
primary-client.exe Add PackagesFile /PackagesFile
"%USERPROFILE%\Desktop\PackagesFile.json" /Name "PuTTY Utils v0.76" /Path
"\\fileserver\FlexApp\Packages" /Installer
"\\fileserver\FlexApp\Installers\putty-0.76-installer.msi"
```

2. Next, create the `DefaultsJSON`:

```
primary-client.exe Create DefaultsJSON /DefaultsJSON
"%USERPROFILE%\Desktop\defaults.json" /PrimaryUsername "fpa_services"
/PrimaryPassword "<pass>" /PathUsername "<domain\user>" /PathPassword "<pass>"
/PuAddress <https://ProuServerAddress:8000> /PuUsername "<domain\user>"
/PuPassword "<pass>"
```

Testing Your Packaging Job

Test your packaging job by creating the specified packages. First check your Capture Agents are available:

```
Primary-Client.exe list agent /PrimaryUsername "fpa_services" /PrimaryPassword
"<pass>"
```

Method 1

Create Packages using PackagesFile and DefaultsJSON, wait for the job to finish:

```
primary-client.exe Create Packages /PackagesFile
"%USERPROFILE%\Desktop\PackagesFile.json" /DefaultsJSON
"%USERPROFILE%\Desktop\defaults.json" /WaitForDone
```

```
*Example console status when complete:
------------------------------------------------------------
Batch Job Report

Total:1 Pending:0 Running:0 Successful:1 Failed:0 Cancelled:0

Batch Job Id:'9871' Status:'Successful'

      Start:'11/8/2024 10:04:11 AM' End:'11/8/2024 10:06:04 AM'

  Primary Job Name:'Notepad++' Id:'9f1b' Status:'Successful'

        Start:'11/8/2024 10:04:12 AM' End:'11/8/2024 10:05:44 AM'

        ExitCode:'0'

    Agent Job Name:'Notepad++' Id:'a9f5' Status:'Successful'

        Start:'11/8/2024 10:04:13 AM' End:'11/8/2024 10:05:43 AM'

        ExitCode:'0' RunLocation:'10.30.61.243'


------------------------------------------------------------
```

Method 2

Create Packages using PackagesFile and DefaultsJSON in background, email results:

```
primary-client.exe create Packages /PackagesFile
"%USERPROFILE%\Desktop\PackagesFile.json" /DefaultsJSON
"%USERPROFILE%\Desktop\defaults.json" /MailServer <smtp.something.com> /MailFrom
<x@y.com> /MailTo <a@b.com>
```

*Example email status when complete: (condensed)

Job Name	Status	Agent	MM:SS	Installer	Installer Args	Type	VHD MBs
Notepad++ v8.1.1	Successful	Agent1	0:30	`npp.8.1.1.Installer.exe`	`/S`	`Cloud`	`196.00`
PuTTY Utils v0.76	Successful	Agent2	0:31	`putty-0.76-installer.msi`		`Cloud`	`164.00`

PackagesFile and DefaultsJSON File Contents

- The example file contents will look like this (excluding the unspecified parameters).

  packagesfile.json

  ```
  "Name": "Notepad\u002B\u002B v8.1.1"
  "Path": "\\\\fileserver\\FlexApp\\Packages"
  "Installer": "\\\\fileserver\\FlexApp\\Installers\\npp.8.1.1.Installer.exe"
  "InstallerArgs": "/S"
  "Name": "PuTTY Utils v0.76"
  "Path": "\\\\fileserver\\FlexApp\\Packages"
  "Installer": "\\\\fileserver\\FlexApp\\Installers\\putty-0.76-installer.msi"
  ```

  defaults.json

  ```
  "PrimaryUsername":
  "537F6AA867FB405lookatmeimencrypted3u9Mt3sXxGcaigPxIngPqyw=="
  "PrimaryPassword":
  "537F6AA867FBlookatmeimencryptedz1MjK/h/ZdYpPBuMyg\u002BQ=="
  "PathUsername": "537F6AA867FB4lookatmeimencryptedA8\u002BycoKQXBOgZ5g=="
  "PathPassword": "537F6AA867FB40598lookatmeimencryptedVysei\u002BXyj1EJakAQ=="
  "PuAddress": "https://MyProuServerAddress:8000"
  "PuUsername": "537F6AA867FB405989836E5EDlookatmeimencrypted02BycoKQXBOgZ5g=="
  "PuPassword": "537F6AA867FB4059898lookatmeimencryptedj1EJakAQ=="
  ```

- Editing your `PackagesFile` can be done with a text editor or by using `primary-client.exe` (Add PackagesFile, Remove PackagesFile, List PackagesFile, Clear PackagesFile).
- The `DefaultsJSON` works a bit differently than the `PackagesFile` in that the `primary-client.exe Create DefaultsJSON` command is used to completely overwrite the file with a new file containing only the newly specified parameters. Therefore, a text editor is likely the easiest method to adjust any non-encrypted values.
- When adding information into a `PackagesFile` or `DefaultsJSON`, you have two encryption options for user/pass fields. Users can never see the credentials stored in these files. Refer to the *Notes About Encryption and Log Paths* section of this guide for more information.

Available Packaging Job Parameters

The following list of parameters are available for use on the CLI and in the PackagesFile or Default-sJSON. CLI usage requires a preceding / (ie, /Name "Some Application v1.0") and usage in a JSON file requires the parameter to be in double quotes (ie, "Name":"Some Application v1.0").

Notes:

- Be sure to wrap all complex passwords like "&()[]{}^=;!'+,`~ my c0mpl3x PW" and paths that include a space in double quotes.
- Double quotes cannot be used in passwords or AES secrets.
- `/InstallerArgs` (if any) must be quoted, be the last parameter, and inner quotes must be escaped with \
- If `/Installer` points to a .msi, then `/InstallerArgs` will automatically be set to `/i "<installer.msi>" /qn` and `/Installer` will implicitly become msiexe.exe, but you can still use `/InstallerArgs` to provide additional args in this scenario.
- If multiple paths are provided to network shares, multiple credentials are provided, and they are to the same server then either only use one set of credentials, or make the second reference to the server be via ip.
- If a PackagesFile contains a parameter already set in DefaultsJSON, PackagesFile wins. If parameter also included on CLI, CLI wins.
- All parameters are optional except Name, Path, and Installer

Job Parameter	Description
Name `"<package-name>"`	Package Name *(Required). *Should be unique within a given packaging batch job.

Path
`"<path>"`
	Target folder where the package(s) will be created * (Required).
	`*Path` ending in `.vhdx` will perform an Extend of existing package.
	*Cloud paths are supported -
	`s3://`
	`az://`
	`gs://`

Job Parameter	Description
CustomStorageUrl <url>	Custom S3 service URL or Azure endpoint suffix.
AzureMaximumConcurrency <integer>	Azure download tuning. Default: 64
AzureInitialTransferSizeMb <integer>	Azure download tuning. Default: 1
AzureMaximumTransferSizeMb <integer>	Azure download tuning. Default: 1
Installer "<path>"	Path of the installer exe to be executed and captured * (Required). *CIFS path like \\server-\share\folder\installer.exe, .msi or any script file.* Web URL's also supported.
InstallerArgs "<args>"	Installer-specific silent install flags required for proper operation. *Args must be wrapped in quotes (ie, "/S")
SizeGb <integer>	Package VHDX size. Default: (20GB)
Fixed	Use Fixed VHDX (allocate all space now). Default: Expandable (Grow as needed)
Test	Plays back the package after save and takes screenshots. Default: Don't Test
PathUsername "<domain\user>"	Username used to access the package's path. *Cloud paths use s3=Access Key, az=Account Name, gs=<omit> *Specifying a PathUsername of "IAM" will allow use of Amazon S3 IAM credentials.
PathPassword "<password>"	Password used to access the package's path. *Cloud use s3=Secret Key, az=Account Key, gs=<credential.json>

Job Parameter	Description
InstallerUsername "<domain\user>"	Username used to access the installer's path.
InstallerPassword "<password>"	Password used to access the installer's path.
InstallerExitCode <integer>	Expected installer SUCCESS exit code. Default: 0
InstallerTimeoutMs <integer>	How long to wait for an installer to finish before failing the capture. Default: 3600000 (1hr)
PuAddress <https://ProuServerNameOrIP:8000>	The ProfileUnity Console where new packages will be imported.
PuUsername "<domain\user>"	Username used to access the ProfileUnity Console.
PuPassword "<password>"	Password used to access the ProfileUnity Console.
NoSystemRestore	Do not perform a System Restore rollback after capture/extend. Default: False (Use System Restore to roll back).
AltRestoreCmd "<pathToScript>"	Instead of System Restore, use a rollback script after capture/extend. *Runs from the Agent VM and implies NoSystemRestore=True Default: Rollback determined by NoSystemRestore parameter.
AltRestoreCmdArgs "<args>"	Args (if any) needed for the AltRestoreCmd *Args must be wrapped in quotes (ie, "/S")
WaitAfterInstallerExitsMs <integer>	How long to wait after installation ends before saving the capture. Default: 0

Job Parameter	Description
DontCopyInstallerLocal	Run the installer directly from the `Installer` path. Default: Copy installers locally before executing (`"C:\Windows\Temp\fpainstaller" or "%TEMP%\fpainstaller"`)
CopyInstallerFolderLocal	Copies the the folder containing installer files locally before executing Default: `Do not copy installer folder`
InstallerFolder `"<path>"`	Path of the folder containing the installer files.
NoHCCapture	Do not use high compatibility capture mode. Default: `False`
Installer2 `"<path>"`	Path of an additional installer to be executed during the capture.
Installer10 `"<path>"`	Path of an additional installer to be executed during the capture. Up to 10 installers are supported.
InstallerArgs2 `"<args>"` InstallerArgs10 `"<args>"`	Installer-specific silent install flags required for proper operation. *Args must be wrapped in quotes (ie,`"/S"`)
InstallerExitCode2 `<integer>` InstallerExitCode10 `<integer>`	Expected installer exit code used to determine successful installation. Default: `0`
PreActivationScript `"<path>"`	CMD file to include in package and execute before playback.
PostActivationScript `"<path>"`	CMD file to include in package and execute after playback.
PreDeactivationScript `"<path>"`	CMD file to include in package and execute before stopping playback.

Job Parameter	Description
PostDeactivationScript "<path>"	CMD file to include in package and execute after stopping playback.
NoCallToHome	Do not send job Installer and InstallerArgs stats to Liquidware Default: False
PackageVersion <major.minor.build.revision>	User specified version to help track package changes.
DontCreateFlexAppOne	Do not automatically create a FlexApp One package.
FlexAppOneCliOverride <args>	Custom FlexApp One bundler.exe command line arguments.
DontCaptureUserProfileData	Do not capture user profile data.
DontCaptureUserRegistry	Don't capture user registry data from HKCU or HKU.
DontCapture	Do not capture at all, just install.
PackagesXml	Write a copy of package info into the specified packages.xml. Primarily used by FlexApp Packaging Console in 'offline' mode.
PuConfiguration <configname>	Replaces an existing package in the specified configuration with the created package. *If defined, /PackageVersion and /PuFilter must match those of the existing package.
PuFilter <filtername>	Add created package to flexapp rule with specified filter. *Requires /PuConfiguration.
PuDescription <descr>	Description to use when adding flexapp rule. *Requires /PuConfiguration.
LogPath "<path>"	Folder in which to store logs. Default: "%TEMP%\fpa".
LogLevel <level>	Logging level – Debug, Info, Warn, Error, Fatal (Advanced Logging Levels: Console, Always). Default: Debug

Job Parameter	Description
MailServer <smtp.company.com> (DefaultsJSON and CLI only)	SMTP server to use as a relay for job-related emails. Default: No emails are sent unless specified
MailPort <integer> (DefaultsJSON and CLI only)	SMTP port used for MailServer. Default: 25
MailSsl (DefaultsJSON and CLI only)	Use SSL/TLS when connecting to MailServer. Default: False (No SSL/TLS)
MailUsername "<username>" (DefaultsJSON and CLI only)	Username to use for relaying emails through Mailserver.
MailPassword "<password>" (DefaultsJSON and CLI only)	Password to use for relaying emails through Mailserver.
MailTo <SomeGuy@company.com> (DefaultsJSON and CLI only)	Email address that should receive job-related emails *Required for /MailServer
MailFrom <noreply@company.com> (DefaultsJSON and CLI only)	Email address to show as sender of job-related emails. *Required for /MailServer
CaptureRetryCount	Number of times a capture job is attempted when a failure is encountered. Defult is the number of agents.
PrimaryAddress <https://PrimaryNameOrIp:9075> (DefaultsJSON and CLI only)	Primary Packaging Manager address. Default: https://localhost:9075
PrimaryUsername "fpa_services" (DefaultsJSON and CLI only)	Primary Packaging Manager username.
PrimaryPassword "<password>"	Primary Packaging Manager password.

Job Parameter	Description
(DefaultsJSON and CLI only)	
WaitForDone (CLI only)	Wait for job to finish and allow job logs to be copied locally. Default: `Return to cmd prompt after submitting packaging job`
Crypto `Aes` (CLI only)	Encrypt the user/pass fields contained in the command line w/Aes *See the Notes About Encryption and Log Paths section below for more information. Default: Encrypt all user/pass fields using machine-specific `DpApi`
CryptoKey `"<aes-secret-key>"` (CLI only)	AES Encryption Passphrase / Key. *Implies `/Crypto Aes`

Notes About Encryption and Log Paths

When adding information into a PackagesFile or DefaultsJSON, you have two encryption options for username and password fields. In any case, users can never see the credentials stored in these files- they can only use them to process primary-client commands. If you do not specify `/Crypt`, the default of `DpApi` is used.

- `/Crypto DpApi` is a self-contained, machine-based encryption using Windows Data Protection API. Anyone with access to the machine can utilize (but not see) the credentials without needing an encryption key. Files encrypted with this method cannot be moved to another machine. **THIS IS THE DEFAULT!**
- `/Crypto Aes` is an AES-based encryption method utilizing a secret key and requires the key to be passed on the command line with `/CryptoKey` to utilize (but not see) the credentials. Files encrypted with this method can be shared between different machines as long as the secret key is known. `/CryptoKey` implies `/Crypto Aes`.
- `/LogPath` and `/LogLevel` used on the CLI apply only to the `primary-client.exe` log for the current command. `"LogPath"` and `"LogLevel"` used within a JSON file apply only to the capture job log on the capture agent. There is no automatic authentication for a UNC-based `LogPath` so the capture agent's machine account or service account will need "unprompted access" to the `LogPath`.

Agent-Client.exe Commands

The following describes the available commands for agent-client.exe. Be sure to wrap all complex passwords in double-quotes like "&()[]{}^=;!'+,`~ my c0mpl3x PW". As a result of this requirement, double quotes cannot be used in passwords or AES secrets. Paths that include a space also require double quotes.

Command	Description
Help	Displays the command line options Usage: `agent-client.exe Help`

Create Package	Creates a new package
	Usage: `agent-client.exe Create Package`
	`[/AgentAddress <https://server:9074>] /AgentUsername <user> /AgentPassword "<pass>"`
	`[/ArgFile <path\agent-client.args.json>]`
	`[/CryptoKey "<my-aes-secret-key>"]`
	`/Name "<name>"`
	`/Path <path>` ***SEE BELOW**
	`[/PathUsername <domain\user> /PathPassword "<pass>"]`
	`[/AzureMaximumConcurrency <int>]` -Azure download tuning. Default 64.
	`[/AzureInitialTransferSizeMb <int>]` -Azure download tuning. Default 1.
	`[/AzureMaximumTransferSizeMb <int>]` -Azure download tuning. Default 1.
	`[/CustomStorageUrl <url>` (Custom S3 service URL or Azure endpoint suffix)].
	`[/SizeGb <GBs>]` -Defaults to 20GB
	`[/Fixed]` -Creates a fixed vhdx instead of the default expandable.
	`[/Test]` -Leaves some .bmp files behind to make it easier to see what shortcuts were captured.
	`[/PuAddress <https://pu.server:8000> /PuUsername "<domain\user>" /PuPassword "<pass>"]` -Registers new packages in inventory.
	`/Installer <path\installer.exe>` ****SEE BELOW**
	`[/InstallerUsername <domain\user> /InstallerPassword "<pass>"]`
	`[/InstallerArgs <args>]`
	`[/InstallerExitCode <int>]` -Expected installer exit code, else error. Default 0

Command	Description
	[/InstallerTimeoutMs <ms>] -How long to wait for the installer to finish, else error. Default 1hr [/WaitAfterInstallerExitsMs <ms>] -Wait after installer exits before completing capture. Default 0 [/DontCopyInstallerLocal] -Will not copy the installer locally before running the installer. Default false; Does copy. [/CopyInstallerFolderLocal] -Copies the folder containing the installer file locally before executing. Default false except for script installers. [/InstallerFolder <path>] -Change from the default installer path of the folder containing the installer to a custom path to be copied locally before executing the installer. [/NoHCCapture] -Do not use high compatibility capture mode. Default false; Use HC Mode. [/NoSystemRestore] -Do not perform a System Restore rollback at the end of the capture/extend. Default false; use System Restore to rollback. [/AltRestoreCmd <path\file>] -Instead of using System Restore, use a cmd to rollback, implies /NoSystemRestore. [/AltRestoreCmdArgs <args>] [/Installer2 <path\installer.exe> **SEE BELOW [/InstallerArgs2 <args>] [/InstallerExitCode2 <int>]] ... [/Installer10 <path\installer.exe> **SEE BELOW [/InstallerArgs10 <args>] [/InstallerExitCode10 <int>]] [/PreActivationScript <path\file>] [/PostActivationScript <path\file>] [/PreDeactivationScript <path\file>] [/PostDeactivationScript <path\file>] [/NoCallToHome] -Do not send installer and installer args data to Liquidware. [/PackageVersion <major.minor.build.revision>] -User specified version to help track package changes.

Command	Description
	`[/DontCreateFlexAppOne]` -Do not automatically create a FlexApp One EXE. Default false; Do create EXE. `[/FlexAppOneCliOverride <args>]` -Custom FlexApp One Bundler command line args. `[/DontCaptureUserProfileData]` -Do not capture user profile data. `[/DontCaptureUserRegistry]` -Don't capture user registry data from HKCU or HKU. `[/DontCapture]` -Do not capture at all, just install. `[/PackagesXml]` -Write a copy of package info into the specified packages.xml so it can be inventoried using FlexApp Packaging Console in "Offline Mode". `[/PuConfiguration <configname> /PuFilter <filtername> [/PuDescription <descr>]]` -Replaces existing package in specified configuration with created package. /Name and /PuFilter must match existing package assignment, with differing /PackageVersion values.

Create ArgFile (Optional, omit all params to get an empty template file)	Creates optional `ArgFile` with default params used by packaging jobs
	Usage:
	`agent-client.exe Create ArgFile`
	`/ArgFile<path\defaults.json>`
	`[/Crypto Aes /CryptoKey "<my-aes-secret-key>"]`
	`[/MailServer <smtp.something.com>`
	`/MailTo <a@b.com>`
	`/MailFrom <x@y.com>`
	`[/MailPort <integer>] [/MailSsl]`
	`[/MailUsername <domain\user> /MailPassword "<pass>"]]`
	`[--All 'Add PackagesFile' parameters are supported here--]`

Get Job	Gets info on an existing job.
	Usage:
	`agent-client.exe Get Job`
	`[/AgentAddress <https://server:port>]`
	`/AgentUsername <user>`
	`/AgentPassword "<pass>"`
	`[/ArgFile <path\agent-client.args.json>]`

Command	Description
	`[/CryptoKey "<my-aes-secret-key>"]` `[/Id <id>]`

Get Status	Gets status info on the agent-service
	Usage:
	`agent-client.exe Get Status [/AgentAddress <https://server:port>]`
	`/AgentUsername <user>`
	`/AgentPassword "<pass>"`
	`[/ArgFile <path\agent-client.args.json>][/CryptoKey "<my-aes-secret-key>"]`

Footnotes:

`*` `/Path` -Specifies the folder where the packages will be created

- Specifying a `/Path` ending in a FlexApp package's filename.vhdx will perform an extend of that existing package.
- Specifying a `/PathUsername` of `"IAM"` will attempt to use the Agent machine's assigned IAM roles to access the S3 bucket.
- There are several ways to use `/Path`, depending on your target storage platform:
  - `/Path "<\\server\share\target-path>" [/PathUsername "<domain\user>" /PathPassword "<pass>"]`
  - `/Path <s3://bucket/folder> /PathUsername <access key> /PathPassword <secret key>`
  - `/Path <s3://bucket/folder> /PathUsername "IAM"`
  - `/Path <az://bucket/folder> /PathUsername <account name> /PathPassword <account key>`
  - `/Path <gs://bucket/folder> /PathPassword "<path\credential.json>"`

`**` `/Installer` -Specifies what should be installed, or executed and captured

- `/Installer` -Supports several file types beyond EXE's and depending on the file extension, are automatically called by the associated interpreter:

    - MSI files will be automatically run via:
      `msiexec.exe /i /qn`.
      Additional Arguments can be passed in via `/InstallerArgs`.
    - Batch scripts (bat,cmd) are run via:
      `cmd.exe /C "installerscript.bat" "</InstallerArgs>"`
    - Windows Script Host files (vbs,vbe,wsf,wsc,js) are run via:
      `cscript.exe /b /nologo "installerscript.vbs" "</InstallerArgs>"`
    - Powershell (ps1) are run via:
      `powershell.exe -ExecutionPolicy Bypass -NoProfile -WindowStyle Hidden -NonInteractive -File "installerscript.ps1" "</InstallerArgs>"`

- `/Installer` -Copies the specified file into the local temp folder before executing it unless `/DontCopyInstallerLocal` is specified.
    - If the file specified is a script, then `/CopyInstallerFolderLocal` will be enabled so that supporting files are included.
    - The "InstallerFolder" is the folder containing the specified `/Installer` file unless otherwise specified using `/InstallerFolder`.
    - `/InstallerFolder` implies `/CopyInstallerFolderLocal`.

- `/Installer` -Supports local disk paths, CIFS shares and http(s) URLs.

Primary-Client.exe Commands

The following describes the available commands for primary-client.exe:

Notes:

- Be sure to wrap all complex passwords like "&()[]{}^=;!'+,`~ my c0mpl3x PW" and paths that include a space in double quotes.
- Double quotes cannot be used in passwords or AES secrets.
- `/InstallerArgs` (if any) must be quoted, be the last parameter, and inner quotes must be escaped with \
- If `/Installer` points to a .msi, then `/InstallerArgs` will automatically be set to `/i "<installer.msi>" /qn` and `/Installer` will implicitly become msiexe.exe, but you can still use `/InstallerArgs` to provide additional args in this scenario.
- If multiple paths are provided to network shares, multiple credentials are provided, and they are to the same server then either only use one set of credentials, or make the second reference to the server be via ip.
- If a PackagesFile contains a parameter already set in DefaultsJSON, PackagesFile wins. If parameter also included on CLI, CLI wins.

Command	Description
Help	Displays the command line options Usage: `primary-client.exe Help`

Add PackagesFile (omit all params to get an empty template file)	Creates or appends to a PackageFile
	Usage: `primary-client.exe Add PackagesFile`
	`/PackagesFile <path\packagesfile.json (or .csv)>`
	`[/CryptoKey "<my-aes-secret-key>"]`
	`/Name "<name>"`
	`/Path <path>` ***SEE BELOW**
	`[/PathUsername <domain\user> /PathPassword "<pass>"]`
	`[/AzureMaximumConcurrency <int>]` -Azure download tuning. Default 64.
	`[/AzureInitialTransferSizeMb <int>]` -Azure download tuning. Default 1.
	`[/AzureMaximumTransferSizeMb <int>]` -Azure download tuning. Default 1.
	`[/CustomStorageUrl <url> (Custom S3 service URL or Azure endpoint suffix)]`

Command	Description
	[/SizeGb <GBs>] -Defaults to 20GB. [/Fixed] -Creates a fixed vhdx instead of the default expandable. [/Test] -Leaves some .bmp files behind to make it easier to see what shortcuts were captured. [/PuAddress <https://pu.server:8000> /PuUsername "<domain\user>" /PuPassword "<pass>"] -Registers new packages in inventory. /Installer <path\installer.exe> **SEE BELOW [/InstallerUsername <domain\user> /InstallerPassword "<pass>"] [/InstallerArgs <args>] [/InstallerExitCode <int>] -Expected installer exit code, else error. Default 0. [/InstallerTimeoutMs <ms>] -How long to wait for the installer to finish, else error. Default 1hr. [/WaitAfterInstallerExitsMs <ms>] -Wait after installer exits before completing capture. Default 0. [/DontCopyInstallerLocal] -Will not copy the installer locally before running the installer. Default false; Does copy. [/CopyInstallerFolderLocal] -Copies the folder containing the installer file locally before executing. Default false except for script installers. [/InstallerFolder <path>] -Change from the default installer path of the folder containing the installer to a custom path to be copied locally before executing the installer. [/NoHCCapture] -Do not use high compatibility capture mode. Default false; Use HC Mode. [/NoSystemRestore] -Do not perform a System Restore rollback at the end of the capture/extend. Default false; use System Restore to rollback. [/AltRestoreCmd <path\file>] -Instead of using System Restore, use a cmd to rollback, implies /NoSystemRestore. [/AltRestoreCmdArgs <args>] [/Installer2 <path\installer.exe> **SEE BELOW [/InstallerArgs2 <args>] [/InstallerExitCode2 <int>]] ... [/Installer10 <path\installer.exe> **SEE BELOW

Command	Description

	`[/InstallerArgs10 <args>]`
	`[/InstallerExitCode10 <int>]]`
	`[/PreActivationScript <path\file>]`
	`[/PostActivationScript <path\file>]`
	`[/PreDeactivationScript <path\file>]`
	`[/PostDeactivationScript <path\file>]`
	`[/NoCallToHome]` -Do not send installer and installer args data to Liquidware.
	`[/PackageVersion <major.minor.build.revision>]` -User specified version to help track package changes.
	`[/DontCreateFlexAppOne]` -Do not automatically create a FlexApp One EXE. Default false; Do create EXE.
	`[/FlexAppOneCliOverride <args>]` -Custom FlexApp One Bundler command line args.
	`[/DontCaptureUserProfileData]` -Do not capture user profile data.
	`[/DontCaptureUserRegistry]` -Don't capture user registry data from HKCU or HKU.
	`[/DontCapture]` -Do not capture at all, just install.
	`[/PackagesXml]` -Write a copy of package info into the specified packages.xml so it can be inventoried using FlexApp Packaging Console in "Offline Mode".
	`[/PuConfiguration <configname> /PuFilter <filtername> [/PuDescription <descr>]]` -Replaces existing package in specified configuration with created package. `/Name` and `/PuFilter` must match existing package assignment, with differing /PackageVersion values.

List PackagesFile (or view the file by hand)	List package information from an existing PackagesFile Usage: `primary-client.exe` -List PackagesFile `/PackagesFile <path\packagesfile.json (or .csv)>`

Remove PackagesFile (or edit the file by hand)	Removes <package name> from an existing PackagesFile
	Usage: `primary-client.exe` -Remove PackagesFile
	`/PackagesFile <path\packagesfile.json (or .csv)>`
	`/Name <package name>`

Clear PackagesFile	Removes ALL entries from an existing packages file Usage: `primary-client.exe` -Clear PackagesFile

Command	Description
(or edit the file by hand)	/PackagesFile <path\packagesfile.json (or .csv)>
Create DefaultsJSON (Optional, omit all params to get an empty template file)	Creates optional DefaultsJSON with default params used by packaging jobs Usage: primary-client.exe -Create DefaultsJSON /DefaultsJSON <path\defaults.json> /CryptoKey "<my-aes-secret-key>"] [/MailServer <smtp.something.com> /MailTo <a@b.com> /MailFrom <x@y.com> [/MailPort <integer>] [/MailSsl] [/MailUsername <domain\user> /MailPassword "<pass>"]] [--All 'Add PackagesFile' parameters are supported here--]
List DefaultsJSON (or view the file by hand)	List parameters from a DefaultsJSON file Usage: primary-client.exe -List DefaultsJSON /DefaultsJSON <path\defaults.json>
Create Packages	Creates the packages listed on the CLI or in the specified file(s) Usage: primary-client.exe -Create Packages [/PrimaryAddress <https://server:9075>] /PrimaryUsername <user> /PrimaryPassword "<pass>" /PackagesFile <path\packagesfile.json (or .csv)> [/DefaultsJSON <path\defaults.json>] [/CryptoKey "<my-aes-secret-key>"] [/MailServer <smtp.something.com> /MailTo <a@b.com> /MailFrom <x@y.com> [/MailPort <integer>] [/MailSsl] [/MailUsername <domain\user> /MailPassword "<pass>"]] [/WaitForDone] [/JobFilter <JobStatus>] -Pending, Running, Successful, Failed, or Cancelled. [/CaptureRetryCount <Int>] -Defaults to number of agents, Zero means no retries. [--All 'Add PackagesFile' parameters are supported here--]
Status Packages	Retrieves the status of Create Packages batch jobs Usage: primary-client.exe -Status Packages

Command	Description
	[/PrimaryAddress <https://server:9075>] /PrimaryUsername <user> /PrimaryPassword "<pass>" [/Id <guid>] [/DefaultsJson <path\defaults.json>] [/CryptoKey "<my-aes-secret-key>"] [/JobFilter <JobStatus>] -Pending, Running, Successful, Failed, or Cancelled.
Wait Packages	Waits for a currently running create packages batch job to complete Usage: primary-client.exe -Wait Packages [/PrimaryAddress <https://server:9075>] /PrimaryUsername <user> /PrimaryPassword "<pass>" /Id <guid> [/DefaultsJson <path\defaults.json>] [/CryptoKey "<my-aes-secret-key>"] [/JobFilter <JobStatus>] -Pending, Running, Successful, Failed, or Cancelled.
Add Agent	Adds a Capture Agent to the Primary Packaging Manager Usage: primary-client.exe -Add Agent [/PrimaryAddress <https://server:9075>] /PrimaryUsername <user> /PrimaryPassword "<pass>" /AgentAddress <https://server:9074> /AgentUsername <user> /AgentPassword "<pass>" [/DefaultsJson <path\defaults.json>] [/CryptoKey "<my-aes-secret-key>"]
Remove Agent	Removes a Capture Agent from the Primary Packaging Manager Usage: primary-client.exe -Remove Agent [/PrimaryAddress <https://server:9075>] /PrimaryUsername <user> /PrimaryPassword "<pass>" /AgentAddress <https://server:9074> [/DefaultsJson <path\defaults.json>] [/CryptoKey "<my-aes-secret-key>"]
List Agent	Lists all Capture Agents currently added to the Primary Packaging Manager Usage: primary-client.exe -List Agent [/PrimaryAddress <https://server:9075>] /PrimaryUsername <user> /PrimaryPassword "<pass>" [/DefaultsJson <path\defaults.json>]

Command	Description
	`[/CryptoKey "<my-aes-secret-key>"]`

Clear Agent	Deletes ALL Capture Agents from the Primary Packaging Manager
	Usage: `primary-client.exe` -Clear Agent
	`[/PrimaryAddress <https://server:9075>]`
	`/PrimaryUsername <user> /PrimaryPassword "<pass>"`
	`[/DefaultsJson <path\defaults.json>]`
	`[/CryptoKey "<my-aes-secret-key>"]`

Footnotes:

`* /Path` -Specifies the folder where the packages will be created

- Specifying a `* /Path` ending in a FlexApp package's filename.vhdx will perform an extend of that existing package.
- Specifying a `* /PathUsername` of `"IAM"` will attempt to use the Agent machine's assigned IAM roles to access the S3 bucket.
- There are several ways to use `* /Path`, depending on your target storage platform:
  - `/Path "<\\server\share\target-path>" [/PathUsername "<domain\user>" /PathPassword "<pass>"]`
  - `/Path <s3://bucket/folder> /PathUsername <access key> /PathPassword <secret key>`
  - `/Path <s3://bucket/folder> /PathUsername "IAM"`
  - `/Path <az://bucket/folder> /PathUsername <account name> /PathPassword <account key>`
  - `/Path <gs://bucket/folder> /PathPassword "<path\credential.json>"`

`** /Installer` -Specifies what should be installed, or executed and captured

- `/Installer` -Supports several file types beyond EXE's and depending on the file extension, are automatically called by the associated interpreter:
  - MSI files will be automatically run via:
    `msiexec.exe /i /qn`.
    Additional Arguments can be passed in via `/InstallerArgs`.

    - Batch scripts (bat,cmd) are run via:
      `cmd.exe /C "installerscript.bat" "</InstallerArgs>"`
    - Windows Script Host files (vbs,vbe,wsf,wsc,js) are run via:
      `cscript.exe /b /nologo "installerscript.vbs" "</InstallerArgs>"`
    - Powershell (ps1) are run via:
      `powershell.exe -ExecutionPolicy Bypass -NoProfile -WindowStyle Hidden -NonInteractive -File "installerscript.ps1" "</InstallerArgs>"`

- `/Installer` -Copies the specified file into the local temp folder before executing it unless `/DontCopyInstallerLocal` is specified.
    - If the file specified is a script, then `/CopyInstallerFolderLocal` will be enabled so that supporting files are included.
    - The "InstallerFolder" is the folder containing the specified `/Installer` file unless otherwise specified using `/InstallerFolder`.
    - `/InstallerFolder` implies `/CopyInstallerFolderLocal`.

- `/Installer` -Supports local disk paths, CIFS shares and http(s) URLs.

Viewing the Automation API Documentation

The FlexApp Packaging Automation API can be used to integrate silent, automatic capturing of FlexApp, and FlexApp One packages into existing automation platforms or scripts.

Scenario 1 - Automated Packaging Queue / Batch Jobs

FPA Primary Package Manager and FPA Package Capture Agent(s) used in a "queue-like" manner. The Primary Package Manager accepts capture job submissions and submits them to available Package Capture Agent(s). Jobs can be submitted in batches or singularly.

1. Enable the API documentation "Swagger" page on the Primary Service and restart it.
2. At an elevated cmd prompt, enter the following:

```
"C:\Program Files (x86)\Liquidware Labs\FlexApp Packaging Automation\Primary-
Service.exe" update /EnableSwagger
               net stop lw-primary-service
            net start lw-primary-service
```

> **Note:** You can now see the available API endpoints by accessing the Primary Service's web port: `https://<primaryName>:9075/`

3. To disable the API Swagger page, re-run the same commands used to enable it, except omit the **/EnableSwagger** argument.

Scenario 2 - Single-instance capture scenarios / No batching

FPA Package Capture Agent used in standalone, single-threaded manner as part of existing automation or script workflows. This can be used in a DEVOPS scenario where an internal application is updated/built regularly. The new builds can automatically, and immediately, be captured as a FlexApp and FlexApp One package.

1. Enable the API documentation "Swagger" page on the Agent Service and restart it.
2. At an elevated cmd prompt, enter the following:

```
"C:\Program Files (x86)\Liquidware Labs\FlexApp Packaging Automation\Agent-
Service.exe" update /EnableSwagger
               net stop lw-agent-service
            net start lw-agent-service
```

Note: You can now see the available API endpoints by accessing the Agent Service's web port: `https://<primaryName>:9075/`

3. To disable the API Swagger page, re-run the same commands used to enable it, except omit the **/EnableSwagger** argument.

Note: For additional information, or to download the latest version of FlexApp Packaging Automation, please refer to the *FlexApp Packaging Automation* documentation.

Getting Help

If you have questions or run into issues while using our software, Liquidware is here to help. Our goal is to provide you with the knowledge, tools, and support you need.

Using Online Resources

Liquidware maintains various kinds of helpful resources on our Customer Support Portal. If you have questions about your product, use these online resources. The Support Portal includes product forums and a searchable knowledge base, as well as the ability to submit a case to the Liquidware Support system on the Liquidware Customer Support Portal. For product documentation, refer to our Liquidware Document Repository.

Contacting Support

If you need to contact our Support staff for technical assistance, log a request on the Liquidware Customer Support Portal. Prior to logging a case you should review these helpful tips:

- Check the Product Documentation included with your Liquidware Product.
- Try to see if the problem is reproducible.
- Check to see if the problem is isolated to one machine or more.
- Note any recent changes to your system and environment.
- Note the version of your Liquidware product and environment details such as operating system, virtualization platform version, etc.