

NCR Aloha RAL (v20.0 and later)

Creating an Aloha Compatible Chocolatey Package

Last Updated: July 22, 2024

Overview

RAL now supports the Chocolatey® (Choco) packages mechanism for deploying and installing integrated applications in the Aloha® least privilege environment. This Choco package mechanism replaces the older proprietary InstallManifest mechanism that was used to configure integrated applications on the terminal devices.

When a Choco package (**.nupkg**) is signed and registered in RAL, it shows up in the RAL configuration, and can be activated for the terminals under RAL's configuration. When the package is active, a package is automatically copied to the terminal and installed using RAL. Likewise, the inactive package is uninstalled from the terminal if it was previously installed using RAL.

What is Chocolatey and why use it?



Reference

See the [Chocolatey package](#) documentation for complete instructions on how to create and maintain Choco packages.

Chocolatey is an industry-standard package management solution. Choco packages are a special form of zip file (**.nupkg**) that contains the installation instructions along with the files to deploy. Chocolatey runs a PowerShell® script to perform the deployment activities so that you have access to powerful Microsoft® Windows® commands. It also contains special operations for **.msi** and other known installer types.

What is the difference between a RAL InstallManifest file and a Choco package?

This section helps you understand the difference between using InstallManifest and Choco packages.

RAL InstallManifest

A RAL InstallManifest is a list of commands that RAL executes on the terminal device. The InstallManifest commands, such as **file** or **regasm**, act on files and also cause the files to be synchronized from the back-of-house (BOH) using RAL.

The InstallManifest encompasses all file synchronization and terminal requirements necessary to run the integrated application. The file references in InstallManifest are referenced from the BOH file, copied to the terminal, and acted on locally. The InstallManifest is only applied once. RAL uses the InstallManifest file time stamp to detect when the file was changed and needs to be reapplied.

Choco packages

The Choco package is a self-contained package intended to be executed locally. All the necessary configuration and deployment instructions are contained in the package, along with the application's files for deployment. Chocolatey is a third-party package management system; therefore, it can be installed directly, outside of the RAL ecosystem.

The package version is part of the package name (by convention). RAL will install the new packages when detected.



Note

This does not prevent someone from generating a valid package with malicious content. We also cannot prevent a malicious user from writing to the RAL registry providing the link to the package.

Converting existing RAL InstallManifest file to Choco package

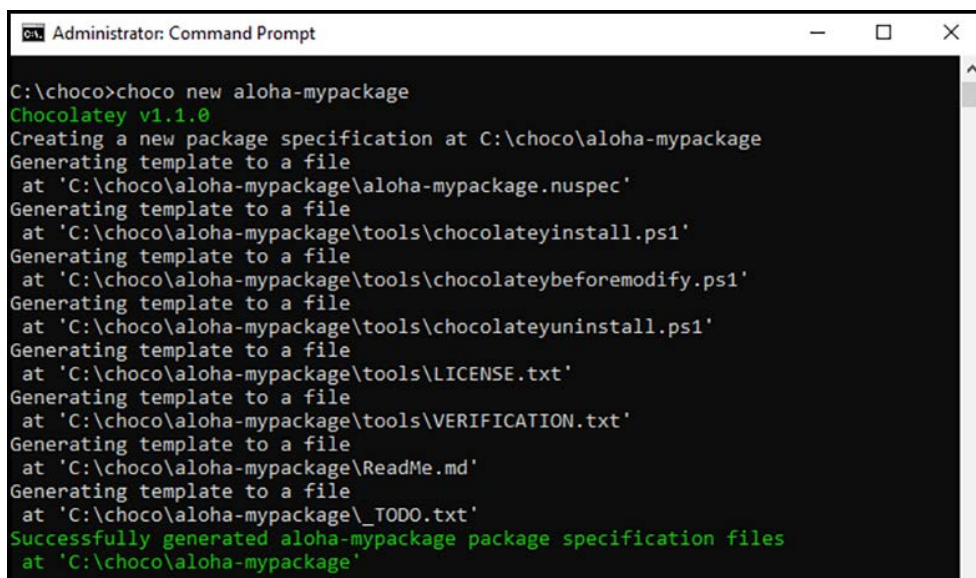
The following steps are required to convert an existing RAL InstallManifest file to a Choco package:

1. Create the initial **Choco package** from the template, **choco new**.
2. Copy the **application files** for deployment into the **Tools** folder.
3. Migrate the InstallManifest commands into PowerShell instructions, inside **tools/chocolateyinstall.ps1**.
4. Include **commands** to deploy the application files into the desired locations.
5. Create the deployable **Choco package**, **choco pack**.

Chocolatey package layout

You do not deploy your package into the public Chocolatey repository as you will deploy the package locally to the BOH machine; then, RAL deploys and installs it on the terminal devices.

The naming convention used is lower case letters with or without a dash. For example, **aloha-mypackage** is an ideal way to represent a Choco package name.



```

Administrator: Command Prompt

C:\choco>choco new aloha-mypackage
Chocolatey v1.1.0
Creating a new package specification at C:\choco\aloha-mypackage
Generating template to a file
  at 'C:\choco\aloha-mypackage\aloha-mypackage.nuspec'
Generating template to a file
  at 'C:\choco\aloha-mypackage\tools\chocolateyinstall.ps1'
Generating template to a file
  at 'C:\choco\aloha-mypackage\tools\chocolateybeforemodify.ps1'
Generating template to a file
  at 'C:\choco\aloha-mypackage\tools\chocolateyuninstall.ps1'
Generating template to a file
  at 'C:\choco\aloha-mypackage\tools\LICENSE.txt'
Generating template to a file
  at 'C:\choco\aloha-mypackage\tools\VERIFICATION.txt'
Generating template to a file
  at 'C:\choco\aloha-mypackage\ReadMe.md'
Generating template to a file
  at 'C:\choco\aloha-mypackage\_TODO.txt'
Successfully generated aloha-mypackage package specification files
at 'C:\choco\aloha-mypackage'
  
```

Figure 1 Choco package command execution

Creating a Choco package using choco new command

1. Open the **PowerShell** or **Windows Command Prompt**.
2. Type **choco new <package-name>**. For example, `choco new aloha-mypackage`.
3. Press **Enter** to execute the command and create the default files for a package. The default files are:

Default file	Description
.nuspec	This file contains the naming and version details for the package.
tools/*.ps1	This is a PowerShell script that Chocolatey uses to perform install or uninstall actions. <ul style="list-style-type: none"> • chocolateyinstall.ps1 - Edit this file to perform the necessary install and configuration commands. • chocolateyuninstall.ps1 - Edit this file to perform specialized uninstall actions (this may not be needed).
tools	The files to be deployed must be included in this folder.

Mapping RAL InstallManifest actions into Choco package commands

We explain the InstallManifest and its corresponding actions with some examples in this section.

Applying folder permissions

RAL now uses the **AlohaUsers** users group to apply permissions to all Aloha users on the system, as required. You can also specify just the terminal user with `$parameters["AlohaUsersGroup"]`.

InstallManifest actions	Example of chocolateyinstall.ps1
<pre><Action="dirwrite">c:\Program Data\MyApplication\Data</ Action></pre>	<pre>\$parameters = Get-PackageParameters \$alohaUsersGroup = \$parameters["AlohaUsersGroup"] \$path = "c:\ProgramData\MyApplication\Data" \$aclEntry = \$alohaUsersGroup, "FullControl", "ContainerInherit, ObjectInherit", "None", "Allow" \$accessRule = New-Object System.Security.AccessControl.FileSystemAccessRule(\$aclE ntry) \$acl = Get-Acl \$path \$acl.SetAccessRule(\$accessRule) \$acl Set-Acl \$path</pre>

Deploy or copy file

You must include **myfile.txt** in the **tools** folder of the package.

InstallManifest actions	Example of chocolateyinstall.ps1
<pre><Action="file" TargetDirectory="c:\local\foh \path\myfile.txt">d:\local\bo h\path\myfile.txt</Action></pre>	<pre>Copy-Item -Path ".\myfile.txt" -Destination "c:\local\foh\path\myfile.txt"</pre>

Installing an msi

You must include **myfile.msi** in the **tools** folder of the package.

InstallManifest actions	Example of chocolateyinstall.ps1
<pre><Action="msi" TargetDirectory="c:\local\foh \path\myfile.msi">d:\local\bo h\path\myfile.msi</Action></pre>	<pre>Copy-Item -Path ".\myfile.msi" -Destination "c:\local\foh\path\myfile.msi" \$packageArgs = @ { PackageName = "My Package Name" SoftwareName = "My Software Name" FileType = "'msi'" File = ".\myfile.msi" SiletArgs = "/quiet /norestart" ValidExitCodes = @(0, 3010, 1641) } Install-ChocolateyInstallPACkage @packageArgs</pre>

Launching a file

You must include **myfile.exe** in the **tools** folder of the package.

InstallManifest actions	Example of chocolateyinstall.ps1
<pre><Action="exe" TargetDirectory="c:\local\foh\path\myfile.exe" CmdLineParameters="param1 param2 param3"> d:\local\boh\path\myfile.exe< /Action></pre>	<pre>& ".\myfile.exe" "param1" "param2" "param3"</pre>

Registering a C# assembly

You must include **myfile.dll** in the **tools** folder of the package. Deploy the file to its intended destination from the **tools** folder and then register it.

InstallManifest actions	Example of chocolateyinstall.ps1
<pre><Action="exe">d:\local\boh\path\myfile.dll</Action></pre>	<pre>Copy-Item -Path ".\myfile.dll" -Destination "c:\local\foh\path\myfile.dll" \$regasm = "C:\Windows\Microsoft.NET\Framework\v4.0.30319\regasm.exe" & \$regasm "c:\local\foh\path\myfile.dll"</pre>

Building a Choco package for distribution

In the **.nuspec** file, update the **<version>__REPLACE__</version>** attribute to apply a default version. With the **ChocolateyInstall.ps1** written and the files for deployment copied into the **tools** folder, you are now ready to create the package. This is done with the **choco pack** command. Even though the **<version>** attribute is changed, make sure you specify the version on the command line, incrementing with each release.

The Choco package command depicted in the following screenshot creates the **aloha-mypackage.1.0.0.0.nupkg** file. Once this file is signed using **nuget sign**, the Choco package is ready for distribution.

```
Administrator: Command Prompt
C:\choco\aloha-mypackage>choco pack aloha-mypackage.nuspec --version=1.0.0.0
Chocolatey v1.1.0
Attempting to build package from 'aloha-mypackage.nuspec'.
Successfully created package 'C:\choco\aloha-mypackage\aloha-mypackage.1.0.0.0.nupkg'
Enjoy using Chocolatey? Explore more amazing features to take your
experience to the next level at
https://chocolatey.org/compare
C:\choco\aloha-mypackage>
```

Figure 2 Choco package command - version

Deploying a Choco package in a RAL environment

An application awaiting RAL to process its package file will write a registry entry to a RAL location on BOH, which communicates to RAL that the package file exists and where to locate it. RAL synchronizes and executes the **choco install** command during the terminal start up, as required. RAL also identifies the package version, which is available as part of the package name, to determine whether the package is the latest version.

The application must add a **REGSZ Value** to the designated RAL registry location, **HKLM\SOFTWARE\Wow6432Node\NCR\RAL\Packages**, which contains the full path of its manifest file.

The **name** must match the internal package name. Below is an example of a BOH where RAL is installed:

```
Key = HKLM\Software\NCR\RAL\Packages
Name = aloha-mypackage
Value = "C:\local\BOH\MyApplication\aloha-mypackage.1.0.0.0.nupkg"
```

RAL-specific Choco package compatibility requirements

RAL installs Choco packages for internal and third-party applications on the FOH terminals. To be fully compatible with RAL, the package must support certain Aloha-specific parameters.

Permission	Description
AlohaUsersGroup	If your package creates users that need access to Aloha folders, you must add the corresponding users to the user group detailed by the AlohaUsersGroup parameter. If your package creates resources (such as data folders, registry entries) that the Aloha application needs to access, the package needs to apply AlohaUsersGroup permissions to these resources.
AlohaTerminalUser	If you need to apply any permissions for the specific Aloha Terminal User, apply that configuration to the account specified by this parameter. This is only for the resources that require specific user permissions rather than the group permissions (specified above). This is not the user running the package installation.
AlohaTerminalType	You may need to take different actions based on the device type. If your package does not need to take any action on a particular terminal type, it should respond to that parameter by simply returning a success code (usually 0). FOH — For order entry devices (iber/QS). AK — For Aloha Kitchen terminal devices. DEDIS — For dedicated server (for the interface server running on a standalone machine).

Important rules for RAL-compatible Choco packages

This section helps you understand some important rules applicable while working with RAL-compatible Choco packages.

- RAL only applies Choco packages to terminal devices (order entry, kitchen, dedicated servers). Choco packages are not executed on the BOH device, even for Interface Servers—any necessary BOH configuration must be part of your BOH installation; however, you may consider building a unified package, which is both compatible with FOH and BOH.
- Do not reboot the device from inside the package. Instead set the package reboot return code **3010**. RAL will reboot the device after running all the packages.
- Implement the **AlohaTerminalType** parameter such that your package can be run on any Aloha device without disrupting its configuration. For instance, if your application applies only to Order Entry terminals, then if you run on an Aloha Kitchen device, it must exit as completed status. (If it exits with failure status, then the corresponding action will be retried on every RAL terminal start up).
- Aloha runs under the terminal auto-log on user, which RAL manages. The specific user name must be unique for each terminal device. Since the user name is configurable, the exact user name will not be known when you create your package; therefore, it cannot be hard coded. If you need to apply additional permissions to your application, use the **AlohaTerminalUser** parameter to obtain the name of the exact terminal user account.
- Add the Aloha users in to the **AlohaUsersGroup** to manage the terminal user permissions. This is to grant permission to the Aloha folders. This is more efficient than applying Aloha permissions to every necessary user. If your application has a separate user account (for instance, if you are installing a Windows service) it should be added to the **AlohaUsersGroup**.
- If your product needs files in the **Aloha\Bin** folder for the terminal, then the BOH installer (or package) must put these files in the **Aloha\Bin** folder on the BOH machine. This is required because RAL purges the terminal **Aloha\Bin** folder to match the BOH **Aloha\Bin** folder. This means you may need to install the package on the BOH for your product, even if there is no direct BOH functionality.

RAL-compatible chocolateyInstall.ps1 template

```
$packageName = $env:chocolateyPackageName
$packageVersion = $env:chocolateyPackageVersion
$toolsPath = Split-Path $MyInvocation.MyCommand.Definition
$parameters = Get-PackageParameters
try
{
    $termType = $parameters["AlohaTerminalType"]
}
catch
{
    exit 1
}
try
{
    $alohaUsersGroup = $parameters["AlohaUsersGroup"]
}
catch
{}
try
{
    $alohaTerminalUser = $parameters["AlohaTerminalUser"]
}
catch
{}
```