# **Introduction to MSIX**

MSIX is a modern app packaging technology that provides a cleaner and more reliable experience for app installation. To learn more about MSIX as a packaging format and all its benefits, check out the [MSIX Overview documentation](https://learn.microsoft.com/en-us/windows/msix/).

This guide will be helpful to you if you have existing installers that are either no longer in development, or you don't own the source code. If you do own the source code, check out the documentation on [how to generate an MSIX from source code](https://learn.microsoft.com/en-us/windows/msix/desktop/source-code-overview).

This documentation will show you how to use the MSIX Packaging Tool to create an MSIX package from any installer you have. Once you've got an MSIX, we've also got information on how to fix runtime issues using the Package Support Framework.

# **Using the MSIX Packaging tool**

The MSIX Packaging Tool enables you to repackage your existing desktop applications to the MSIX format. It offers both an interactive UI and a command line for conversions and gives you the ability to convert an application without having the source code.

**Step 1: Create a Hyper-V environment**

When preparing to convert your application to MSIX, we recommend that you get a [clean environment](https://learn.microsoft.com/en-us/windows/msix/packaging-tool/prepare-your-environment). It ensures the capture process is optimal during package conversion. We recommend you create a virtual environment for MSIX packaging projects using the [Hyper-V Quick Create](https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-start/quick-create-virtual-machine) feature. This feature is available starting in Windows 10, version 1709. In case this is not enabled, it can be activated by using ‘Turn Windows features on or off’.

To get started, type 'Hyper-V Quick Create' in your Start menu, select MSIX Packaging Tool Environment, and click Create Virtual Machine. The MSIX Packaging Tool Environment is a custom Windows 10 evaluation build (version 1909) that includes the MSIX Packaging Tool and other pre-requisites so that you can get started quickly with limited setup tasks.

**Step 2: Update MSIX Packaging Tool to the latest version**

The MSIX Packaging Tool come pre-installed with the Hyper-V MSIX Packaging Tool Environment, but it must be updated to the latest version. MSIX Packaging Tool is [available on the Microsoft Store](https://www.microsoft.com/p/msix-packaging-tool/9n5lw3jbcxkf). In case you are working in a disconnected environment, or cannot access Microsoft Store, you can [download an offline copy](https://learn.microsoft.com/en-us/windows/msix/packaging-tool/disconnected-environment#get-the-msix-packaging-tool) of MSIX Packaging Tool.

**Step 3: Start using the MSIX Packaging Tool**

Let us look at an example of converting a sample desktop application (PowerBI desktop in this case) to a MSIX package using MSIX Packaging Tool.

Open the MSIX Packaging Tool, and click on Application Package to create a new package for your application:

Logo

Description automatically generated

Select the first option – ‘Create Package on this computer’, and then click ‘Next’.

Graphical user interface, text, application, email

Description automatically generated

Click ‘Next’ once your MSIX Packaging Tool driver is installed and Windows updates disabled (if applicable). In case the driver installation fails, please refer to the ‘Known Issues’ section below for possible workarounds. Also, please restart your VM if you get a ‘Pending Reboot’ message in the Additional preparations section.

Graphical user interface, text, application, email

Description automatically generated

Browse to select your (legacy) application package, select your signing preferences, and then click on ‘Next’.

Graphical user interface, text, application

Description automatically generated

Fill in the details on the next screen, and then proceed to the Installation step by clicking on the ‘Next’ button.

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Proceed with the Installation  
Graphical user interface, text, application

Description automatically generated

Wait for the system to restart post installation, and then click on ‘Next’.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Click ‘Next’ and then select ‘Yes, move on’ button to continue packaging the application.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Click ‘Next’ to move on to the final step. Browse to the save location, and then click on ‘Create’.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Your MSIX package is now created and ready to be distributed.

# **Known Issues**

**MSIX Packaging Tool driver**

The MSIX Packaging Tool driver is delivered as a Feature on Demand (FoD) package from Windows Update. It will fail to install if the Windows Update service is disabled or if Windows Insider flight ring settings do not match the OS build of the computer.

If you are running into problems acquiring the driver, or you are working in an offline environment, you can find links to download and install the driver [here](https://learn.microsoft.com/en-us/windows/msix/packaging-tool/disconnected-environment#get-the-msix-packaging-tool-driver).

If you have downloaded the driver and are running into issues during your package conversion, you can refer to [this documentation](https://learn.microsoft.com/en-us/windows/msix/packaging-tool/tool-known-issues#msix-packaging-tool-driver).

**Frameworks and drivers**

If the app requires a framework, make sure the framework is installed during the monitoring phase of the conversion. Go through the logs to ensure this is happening. If your app requires a driver to install, you need to evaluate whether this is required for your app to run properly. Currently MSIX does not support driver installation.

You can read about more [Known Issues and some tips for troubleshooting here](https://learn.microsoft.com/en-us/windows/msix/packaging-tool/tool-known-issues).

# **Additional Resources**

1. Demo videos:
   * [MSIX Packaging Tool](https://microsoftapc-my.sharepoint.com/:v:/g/personal/anbanerjee_microsoft_com/EVV0n0S1CPxKvQuwbIukoycB0c2Sg2P2ybIuHAi2eHoQOw?e=heCUDq)
   * [Package Support Framework](https://microsoftapc-my.sharepoint.com/:v:/g/personal/anbanerjee_microsoft_com/EW5iuGDctrRHt6Ea7eErbuEBLWcTDOvfFH7-3b6I0fwrrQ?e=4srVUe)
2. Submit Feedback and Ideas:
   * [Join the MSIX Tech Community](https://techcommunity.microsoft.com/t5/msix/bd-p/MSIX-Discussions)
   * [Submit your MSIX Ideas and feature requests](https://techcommunity.microsoft.com/t5/msix-feedback/idb-p/MSIXIdeas)
   * [Submit Feedback on the MSIX Packaging Tool](https://support.microsoft.com/en-us/windows/send-feedback-to-microsoft-with-the-feedback-hub-app-f59187f8-8739-22d6-ba93-f66612949332)
3. [Advanced step-by-step guide to learn more about the MSIX Packaging Tool and Package Support Framework](https://github.com/microsoft/MSIX-Labs)

# **For Advanced users: Applying runtime fixes with the Packaging Support Framework**

The [Package Support Framework](https://learn.microsoft.com/en-us/windows/msix/psf/package-support-framework-overview) library helps to find and/or fix the runtime errors for the existing Desktop Application without modifying or rebuilding the Application code. Most Desktop Applications usually read or write files to the local directory or from a predefined folder, but a Desktop Application will fail to read or write the file in the MSIX container predefined folder because of restrictions imposed by the container. Including the Package Support Framework helps in converting the traditional Windows installer package into an MSIX package so that in many cases the Desktop Application can be run within the container without any code changes. The Package Support Framework provides multiple types of fixups to fix the Application runtime errors, such as the File Redirection Fixup, RegLegacyFixup, DynamicDLLFixup, and EnvVarFixup.

Here are some of the most used fixups available:

1. **File Redirection Fixup** – This is used to redirect attempts to write or read data in a location that isn't accessible from the application as it runs in a MSIX container
2. **RegLegacy Fixup** – This is used to modify registry calls to an acceptable form for being in the container
3. **EnvVar Fixup** – This is used to provide support for package-level environment variables

Now let us see an example of PSF in action. [Assume you have an application called ‘MyEmployees’ which was packaged in MSIX format and installed on your system](https://github.com/microsoft/MSIX-Labs/releases/tag/1.1). The application is running fine from the user’s point-of-view, but one of the features isn’t working as expected. The issue is diagnosed to the fact that when the application is installed as an MSIX app, the log file isn’t getting created inside the installation folder. This is because the application is unable to write inside the installation folder. To solve this problem, we need to add the Package Support Framework to the MSIX package, so that the file is created in a folder where the application has ‘Write’ access.

The first step is to unpack the MSIX package which contains the ‘MyEmployees’ application, so that the Package Support Framework can be injected into it.

Start by Right clicking on the package, and editing the package in MSIX Packaging Tool.

Graphical user interface, application

Description automatically generated

Click on ’Package files’ on the left-hand pane.

Graphical user interface, text, application

Description automatically generated

Now download the [latest release of PSF](https://www.nuget.org/packages/Microsoft.PackageSupportFramework/) from NuGet. Right click on the Package, and add the following PSF files into the package folder.

Graphical user interface, table

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Now, create a config.json with the modified working directory and the fixups mentioned. Add that to the package as well.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Now, click on the ‘Package Information’ tab on the left hand pane, scroll to the bottom to the Manifest file section, and click on ‘Open File’ button to edit the AppxManifest to change the entry point of the application.

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Sign the package using your preferred method (.pfx certificate in this case).

Graphical user interface, application

Description automatically generated

Now, save the file and click on ‘Save’ button to recreate a MSIX package after the changes in the package.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Install the created package.

Graphical user interface, text, application, Teams

Description automatically generated

You can see that the log file is now getting created in the VFS of the application.

Graphical user interface, text, application, email

Description automatically generated