Basler Cameras

PYLON DEPLOYMENT GUIDE

Document Number: AW001362
Version: 07 Language: 000 (English)
Release Date: 02 July 2019
Software Version: 6.x

BASLER
the power of sight

Contacting Basler Support Worldwide

Europe, Middle East, Africa

Basler AG
An der Strusbek 60–62
22926 Ahrensburg
Germany

Tel. +49 4102 463 515
Fax +49 4102 463 599

support.europe@baslerweb.com

The Americas

Basler, Inc.
855 Springdale Drive, Suite 203
Exton, PA 19341
USA

Tel. +1 610 280 0171
Fax +1 610 280 7608

support.usa@baslerweb.com

Asia-Pacific

Basler Asia Pte. Ltd.
35 Marsiling Industrial Estate Road 3
#05–06
Singapore 739257

Tel. +65 6367 1355
Fax +65 6367 1255

support.asia@baslerweb.com

www.baslerweb.com

Table of Contents

1 Introduction

To deploy pylon software and camera drivers, two options are available:

- **Copy deployment**: Deploy the necessary files simply by copying them. This is the recommended option if you want to deploy pylon as part of your own application. Camera drivers must be installed separately using **.msi** installers.
  For more information, see Chapter 3 on page 3.
- **pylon runtime redistributable package** (also known as "runtime installer"): Deploy the necessary files and drivers using an installation program. This is the recommended option if you want to perform an unattended or silent installation of the pylon software.
  For more information, see Chapter 2 on page 2.

For the advantages and disadvantages of each option, see Table 1.

Deployment Option	Advantages	Disadvantages
Copy Deployment	▪ Side-by-side installation: Files installed via copy deployment can run side by side with other pylon installations. Newer pylon versions do not conflict with the deployed version. ▪ If desired, pylon can be deployed without shortcuts in the Windows start menu or on the desktop. ▪ Full control over which files should be installed and which should not.	▪ You must choose each installation file manually and individually. ▪ No automatic installation routine. You must provide and maintain your own installer.
pylon Runtime Redistributable Package	▪ Installation files are bundled in feature sets, e.g., "USB_Runtime". You don’t have to deal with individual installation files.	▪ No side-by-side installation. If a newer pylon version is installed, it replaces the existing runtime version. If the newer version is not binary compatible, this can render your application unusable.

Table 1: Advantages and Disadvantages of pylon Deployment Options

2 Runtime Redistributable Package

The pylon runtime redistributable package allows you to deploy the pylon files and drivers by running an installation program.

This is the recommended option if you want to perform an unattended or silent installation of the pylon software.

You can choose whether to install support for all camera interfaces (GigE, USB 3.0, CXP 2.0, Camera Link) or whether to pick individual interfaces as required. If you want to install support only for selected camera interfaces, see "Configuring and Customizing the Installation".

The redistributable package includes 32-bit and 64-bit files.

If the installer detects a 64-bit operating system on the target machine, the installer automatically installs the 64-bit **and** the 32-bit files. Otherwise, it installs only the 32-bit files.

Configuring and Customizing the Installation

The runtime redistributable package (**.exe** file) accepts command line switches to customize and configure the installation.

The options include, e.g.,

- running a "silent installation" without displaying any messages or windows during installation
- installing only specific components (e.g., only the C++ runtime files, the USB runtime files, the USB camera driver, and the pylon Viewer)
- uninstalling specific components.

To access information about the available switches, launch the runtime redistributable package (**.exe** file) from the command prompt with the **/help** or **/?** switch.

Example:

```
"Basler pylon Runtime 6.x.x.xxxx.exe" /help
```

A help window will appear, giving information about the available command line switches.

3 Copy Deployment

3.1 Using Copy Deployment

Copy deployment, also known as XCopy deployment, allows you to deploy the necessary files simply by copying them to the target computer. Because there is no automatic installation routine, you must provide and maintain your own installation routine to automate the process.

This is the recommended option if you want to deploy pylon as part of your own application.

When using copy deployment, do not deploy the files to a target directory that is included in the Windows PATH environment variable and do not extend the PATH variable to include the target directory.

Otherwise, the deployed version may conflict with future pylon versions you install on the target computer.

To deploy pylon via copy deployment:

- Set up your installation routine to
  - copy the required pylon files to the target computer.
    For a list of the pylon files, see Section 3.2.1 on page 4.
  - execute the Visual C++ Redistributable Package installer.
    For more information, see Section 3.2.2 on page 8.
  - execute the appropriate camera driver installer (**.msi** file).
    For a list of pylon camera driver installers, see Section 3.2.3 on page 10.
- Make sure your application's executable file(s) can find the pylon DLLs on the target computer.
  For more information, see Section 3.3 on page 11.

These steps and sub-steps can be carried out in any order.

3.2 Choosing Files for Copy Deployment

3.2.1 pylon Files

To deploy pylon using copy deployment, a set of **.dll** and **.zip** files must be copied to the target computer. The file names and the number of files vary depending on the camera interface(s) that you want your application to support (GigE Vision, USB 3.0 Vision, Camera Link, CoaXPress 2.0, or multiple interfaces).

The files required for each interface are shown in Table 2. All files are available in 32-bit and 64-bit versions, with identical file names. If your application is available in both 32-bit and 64-bit versions, you must provide each file twice.

To obtain the files, install the pylon Camera Software Suite on your computer. During installation, select the "Developer" profile and choose the desired interface(s).

After the installation, you can find the files in the **\Runtime\x64** and **\Runtime\Win32** subdirectories of the pylon installation directory, unless otherwise noted. For details, see the "Notes" column in Table 2.

To determine whether your application requires additional files that are not listed in Table 2, you can run the Microsoft Dependency Walker tool (**depends.exe**), as described in the "Understanding the Dependencies of a Visual C++ Application" topic (https://docs.microsoft.com/en-us/cpp/windows/understanding-the-dependencies-of-a-visual-cpp-application?view=vs-2019).

File Name	GigE	USB 3.0	CXP 2.0	Camera Link	Notes
PylonBase_v6_0.dll	x	x		x	
GCBase_MD_VC141_v3_1_Basler_pylon.dll	x	x		x	
GenApi_MD_VC141_v3_1_Basler_pylon.dll	x	x		x	
log4cpp_MD_VC141_v3_1_Basler_pylon.dll	x	x		x	
Log_MD_VC141_v3_1_Basler_pylon.dll	x	x		x	
NodeMapData_MD_VC141_v3_1_Basler_pylon.dll	x	x		x	
XmlParser_MD_VC141_v3_1_Basler_pylon.dll	x	x		x	
MathParser_MD_VC141_v3_1_Basler_pylon.dll	x	x		x	
PylonGigE_v6_0_TL.dll	x				
gxapi_v11.dll	x				

Table 2: Files Required for Copy Deployment

File Name	GigE	USB 3.0	CXP 2.0	Camera Link	Notes

ProducerGEV.cti	x				Only required in combination with a GenTL consumer.
					To use it, the environment variable GENICAM_GENTL32_PATH or GENICAM_GENTL64_PATH resp. needs to be extended.
					This works similarly to extending the Windows PATH variable. For example, set
					`GENICAM_GENTL32_PATH`
					to
					`GENICAM_GENTL32_PATH=%GENICAM_GENTL32_PATH%;<copy_dir>/Runtime/Win32`

PylonUsb_v6_0_TL.dll		x			
uxapi_v11.dll		x			

ProducerU3V.cti		x			Only required in combination with a GenTL consumer.
					To use it, the environment variable GENICAM_GENTL32_PATH or GENICAM_GENTL64_PATH resp. needs to be extended.
					This works similarly to extending the Windows PATH variable. For example, set
					`GENICAM_GENTL32_PATH`
					to
					`GENICAM_GENTL32_PATH=%GENICAM_GENTL32_PATH%;<copy_dir>/Runtime/Win32`

pylonCXP or **pylonCXP** folders			x		The entire **pylonCXP** folders in the **Runtime\Win32\pylonCXP** or **Runtime\x64\pylonCXP** subdirectories of the pylon installation directory are required.
					To use it, the environment variable GENICAM_GENTL32_PATH or GENICAM_GENTL64_PATH resp. needs to be extended.
					This works similarly to extending the Windows PATH variable. For example, set
					`GENICAM_GENTL32_PATH`
					to
					`GENICAM_GENTL32_PATH=%GENICAM_GENTL32_PATH%;<copy_dir>\pylonCXP\bin`

PylonGtc_v6_0_TL.dll			x		
PylonCLSer_v6_0_TL.dll				x	

Table 2: Files Required for Copy Deployment

File Name	GigE	USB 3.0	CXP 2.0	Camera Link	Notes
CLAllSerial_MD_VC141_v3_1_Basler_pylon.dll				x	
CLProtocol_MD_VC141_v3_1_Basler_pylon.dll				x	
CLSerCOM.dll				x	Usually, you should include the CLSer*XYZ*.dll provided by the manufacturer of your Camera Link frame grabber. The CLSerCOM.dll file can only be used to establish a Camera Link connection via standard COM ports.
Basler_CameraLink.zip				x	Include this file if the camera description file cannot be downloaded from the camera device you are using.
BaslerCLProtocol.dll				x	This file must be copied from the **Runtime\CLProtocol\Win64_x64** subdirectory of the pylon installation directory. **Place this file in a separate directory.** The 32-bit file must be placed in a subdirectory named **Win32_i86**. The 64-bit file must be placed in a subdirectory named **Win64_x64**. **Example:** If you have set up your installation routine to copy the 32-bit pylon DLLs to **C:\Program Files\Grab\dll\pylon32**, the file must be copied to **C:\Program Files\Grab\dll\pylon32\Win32_i86**. Also, the GenICam path variable GENICAM_CLPROTOCOL needs to be extended. This works similarly to extending the Windows PATH variable. For example, change `GENICAM_CLPROTOCOL=C:\Program Files\OtherVendor\CLProtocol` to `GENICAM_CLPROTOCOL=C:\Program Files\OtherVendor\CLProtocol; C:\Program Files\MyApp\CLProtocol`
PylonC_v6_0.dll	o	o	o	o	**Optional:** Include this DLL if your application uses the pylon C API or the pylon C .NET API.

Table 2: Files Required for Copy Deployment

File Name	GigE	USB 3.0	CXP 2.0	Camera Link	Notes
Basler.Pylon.dll	o	o	o	o	**Optional:** Include this DLL if your application uses the pylon .NET API. This file must be copied from the **Development\Assemblies** subdirectory of the pylon installation directory.
PylonC.NET.dll	o	o	o	o	**Optional:** Include this DLL if your application uses the pylon C .NET API. This file must be copied from the **Development\Assemblies** subdirectory of the pylon installation directory.
PylonGUI_v6_0.dll	o	o	o	o	**Optional:** Include this file if your application uses the pylon Image Window feature.
PylonUtility_v6_0.dll	o	o	o	o	**Optional:** Include this file if your application uses the pylon Image Handling Support features, e.g., the Image Format Converter.
x = Required file, o = Optional file					

Table 2: Files Required for Copy Deployment

3.2.2 Visual C++ Runtime Files

The pylon DLLs depend on the following Visual C++ runtime files:

File Name	GigE	USB 3.0	CXP 2.0	Camera Link	Notes
vcruntime140.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
msvcp140.dll	x	x	x	x	C run-time library, standard C++ library support (version: Visual Studio 2015 or 2017)
ucrtbase.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-core-file-l1-2-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-core-file-l2-1-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-core-localization-l1-2-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-core-processthreads-l1-1-1.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-core-synch-l1-2-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-core-timezone-l1-1-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-crt-convert-l1-1-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-crt-environment-l1-1-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-crt-filesystem-l1-1-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-crt-heap-l1-1-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-crt-locale-l1-1-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-crt-math-l1-1-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-crt-multibyte-l1-1-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-crt-runtime-l1-1-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-crt-stdio-l1-1-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)

Table 3: Visual C++ Runtime Files

File Name	GigE	USB 3.0	CXP 2.0	Camera Link	Notes
api-ms-win-crt-string-l1-1-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-crt-time-l1-1-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)
api-ms-win-crt-utility-l1-1-0.dll	x	x	x	x	C runtime (CRT) library (version: Visual Studio 2015 or 2017)

Table 3: Visual C++ Runtime Files

These files **must always** be deployed together with the pylon DLLs. The files are available in 32-bit and 64-bit versions, with identical file names. If your application is available in both 32-bit and 64-bit versions, you must provide each file twice.

To do so, two options are available:

- Run the Visual C++ Redistributable Package for Visual Studio 2015 or 2017, provided by Microsoft. This is the recommended method. The installer will automatically copy the required files to the respective Windows system directory (`%windir%\System32` or `%windir%\SysWOW64`).
  You can download the package from the Microsoft website:
  2015 version: https://www.microsoft.com/en-us/download/details.aspx?id=48145
  2017 version: https://visualstudio.microsoft.com/vs/older-downloads/. Expand the **Redistributables and Build Tools** section and select the desired operating system version of the redistributable package.
- Manually copy the files listed in Table 3 to the same target directory as the pylon DLLs. You can use this deployment method to enable installation by users who don't have administrator rights, or for applications that can be run from a network share.

For more information, see the "Choosing a Deployment Method" topic on the Microsoft Docs websites:
https://docs.microsoft.com/en-us/cpp/windows/choosing-a-deployment-method?view=vs-2019
https://docs.microsoft.com/en-us/cpp/windows/universal-crt-deployment?view=vs-2019

3.2.3 pylon Camera Driver Packages

Camera drivers must be installed separately using **.msi** packages.

To install the camera drivers on the target computer, set up your installation routine to execute the corresponding **.msi** file.

The installer will automatically install the driver without displaying any messages or windows during its progress ("silent installation").

The **.msi** packages for each interface are shown in Table 4. If you want your application to support multiple camera interfaces, execute the respective **.msi** files one after the other.

To obtain the files, install the pylon Camera Software Suite on your computer. During installation, select the "Developer" profile and choose the desired interface(s).

After the installation, you can find the files in the **\Development\Redist\Drivers** subdirectory of the pylon installation directory.

- All camera driver packages include 32-bit and 64-bit files. If the installer detects a 64-bit operating system on the target machine, the installer automatically installs the 64-bit drivers. Otherwise, it installs the 32-bit drivers.
- Basler recommends executing the **.msi** file(s) via the Windows Installer Tool, **Msiexec.exe**.

File Name (*)	GigE	USB 3.0	CXP 2.0	Camera Link	Notes
pylon_GigE_Performance_Driver.msi	x				Install this driver if you are using the GigE Performance Driver with a supported Intel Network Adapter.
pylon_GigE_Filter_Driver.msi	x				Install this driver if you are not using the GigE Performance Driver.
pylon_USB_Camera_Driver.msi		x			
pylon_CXP_Driver.msi			x		

Table 4: pylon Camera Driver Packages

3.3 Locating the pylon DLLs

To deploy pylon via copy deployment, you must make sure your application's executable file(s) can find the required pylon DLLs on the target computer. Otherwise, your application won't start or issue an error message.

Enabling your application to find the pylon DLLs can be done in several ways:

- Place the pylon DLLs in the application's working directory.
  For more information, see Section 3.3.1 on page 13.
- Register the application path in the Windows registry.
  For more information, see Section 3.3.2 on page 14.
- Run the application using a batch file.
  For more information, see Section 3.3.3 on page 15.
- Use the SetDllDirectory function in the source code of your application.
  For more information, see Section 3.3.4 on page 17.
- Create assembly manifests and embed them in your application project.
  For more information, see Section 3.3.5 on page 20.

See Fig. 1 for a flowchart that helps you choose the right method.

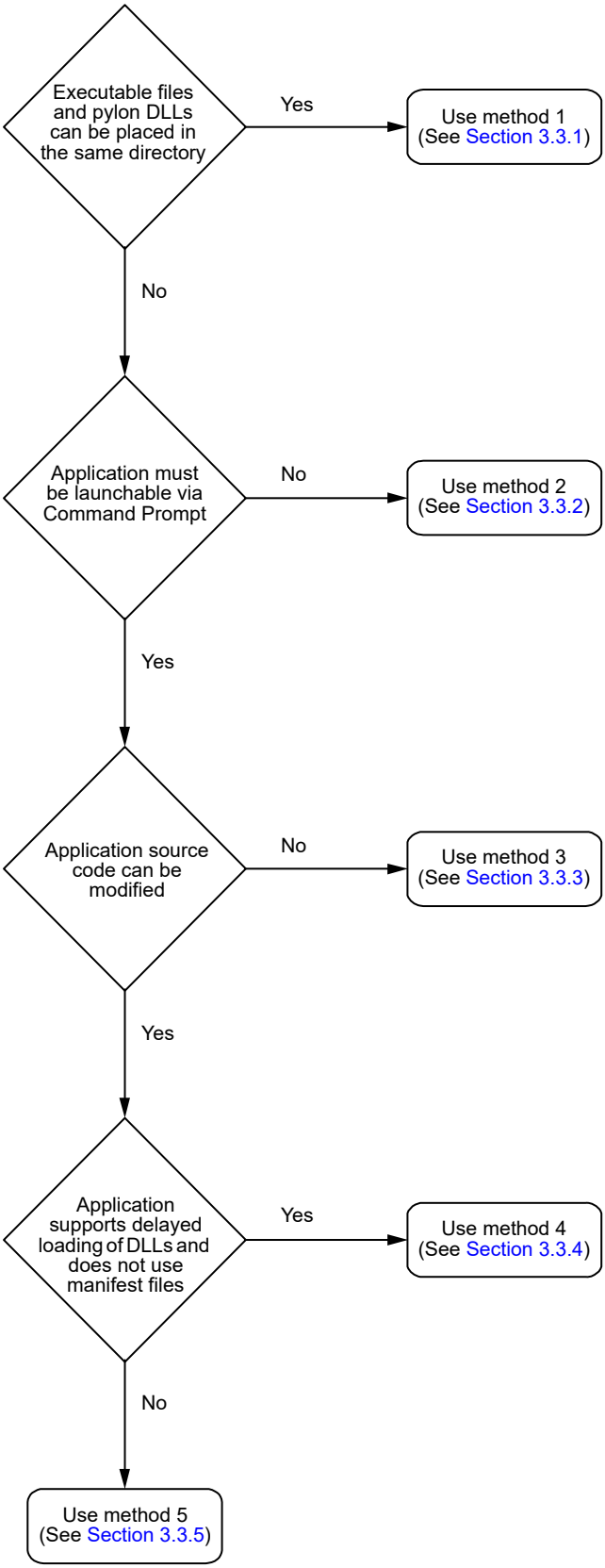


Fig. 1: Flowchart for Choosing the Right Method to Provide pylon DLLs

3.3.1 Method 1: Placing the pylon DLLs in the Application's Working Directory

The easiest copy deployment method is to set up your installation routine to copy the required pylon DLLs to the same target directory as your application's executable files. This enables your application to find the pylon DLLs automatically.

Because this is also the most reliable method, you should always consider this option first.

This method will **not work**

- if you copy the pylon DLLs and your application's DLLs to a common DLL directory and copy your application's executable files to a different directory. The pylon DLLs and your application's executable files must reside in the same directory.
- if your application is available in 32-bit and 64-bit versions and if the 32-bit and 64-bit executable files are copied to the same target directory. You can't copy all files to the same directory as the depending 32-bit and 64-bit pylon DLLs share the same file names.

3.3.2 Method 2: Registering the Application Path in the Windows Registry

To point your application to the pylon DLLs, you can register the application path ("AppPath") in the Windows registry.

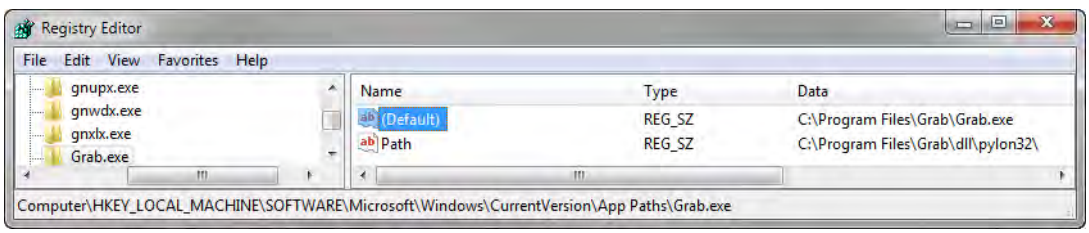
This is the recommended method if your application's executable file(s) and the dependent pylon DLLs can't be placed in the same directory.

If you use this method, your application will not be launchable via Windows Command Prompt. For methods that allow launching the application via Command Prompt, see Section 3.3.3 on page 15, Section 3.3.4 on page 17, and Section 3.3.5 on page 20.

To deploy the pylon DLLs by registering the application path:

1. Set up your installation routine to copy the required pylon DLLs. If your application is available in 32-bit and 64-bit versions, the 32-bit pylon DLLs must be copied to one target directory and the 64-bit pylon DLLs to another.
2. Set up your installation routine to add or change the following registry entries:
   a. Add the subkey `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\`*`file.exe`* where *`file.exe`* must be replaced by the name of your application's executable file, e.g., Grab.exe.
   b. Set the value of the subkey's **(Default)** entry to the path of the application's executable file, e.g., `C:\Program Files\Grab\Grab.exe`.
   c. Add a **Path** entry and set its value to the path of the directory containing the pylon DLLs, e.g., `C:\Program Files\Grab\dll\pylon32\`.

   In the Windows Registry Editor, the result should look like this (sample values):



3. If your application is available in 32-bit and 64-bit versions, repeat step 2 for the application's 64-bit executable file and the directory containing the 64-bit pylon DLLs. For example, add a second subkey `Grab64.exe` with the entries `C:\Program Files\Grab\Grab64.exe` and `C:\Program Files\Grab\dll\pylon64\`.

3.3.3 Method 3: Running the Application Using a Batch File

To point your application to the pylon DLLs, you can write a batch file that will temporarily add the pylon DLL directories to the Windows PATH environment variable and then launch the application.

This is the recommended method if

- your application's executable file(s) and the dependent pylon DLLs can't be placed in the same directory (see Section 3.3.1 on page 13) and
- your application must be launchable via Command Prompt and
- you can't modify the source code of your application.

To deploy the pylon DLLs using a batch file:

1. Set up your installation routine to copy the required pylon DLLs. If your application is available in 32-bit and 64-bit versions, the 32-bit pylon DLLs must be copied to one target directory and the 64-bit pylon DLLs to another.
2. Open a text editor, e.g., Notepad.
3. If your application is available
   - **only** in a 32-bit version or **only** in a 64-bit version, add the following lines:

     ```
     @ECHO OFF
     SET relPath=DllDir
     SET PATH=%~dp0%relPath%;%PATH%
     SET app=%~dp0%~n0.exe
     "%app%"
     ```

   - in **both** 32-bit and 64-bit versions, add the following lines:

     ```
     @ECHO OFF
     SET relPathWin32=DllDir32
     SET relPathX64=DllDir64
     SET PATH=%~dp0%relPathWin32%;%~dp0%relPathX64%;%PATH%
     SET app=%~dp0%~n0.exe
     "%app%"
     ```

4. Replace the *DllDir*, *DllDir32*, and/or *DllDir64* entries by the paths to the pylon DLL directories. The paths must be relative to the directory containing the application's executable files.
   Example: Assuming the following target directories (sample values):
   - Executable files: **C:\Program Files\Grab**
   - 32-bit pylon DLLs: **C:\Program Files\Grab\dll\pylon32**
   - 64-bit pylon DLLs: **C:\Program Files\Grab\dll\pylon64**

   In this case, you must set up the batch file like this:

```
@ECHO OFF
SET relPathWin32=dll\pylon32
SET relPathX64=dll\pylon64
SET PATH=%~dp0%relPathWin32%;%~dp0%relPathX64%;%PATH%
SET app=%~dp0%~n0.exe
"%app%"
```

5. If your application is available
   - **only** in a 32-bit version or **only** in a 64-bit version, save the batch file with the same file name as your application's executable file and with a **.cmd** file extension.
     Example: If your application's executable file is named Grab.exe, save the batch file as Grab.cmd.
   - in **both** 32-bit and 64-bit versions, save the batch file twice: The first time with the file name of your 32-bit executable file, and the second time with the file name of your 64-bit executable file. The contents of the batch file can remain exactly the same.
     Example: If your application's executable files are Grab32.exe and Grab64.exe, save the batch file as Grab32.cmd and Grab64.cmd.
6. Set up your installation routine to copy the batch file(s) to the same directory as your application's executable file(s).
7. Make sure the user of your application launches the application via the batch file and not via the executable file (e.g., by creating desktop shortcuts to Grab32.cmd and Grab64.cmd).

3.3.4 Method 4: Using SetDllDirectory in the Source Code

To point your application to the pylon DLLs, you can use the SetDllDirectory function (C++ only) in the source code of your application.

This is the recommended method if

- your application's executable file(s) and the dependent pylon DLLs can't be placed in the same directory (see Section 3.3.1 on page 13) and
- your application must be launchable via Command Prompt and
- you can modify the source code of your application.

To use SetDllDirectory, the pylon DLLs must be "delay loaded". Otherwise, the application may search for the pylon DLLs too soon, i.e., before reaching the SetDllDirectory function call, and exit with an error message.

If delay loading is known to cause problems in your application, you should use assembly manifests instead of calling SetDllDirectory. Also, if your application already uses assembly manifests, editing the existing manifest files is generally an easier method. For more information, see Section 3.3.5 on page 20.

To deploy the pylon DLLs using the SetDllDirectory function:

1. Set up your installation routine to copy the required pylon DLLs. If your application is available in 32-bit and 64-bit versions, the 32-bit pylon DLLs must be copied to one target directory and the 64-bit pylon DLLs to another.
2. Open your application project in an IDE, e.g., Visual Studio.
3. At the beginning of your application's source code, add the following preprocessor directives:

```
// Preprocessor directives for SetDllDirectory
#include <pylon/Platform.h>
// Path helper functions
#include "Shlwapi.h"
#pragma comment(lib, "Shlwapi.lib")
#include "TCHAR.h"
// Link delay load implementation needed when using SetDllDirectory
#pragma comment(lib, "delayimp")
```

4. In the application's main() function, add the following lines:

```
// Set dll relative path
#if defined PYLON_32_BUILD
const TCHAR* relPath = TEXT("dllDirectory32");
#elif defined PYLON_64_BUILD
const TCHAR* relPath = TEXT("dllDirectory64");
#endif
```

```
// Path length of dll directory must be smaller than MAX_PATH
TCHAR dirPath[MAX_PATH];

// The following code is supported by Windows 7.
// For Windows 8 or higher, other Windows Path Functions are recommended esp.
// to avoid buffer overrun.
const int cErrorGetModuleFileName = 1; // replace with your special exit code
const int cErrorPathAppend = 1; // replace with your special exit code
const int cErrorSetDllDirectory = 1; // replace with your special exit code

// Assume this is the main function of a .exe, not of a dll. Otherwise use GetModuleHandle).
if ((0 == GetModuleFileName(NULL, dirPath, MAX_PATH)) || (ERROR_INSUFFICIENT_BUFFER ==
GetLastError()))
{
    return cErrorGetModuleFileName;
}
// For Windows 8 or higher the use of PathCchRemoveFileSpec is recommended.
PathRemoveFileSpec(dirPath);

// Avoid buffer overrun when using PathAppend (i.e. here we don't care about the
// (also restricted) length of the dll's full path).
// For Windows 8 or higher the use of PathCchAppend is recommended.
if ((_tcslen(dirPath) + 1 + _tcslen(relPath) + 1 > MAX_PATH) || !PathAppend(dirPath,
relPath))
{
    return cErrorPathAppend;
}

if (!SetDllDirectory(dirPath))
{
    return cErrorSetDllDirectory;
}
```

5. Replace *dllDirectory32* and *dllDirectory64* by relative paths to the pylon DLL target directories (see step 1). The paths must be relative to the directory containing the application's executable files. You can use the double-dots specifier to denote the parent directory of the current directory.
   Example: Assuming the following target directories (sample values):
   - Executable files: **C:\Program Files\Grab\bin**
   - 32-bit pylon DLLs: **C:\Program Files\Grab\dll\pylon32**
   - 64-bit pylon DLLs: **C:\Program Files\Grab\dll\pylon64**

   In this case, you must replace `dllDirectory32` by `..\\dll\\pylon32` and `dllDirectory64` by `..\\dll\\pylon64`.

6. Use the /DELAYLOAD linker switch to specify which DLLs to delay load:
   a. In the **Solution Explorer** pane, select a project.
   b. In the **View** menu, select **Properties Pages**.
   c. Navigate to **Configuration Properties > Linker > Input**.
   d. Modify the **Delay Loaded Dlls** entry.
      Example: `/DELAYLOAD:PylonBase_v6_0.dll;PylonGUI_v6_0.dll`
7. Build all versions of your project and deploy the newly created builds.

3.3.5 Method 5: Creating and Embedding Assembly Manifests

To point your application to the pylon DLLs, you can create assembly manifests and embed them in your application project.

This is the recommended method if

- your application's executable file(s) and the dependent pylon DLLs can't be placed in the same directory (see Section 3.3.1 on page 13) and
- your application must be launchable via Command Prompt and
- you can modify the source code of your application and
- your application does not support delayed loading of DLLs or your application already uses assembly manifests.

If your application already uses assembly manifests, simply add the required pylon DLLs to the existing **.manifest** files.

To deploy the pylon DLLs using assembly manifests:

1. Create and deploy the assembly manifest files (**.manifest** files).
   For more information, see Section 3.3.5.1 on page 21.
2. Create and deploy the application configuration files (**.config** files).
   For more information, see Section 3.3.5.2 on page 22.
3. Embed the assembly manifest in your application project.
   For more information, see Section 3.3.5.3 on page 23.

3.3.5.1 Creating the Assembly Manifests (.manifest Files)

The **.manifest** files contain references to the pylon DLLs. For each version of your application (32 bit or 64 bit), you must create a separate **.manifest** file. Each file must be copied to the corresponding directory containing the pylon DLLs.

To create and deploy the .manifest file(s):

1. Open a text editor, e.g., Notepad.
2. If your application is available in a **32-bit** version:
   a. Create a new text file and add the following lines:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <assemblyIdentity type="win32" name="assemblyName32" version="1.0.0.0"
  processorArchitecture="x86"/>
  <file name="dll32FileName1" hashalg="SHA1"/>
  <file name="dll32FileName2" hashalg="SHA1"/>
</assembly>
```

   b. Replace *assemblyName32* by a unique identifier for the 32-bit assembly, e.g., `Grab.Assembly`.
   c. Use the `<file>` tags to refer to each required 32-bit pylon DLL. Replace each *dll32FileName* entry by the file name of a required DLL. Add more lines if required.
   Example:

```
<file name="GCBase_MD_VC141_v3_1_Basler_pylon.dll" hashalg="SHA1"/>
<file name="GenApi_MD_VC141_v3_1_Basler_pylon.dll" hashalg="SHA1"/>
<file name="Log_MD_VC141_v3_1_Basler_pylon.dll" hashalg="SHA1"/>
```

   d. Save the file with the same file name as *assemblyName32* and with a **.manifest** file extension, e.g., save it as Grab.Assembly.manifest.
   e. Set up your installation routine to copy the .manifest file to the directory containing the 32-bit pylon DLLs, e.g., `C:\Program Files\Grab\dll\pylon32`.
3. If your application is available in a **64-bit** version:
   a. Create a new text file and add the following lines:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <assemblyIdentity type="win32" name="assemblyName64" version="1.0.0.0"
  processorArchitecture="amd64"/>
  <file name="dll64FileName1" hashalg="SHA1"/>
  <file name="dll64FileName2" hashalg="SHA1"/>
</assembly>
```

   Note: `type` must always be `win32`, also for 64-bit applications.
   b. Replace *assemblyName64* by a unique identifier for the assembly, e.g., `Grab64.Assembly`.
   If your application is available in both 32-bit and 64-versions, the identifier must be different from the identifier specified in step 2.

c. Use the `<file>` tags to refer to each required 64-bit pylon DLL. Replace each *`dll64FileName`* entry by the file name of a required DLL. Add more lines if required.
Example:
```
<file name="GCBase_MD_VC141_v3_1_Basler_pylon.dll" hashalg="SHA1"/>
<file name="GenApi_MD_VC141_v3_1_Basler_pylon.dll" hashalg="SHA1"/>
<file name="Log_MD_VC141_v3_1_Basler_pylon.dll" hashalg="SHA1"/>
```

d. Save the manifest file with the same file name as *`assemblyName64`* and with a **.manifest** file extension, e.g., save it as Grab64.Assembly.manifest.

e. Set up your installation routine to copy the .manifest file to the directory containing the 64-bit pylon DLLs, e.g., `C:\Program Files\Grab\dll\pylon64`.

3.3.5.2 Creating the Application Configuration Files (.config Files)

The **.config** files point each application's executable file to its corresponding **.manifest** file. For each executable file (32 bit or 64 bit), you must create a separate **.config** file.

Each **.config** file must be copied to the same directory as the corresponding executable file.

To create and deploy the .config file:

1. Open a text editor, e.g., Notepad.
2. Add the following lines:
```
<configuration>
  <windows>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <probing privatePath="dllPath"/>
    </assemblyBinding>
  </windows>
</configuration>
```
3. Replace *`dllPath`* by the relative paths to the directories containing the 32-bit and 64-bit pylon DLLs. Delimit each directory path with a semicolon. Each path must be relative to the directory containing the application's executable files. You can use the double-dots specifier to denote the parent directory of the current directory.
Example: Assuming the following target directories (sample values):
   - Executable files: **C:\Program Files\Grab\bin**
   - 32-bit pylon DLLs: **C:\Program Files\Grab\dll\pylon32**
   - 64-bit pylon DLLs: **C:\Program Files\Grab\dll\pylon64**

   In this case, you must replace *`dllPath`* by `..\dll\pylon32;..\dll\pylon64`.
4. Save the file with the same file name as your application's executable and with a **.config** file extension, e.g., save it as Grab.config.
5. If your application is available in 32-bit and 64-bit versions, save the file again with the file name of your 64-bit executable file, e.g., Grab64.config.
6. Set up your installation routine to copy the **.config** file(s) to the directory containing your application's executable file(s).

3.3.5.3 Embedding the Manifest

After you have created the **.manifest** and **.config** files, you must embed the manifest in the source code of your application.

To embed the assembly manifest in your application project:

1. Open your application project in an IDE, e.g., Visual Studio.
2. If your application is available **only** in a 32-bit version or **only** in a 64-bit version:
   a. Add the following lines at the beginning of your application's source code:

   ```
   #pragma comment(linker,"/manifestdependency:\"type='win32' name='assemblyName' \
   version='1.0.0.0' processorArchitecture='procArch' \"")
   ```

   b. Replace *assemblyName* by the name of your assembly, e.g., `Grab.Assembly`.

   Replace *procArch* by `x86` for a 32-bit application or by `amd64` for a 64-bit application.
   Note: `type` must always be `win32`, also for 64-bit applications.
3. If your application is available in **both** 32-bit and 64-bit versions:
   a. Add the following lines at the beginning of your application's source code:

   ```
   #if defined _M_IX86
   #pragma comment(linker,"/manifestdependency:\"type='win32' name='assemblyName32' \
   version='1.0.0.0' processorArchitecture='x86' \"")
   #elif defined _M_X64
   #pragma comment(linker,"/manifestdependency:\"type='win32' name='assemblyName64' \
   version='1.0.0.0' processorArchitecture='amd64' \"")
   #endif
   ```

   b. Replace *assemblyName32* by the 32-bit assembly name (e.g., `Grab.Assembly`)
   c. Replace *assemblyName64* by the 64-bit assembly name (e.g., `Grab64.Assembly`).
4. Build all versions of your project and deploy the newly created builds.

Revision History

Document Number	Date	Changes
AW00136201000	06 Nov 2015	Initial release of the document.
AW00136202000	02 May 2017	Added Basler.Pylon.dll in Table 2 on page 4.
AW00136203000	10 May 2017	Removed information about camera interface-specific runtime redistributable packages in Chapter 1 on page 1 and Chapter 2 on page 2.
AW00136204000	06 Nov 2017	Added required files in Table 2 on page 4.
AW00136205000	12 Jun 2018	Updated file names throughout the document.
AW00136206000	12 Feb 2019	Updated file names throughout the document.
AW00136207000	02 Jul 2019	Revised the entire document to reflect changes due to the release of the pylon Camera Software Suite version 6. Among other things, information for IEEE 1394 has been removed and information for CoaXPress 2.0 has been added.