**Why Basler’s pylon APIs?**

Because then setting up an image acquisition only needs **3 lines of code.** Reduce time, cost and headache.

**Increase your productivity and get fast results**

**80% time savings** – Studies showed that independent developers finished tasks in only 20% or less of the time that they needed to complete the same tasks with other comparable APIs.

**Faster results through simplicity** – The simple structure of the pylon APIs lets users focus on the application itself during the development instead of having to deal with complicated camera connection issues.

**Easy to learn, thorough documentation** – With the easy-to-learn pylon APIs and goal-oriented, context-related developer documentation, even new employees can become productive right away.

**Make your software more stable**

**60% less programming code** – Compared to the APIs from other providers, applications developed with pylon APIs require only about 40% of the programming code for the same functions. This also helps increase the stability and robustness of your applications.

**Fewer causes of failure, more stability** – Thanks to the reduced programming scope and the simplicity of developing your own application, the search for errors is also greatly simplified. Many potential causes of failure are completely avoided from the start.

**Certified drivers, reliable performance** – Tried and used thousands of times, certified drivers and the performance speak for the stability of the pylon APIs, which have been optimized continuously for over 12 years.

**Minimize your hardware costs**

**Camera features without straining the CPU** – Use the resources of your hardware for your application. The pylon APIs are optimized for efficient use of the CPU resources and utilize the firmware features in the cameras.

**Minimal footprint** – Let your application use the most out of your hardware and avoid unnecessary hardware upgrades.

**Stay independent**

**Choose any interface** – Thanks to GenICam and GenTL support, it shouldn’t matter which interface you use to connect your camera. If you do want to use a camera connected via GigE rather than a USB camera, you don’t have to make any changes in your application.

**Use GenICam and GenTL without training** – The pylon APIs rely on these standards. The complex details of these standards are encapsulated by the pylon APIs. You can benefit from the advantages of GenICam and GenTL during the development without needing specific knowledge of the standards.

**Platform-independent** – The target platform of the developed application doesn’t play any role either. It’s very easy to switch from a Windows environment to a Linux ARM environment without major code changes.

**Flexibility in the development process** – The pylon APIs make it possible, for example, to easily develop an application in Windows with USB cameras which will then be tested in a Linux-based ARM environment and used with GigE cameras.

**Secure your future viability**

**Preserve downward capability** – When developing new pylon versions, preserving the compatibility with previous versions is always a top priority. Applications that were developed with pylon APIs should benefit from improvements and advancements in newer pylon API versions with the least amount of effort.

**State-of-the-art camera support** – With the help of pylon APIs, this allows you to always use the latest camera models and features in your application without much effort.

**Older cameras with new pylon** – Even older camera models can be controlled by the latest pylon versions. For example, you can use a ten-year-old camera design with the current pylon to benefit from the advantages offered by the newest optimizations and simplifications of the pylon APIs.
The following example shows the pylon code in C++ that is needed to acquire an image and save it as a file.

```cpp
#include <pylon/PylonIncludes.h>
using namespace Pylon;
using namespace std;

int main(int argc, char* argv[]) {
    PylonInitialize();

    try {
        CInstantCamera camera(CTlFactory::GetInstance().CreateFirstDevice());
        CGrabResultPtr ptrImg;
        camera.GrabOne(10000, ptrImg);
        CImagePersistence::Save(ImageFileFormat::_Png, "myImage.png", ptrImg);
    }
    catch (const GenericException& e) {
        cerr << "Exception: " << endl << e.GetDescription() << endl;
    }

    PylonTerminate();
    return 0;
}
```
Sample code of a typical camera API

The following example shows the code of a typical camera API compared to the code of the pylon API in C++ that is needed to acquire an image and output the value of the first pixel.

**Typical camera API**

```cpp
int main()
{
    some::Library::Initialize();
    auto deviceManager = some::DeviceManager::Instance();
    try
    {
        deviceManager.Update();
        auto device = deviceManager.Devices().at(0)->OpenDevice(some::DeviceAccessType::Control);
        auto nodeMapRemoteDevice = FindNode<somenodes::EnumerationNode>("TriggerSelector")
            ->SetCurrentEntry("ExposureStart");
        auto nodeMapRemoteDevice = FindNode<somenodes::EnumerationNode>("TriggerMode")
            ->SetCurrentEntry("Off");
        nodeMapRemoteDevice = FindNode<somenodes::FloatNode>("AcquisitionFrameRate")
            ->SetValue(10);  
        device = DataStreams().at(0)->OpenDataStream();
        nodeMapRemoteDevice = FindNode<somenodes::CommandNode>("AcquisitionStart")
            ->Execute();
        auto buffer = DataStreams()->WaitForFinishedBuffer(5000);
        auto image = some::BufferTo<some::ipl::Image>(buffer);
        std::cout << "First pixel: " << static_cast<uint16>(image.Pixels(0, 0)) << " \n;"
    }
    catch (const std::exception& e)
    {
        std::cout << "EXCEPTION: " << e.what() << std::endl;
    }
    some::Library::Close();
    return 0;
}
```

**pylon API**

```cpp
int main(int argc, char* argv[])
{
    PylonInitialize();
    try
    {
        CInstantCamera camera = CTIFactory::GetInstance().CreateFirstDevice();
        CGrabResultPtr ptrGrabResult;
        camera.GrabOne(10000, ptrGrabResult);
        const uint8_t* pImageBuffer = (uint8_t*)ptrGrabResult->GetBuffer();
        cout << "First pixel: " << pImageBuffer[0] << endl;
    }
    catch (const GenericException& e)
    {
        cerr << "Exception:" << endl << e.GetDescription() << endl;
    }
    PylonTerminate();
    return 0;
}
```