

Basler blaze-101

```
// Create an instant camera object with the first
Camera_t camera( CTIFactory::GetInstance().Creat

// Register an image event handler that accesses
camera.RegisterImageEventHandler( new CSampleIma
Ownership_TakeOwnership);

// Open the camera.
camera.Open();
```

MIGRATING FROM TOF640 TO BLAZE-101

Document Number: DG001625

Version: 01 Language: 000 (English)

Release Date: 9 July 2020

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

**All material in this publication is subject to change without notice and is copyright
Basler AG.**

1 Introduction

If you have been working with a tof640-20gm_850nm camera before and you want to switch to the Basler blaze-101 camera, you may find the following information useful to migrate any existing application code. This documents contains a list of required changes as well information if you want to continue working with both camera models.

2 Migration Information

The SDK for the blaze-101 differs from the Basler ToF Driver software package delivered with the tof640-20gm_850nm. To update your application code, make the changes detailed in this section.

2.1 Class Name Changes

Replace All Occurrences of	By
CToFCamera	CBlazeCamera
#include <ConsumerImplHelper/ToFCamera.h>	#include <ConsumerImplHelper/BlazeCamera.h>

Table 1: Class Name Changes

2.2 Point Clouds: Identifying Missing Depth Data

The method to identify missing depth data differs between the camera models.

tof640-20gm_850nm: When there is no depth information available for a certain pixel, all three coordinates of the corresponding point in the point cloud are set to NaN (Not a Number). The CToFCamera::Coord3D::IsValid() convenience method allowed you to check for NaNs.

blaze-101: Instead of setting the coordinates to NaN, they are set to the value specified by the Scan3dInvalidDataValue parameter. The default value is zero.

When migrating your application, you have two options:

- Continue using NaNs to mark missing depth data
- Use 0 (or another user-defined value) to mark missing depth data

Both options are explained in the following sections.

2.2.1 Continue Using NaNs

If you want to continue using NaNs to mark missing depth data, set the `Scan3dInvalidDataValue` parameter to the following value:

```
std::numeric_limits<double>::quiet_NaN();
```

The following sample illustrates this:

```
// Configure the camera to use NaNs to mark missing depth data.
GenApi::CEnumerationPtr ptrCoordinateSelector = camera.GetParameter("Scan3dCoordinateSelector");
GenApi::CFloatPtr ptrInvalidDataValue = camera.GetParameter("Scan3dInvalidDataValue");
for (auto axis : { "CoordinateA", "CoordinateB", "CoordinateC" })
{
    // Choose axis to configure.
    ptrCoordinateSelector->FromString(axis);
    // Set invalid data value for that axis.
    ptrInvalidDataValue->SetValue(std::numeric_limits<float>::quiet_NaN());
}
```



The `CBlazeCamera::Coord3D` struct doesn't have a `IsValid()` method anymore. You have to replace previous checks like

```
if ( pCoordinate->isValid() )
by
if ( pCoordinate->z == pCoordinate->z )
or
if (! std::isnan(pCoordinate->z))
```

2.2.2 Using 0 (Zero) or Another User-Defined Value

The following code snippet demonstrates how to set the value to mark missing depth data:

```
const float invalidDepthValue = 0;
GenApi::CEnumerationPtr ptrCoordinateSelector = camera.GetParameter("Scan3dCoordinateSelector");
GenApi::CFloatPtr ptrInvalidDataValue = camera.GetParameter("Scan3dInvalidDataValue");
for (auto axis : { "CoordinateA", "CoordinateB", "CoordinateC" })
{
    // Choose axis to configure.
    ptrCoordinateSelector->FromString(axis);
    // Set invalid data value for that axis.
    ptrInvalidDataValue->SetValue(invalidDepthValue);
}
```

When processing the point cloud data, check whether a point represents valid depth data:

```
if ( pCoordinate->z != invalidDepthDataValue )
{
    // depth information is valid
}
```

2.3 Depth Map: Extracting 3D Data From the 2D Depth Map

The way how depth information is encoded as gray values in the 2D range map differs between the ToF-ES and the blaze-101.

To extract 3D data from the 2D depth map on a blaze-101 camera:

1. Retrieve the required information from camera:

```
// Retrieve calibration data from the camera.
const double cx = GenApi::CFloatPtr(camera.GetParameter("Scan3dPrincipalPointU"))->GetValue();
const double cy = GenApi::CFloatPtr(camera.GetParameter("Scan3dPrincipalPointV"))->GetValue();
const double f = GenApi::CFloatPtr(camera.GetParameter("Scan3dFocalLength"))->GetValue();
GenApi::CEnumerationPtr(camera.GetParameter("Scan3dCoordinateSelector"))->
    FromString("CoordinateC");
const double grey2mm = GenApi::CFloatPtr(camera.GetParameter("Scan3dCoordinateScale"))->GetValue();
```

2. Using this information, transform gray values to 3D coordinates like this:

```
uint16_t g = ((uint16_t*)parts[0].pData)[u + v * width];
if (g != (uint16_t) invalidDepthDataValue)
{
    // Calculate 3D coordinate from grey value and pixel coordinate.
    const double z = g * grey2mm;
    const double x = (u - cx) * z / f;
    const double y = (v - cy) * z / f;
}
```

2.4 Confidence Map

Confidence maps acquired by tof640-20gm_850nm cameras encode the reliability of the depth measurement as 16-bit integer values. The higher the value, the more light has been captured and the more reliable the measurement is.

In contrast, the confidence maps acquired by a blaze-101 camera are binary confidence maps. This means that a value of 0 indicates that there is no depth information available, and a value of 255 (with the PixelFormat parameter set to Confidence8) or 65535 (with the PixelFormat parameter set to Confidence16) indicates that depth information is available.

If this binary information is not sufficient for your application, you can combine the confidence map with the intensity image. If for a certain pixel the confidence map isn't zero, you can take the corresponding pixel value of the intensity image as a measure of how reliable the depth data is. When using the intensity image as a tool to determine the confidence, Basler recommends disabling the gamma correction applied to the intensity image by setting the `GammaCorrection` parameter to false.

2.5 Parameter Changes

Value Range of Confidence Threshold

On the blaze-101, the value range of the `ConfidenceThreshold` parameter is 0–255.

When you set this parameter in your application, you determine and set a new confidence threshold value that suits your needs.

PTP / Synchronous Free Run

Clock synchronization via PTP and the synchronous free run feature are not yet supported by the current blaze-101 firmware.

FilterStrength

If your application sets the `FilterStrength` parameter, you may have to check and probably adjust that value.

FilterRange

This parameter is not available on the blaze-101 camera.

OutlierTolerance

This parameter is not available on the blaze-101 camera.

Digital IOs

There is no digital IO control for the blaze-101 camera.

User Sets / Factory Sets

The current firmware does not yet support user sets and factory sets.

3 Comparison of Samples

Following you can find 2 samples with the differences highlighted.

Grab Loop - blaze-101

```
#include <iostream>
#include <iomanip>
#include <ConsumerImplHelper/BlazeCamera.h>

using namespace GenTLConsumerImplHelper;

class Sample
{
public:
    int run();
    bool onImageGrabbed(GrabResult grabResult, BufferParts);
private:
    CBlazeCamera m_Camera;
    int m_nBuffersGrabbed;
};

int Sample::run()
{
    m_nBuffersGrabbed = 0;
    try
    {
        m_Camera.OpenFirstCamera();

        GenApi::CEnumerationPtr ptrComponentSelector = m_Camera.GetParameter("ComponentSelector");
        GenApi::CBooleanPtr ptrComponentEnable = m_Camera.GetParameter("ComponentEnable");
        GenApi::CEnumerationPtr ptrPixelFormat = m_Camera.GetParameter("PixelFormat");

        ptrComponentSelector->FromString("Range");
        ptrComponentEnable->SetValue(true);
        ptrPixelFormat->FromString("Coord3D_ABC32f");

        ptrComponentSelector->FromString("Intensity");
        ptrComponentEnable->SetValue(true);
        ptrPixelFormat->FromString("Monol6");

        ptrComponentSelector->FromString("Confidence");
        ptrPixelFormat->FromString("Confidence16");
        ptrComponentEnable->SetValue(true);

        // Configure the value used for identifying missing depth information.
        GenApi::CEnumerationPtr(m_Camera.GetParameter("Scan3dCoordinateSelector"))->
            FromString("CoordinateC");
        // z-coordinate will be set to this value if there is no valid depth data available
        GenApi::CFloatPtr(m_Camera.GetParameter("Scan3dInvalidDataValue"))->SetValue(0.0);

        m_Camera.GrabContinuous(5, 1000, this, &Sample::onImageGrabbed);

        m_Camera.Close();
    }
    catch (const GenICam::GenericException& e)
    {
        std::cerr << "Exception occurred: " << e.GetDescription() << std::endl;
        if (m_Camera.IsOpen() && !m_Camera.IsConnected())
        {
            std::cerr << "Camera has been removed." << std::endl;
        }
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}

bool Sample::onImageGrabbed(GrabResult grabResult, BufferParts parts)
{
    if (grabResult.status == GrabResult::Timeout)
    {
        std::cerr << "Timeout occurred. Acquisition stopped." << std::endl;
        if (!m_Camera.IsConnected())
        {
            std::cerr << "Camera has been removed." << std::endl;
        }
        return false; // Indicate to stop acquisition
    }
    m_nBuffersGrabbed++;
}
```

Grab Loop - ToF-ES

```
#include <iostream>
#include <iomanip>
#include <ConsumerImplHelper/ToFCamera.h>

using namespace GenTLConsumerImplHelper;

class Sample
{
public:
    int run();
    bool onImageGrabbed(GrabResult grabResult, BufferParts);
private:
    CToFCamera m_Camera;
    int m_nBuffersGrabbed;
};

int Sample::run()
{
    m_nBuffersGrabbed = 0;
    try
    {
        m_Camera.OpenFirstCamera();

        GenApi::CEnumerationPtr ptrComponentSelector = m_Camera.GetParameter("ComponentSelector");
        GenApi::CBooleanPtr ptrComponentEnable = m_Camera.GetParameter("ComponentEnable");
        GenApi::CEnumerationPtr ptrPixelFormat = m_Camera.GetParameter("PixelFormat");

        ptrComponentSelector->FromString("Range");
        ptrComponentEnable->SetValue(true);
        ptrPixelFormat->FromString("Coord3D_ABC32f");

        ptrComponentSelector->FromString("Intensity");
        ptrComponentEnable->SetValue(true);
        ptrPixelFormat->FromString("Monol6");

        ptrComponentSelector->FromString("Confidence");

        ptrComponentEnable->SetValue(true);

        m_Camera.GrabContinuous(5, 1000, this, &Sample::onImageGrabbed);

        m_Camera.Close();
    }
    catch (const GenICam::GenericException& e)
    {
        std::cerr << "Exception occurred: " << e.GetDescription() << std::endl;
        if (m_Camera.IsOpen() && !m_Camera.IsConnected())
        {
            std::cerr << "Camera has been removed." << std::endl;
        }
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}

bool Sample::onImageGrabbed(GrabResult grabResult, BufferParts parts)
{
    if (grabResult.status == GrabResult::Timeout)
    {
        std::cerr << "Timeout occurred. Acquisition stopped." << std::endl;
        if (!m_Camera.IsConnected())
        {
            std::cerr << "Camera has been removed." << std::endl;
        }
        return false; // Indicate to stop acquisition
    }
    m_nBuffersGrabbed++;
}
```

```

if (grabResult.status != GrabResult::Ok)
{
    std::cerr << "Image " << m_nBuffersGrabbed << " was not grabbed." << std::endl;
}
else
{
    // Retrieve the values for the center pixel
    const int width = (int)parts[0].width;
    const int height = (int)parts[0].height;
    const int x = (int)(0.5 * width);
    const int y = (int)(0.5 * height);
    CBlazeCamera::Coord3D* p3DCoordinate = (CBlazeCamera::Coord3D*) parts[0].pData + y*width + x;
    uint16_t* pIntensity = (uint16_t*)parts[1].pData + y * width + x;
    uint16_t* pConfidence = (uint16_t*)parts[2].pData + y * width + x;

    std::cout << "Center pixel of image " << std::setw(2) << m_nBuffersGrabbed << ": ";
    std::cout.setf(std::ios_base::fixed);
    std::cout.precision(1);
    if (p3DCoordinate->z != 0)
        std::cout << "x=" << std::setw(6) << p3DCoordinate->x << " y=" << std::setw(6)
            << p3DCoordinate->y << " z=" << std::setw(6) << p3DCoordinate->z;
    else
        std::cout << "x=  n/a y=  n/a z=  n/a";
    std::cout << " intensity=" << std::setw(5) << *pIntensity << " confidence="
        << std::setw(5) << *pConfidence << "\n";
}
return m_nBuffersGrabbed < 10; // Indicate to stop acquisition when 10 buffers are grabbed
}

int main(int argc, char* argv[])
{
    int exitCode = EXIT_SUCCESS;
    try
    {
        CBlazeCamera::InitProducer();
        exitCode = Sample().run();
    }
    catch (GenICam::GenericException& e)
    {
        std::cerr << "Exception occurred: " << std::endl << e.GetDescription() << std::endl;
        exitCode = EXIT_FAILURE;
    }
    if (CBlazeCamera::IsProducerInitialized())
        CBlazeCamera::TerminateProducer();

    return exitCode;
}

```

```

if (grabResult.status != GrabResult::Ok)
{
    std::cerr << "Image " << m_nBuffersGrabbed << " was not grabbed." << std::endl;
}
else
{
    // Retrieve the values for the center pixel
    const int width = (int)parts[0].width;
    const int height = (int)parts[0].height;
    const int x = (int)(0.5 * width);
    const int y = (int)(0.5 * height);
    CToFCamera::Coord3D* p3DCoordinate = (CToFCamera::Coord3D*) parts[0].pData + y*width + x;
    uint16_t* pIntensity = (uint16_t*)parts[1].pData + y * width + x;
    uint16_t* pConfidence = (uint16_t*)parts[2].pData + y * width + x;

    std::cout << "Center pixel of image " << std::setw(2) << m_nBuffersGrabbed << ": ";
    std::cout.setf(std::ios_base::fixed);
    std::cout.precision(1);
    if (p3DCoordinate->IsValid())
        std::cout << "x=" << std::setw(6) << p3DCoordinate->x << " y=" << std::setw(6)
            << p3DCoordinate->y << " z=" << std::setw(6) << p3DCoordinate->z;
    else
        std::cout << "x=  n/a y=  n/a z=  n/a";
    std::cout << " intensity=" << std::setw(5) << *pIntensity << " confidence="
        << std::setw(5) << *pConfidence << "\n";
}
return m_nBuffersGrabbed < 10; // Indicate to stop acquisition when 10 buffers are grabbed
}

int main(int argc, char* argv[])
{
    int exitCode = EXIT_SUCCESS;
    try
    {
        CToFCamera::InitProducer();
        exitCode = Sample().run();
    }
    catch (GenICam::GenericException& e)
    {
        std::cerr << "Exception occurred: " << std::endl << e.GetDescription() << std::endl;
        exitCode = EXIT_FAILURE;
    }
    if (CToFCamera::IsProducerInitialized())
        CToFCamera::TerminateProducer();

    return exitCode;
}

```

Depth Map blaze-101

```
// Retrieve calibration data from the camera.
const double cx = GenApi::CFloatPtr(camera.GetParameter("Scan3dPrincipalPointU"))->GetValue();
const double cy = GenApi::CFloatPtr(camera.GetParameter("Scan3dPrincipalPointV"))->GetValue();
const double f = GenApi::CFloatPtr(camera.GetParameter("Scan3dFocalLength"))->GetValue();

GenApi::CEnumerationPtr(camera.GetParameter("Scan3dCoordinateSelector"))->FromString("CoordinateC");
const double grey2mm = GenApi::CFloatPtr(camera.GetParameter("Scan3dCoordinateScale"))->GetValue();

// Inside of grab loop: Conversion grey value -> 3D coordinate
uint16_t g = ((uint16_t*) parts[0].pData)[u + v * width];
if (g != missingDepth)
{
    // Calculate 3D coordinate from grey value and pixel coordinate.
    const double z = g * grey2mm;
    const double x = (u - cx) * z / f;
    const double y = (v - cy) * z / f;
}
```

Depth Map ToF-ES

```
// Retrieve calibration data from the camera.

const double cx = GenApi::CFloatPtr(camera.GetParameter("CenterX"))->GetValue();
const double cy = GenApi::CFloatPtr(camera.GetParameter("CenterY"))->GetValue();
const double f = GenApi::CFloatPtr(camera.GetParameter("FocalLength"))->GetValue();

GenApi::CIntegerPtr ptrDepthMin = camera.GetParameter("DepthMin");
GenApi::CIntegerPtr ptrDepthMax = camera.GetParameter("DepthMax");
const int64_t DMin = ptrDepthMin->GetValue();
const int64_t DMax = ptrDepthMax->GetValue();

const double grey2mm = (double)(DMax - DMin) / 0xffff;

// Inside of grab loop: Conversion grey value -> 3D coordinate
uint16_t g = ((uint16_t*) parts[0].pData)[u + v * width];
if (g != missingDepth)
{
    // Calculate 3D coordinate from grey value and pixel coordinate.
    const double z = g * grey2mm + DMin;
    const double x = (u - cx) * z / f;
    const double y = (v - cy) * z / f;
}
```

4 Using Both blaze-101 and tof640-20gm_850nm Cameras

If your application has to support both camera models, at runtime both the binaries of the blaze SDK and the Basler ToF Driver software package are required, i.e., on the target system the blaze SDK and the ToF driver package must be installed.

The blaze SDK contains all header files and classes required for the tof640-20gm_850nm camera which makes it suitable for developing applications for both camera models. The same isn't possible with the Basler ToF Driver software package. Therefore, to develop your application, use the blaze SDK only. This means that you must configure the include path settings in such a way that only the **include** folder of the blaze SDK is included in the compiler's include path.

Revision History

Doc. ID Number	Date	Changes
AW00162501000	9 Jul 2020	Initial release of this document.