# NEW CVB API - .NET - C++ - PYTHON

A NEW APPROACH TO MODERN
APPLICATION DEVELOPMENT
WITH COMMON VISION BLOX

**ANDREAS RITTINGER, IMAGE ACQUISITION,
STEMMER IMAGING**


STEMMER® IMAGING

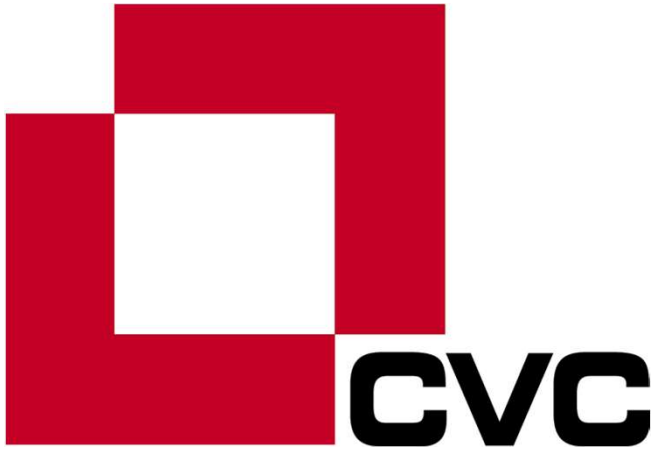
STEMMER® IMAGING

# CONTENT

STEMMER® IMAGING

# RETROSPECT 1997

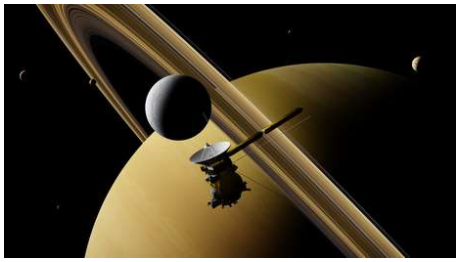
***Autumn 1997: Common Vision Concept*** *(later on renamed Common Vision Blox) is publicly presented for the first time on the Vision Show 1997 in Stuttgart*



# RETROSPECT 1997

***Autumn 1997: Common Vision Concept***
*(later on renamed Common Vision Blox) is publicly presented for the first time on the Vision Show 1997 in Stuttgart*

… and this is what else 1997 had in store:


space probe Cassini starts its mission


comet Hale-Bopp passes Earth

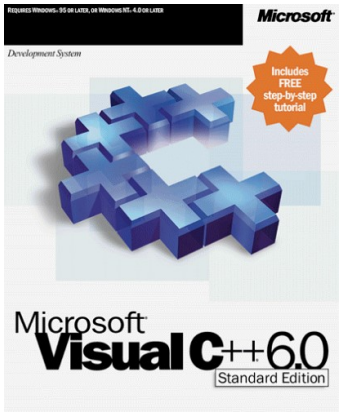

Pathfinder lands on Mars


Garri Kasparov loses against Deep Blue

STEMMER® IMAGING

# THE C-API OF CVB

*The environment in the mid 90's significantly influenced the design of the API of Common Vision Concept (CVC)*




- Procedural/imperative programming is still predominant in the beginning and mid 90's. Object oriented programming only just started its rise to significance.


- CVC is supposed to be usable with the most popular programming languages and development environments for the PC, which were by and large:

  (see e. g. https://www.complang.tuwien.ac.at/anton/comp.lang-statistics/)

  - Pascal/Delphi (3)
  - (Visual) Basic (5.0)
  - (Visual) C/C++ (5.0)

STEMMER® IMAGING

# THE C-API OF CVB

- Programming libraries (DLLs) that export functions declared as
  ```
  extern "C" __stdcall ...
  ```
  are usable with all these languages and development environments


- Use of these DLLs is simple, not least because they shield the caller from the callee's dependencies (unlike e.g. class DLLs) and because the interface is limited to few and elementary data types.
- They are also usable in a number of other environments (like e.g. LabVIEW, Matlab, …)
- Adaption to the .NET runtime released in 2001 also was straightforward and CVB is therefore available for C# and VB.NET programmers starting in 2002.
- ActiveX Controls for GUI components (e.g. interactive display), also usable with all the relevant environments back then, complete the product.

STEMMER® IMAGING

# THE C-API OF CVB

*In summary, the goal has been attained: Common Vision Blox, launched in the 2nd half of the 90's, is a machine vision SDK which…*

- … is state of the art.
- … comes with powerful algorithms for image analysis and pattern recognition which 20 years on are still competitive.
- … allows for hardware-independent coding of image acquisition.
- … is usable with all the currently popular development tools.

STEMMER® IMAGING

# 2018 – THE WORLD KEEPS TURNING…

*The basic parameters in 2018 differ notably from those 21 years ago.*

- Object oriented development is the standard. Graduates usually have learned the relevant languages and techniques and to this target audience a purely procedural API looks alien and uncomely.
- With the .NET framework taking root, the replacement of VB 6.0 by VB.NET and the 64 bit versions of Windows becoming mainstream ActiveX controls have lost a lot of their former significance and support for them in development environments is crumbling. ActiveX can no longer be considered forward-looking.

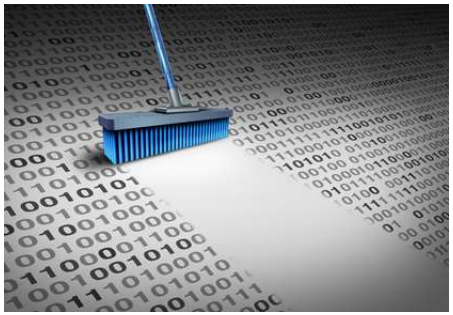
# 2018 – THE WORLD KEEPS TURNING…

*The basic parameters in 2018 differ notably from those 21 years ago.*

- The availability of powerful low-cost embedded computers promotes diversity of the hardware and operating system platforms for machine vision applications.
- Technologies with cross-platform applicability (like e.g. C++, Python, C#) are becoming more important.
- *A new approach is needed to prepare CVB for a new generation of developers and applications.*

STEMMER® IMAGING

# AND WHILE WE'RE AT IT…



*… we might as well address some of the rough edges and introduce improvements that were hitherto impossible to do without breaking backward compatibility of the API.*

- Differing error reporting in different modules (bool, HRESULT, integers).
- Partly confusing call semantics, e.g.:

```
LoadImageFile(const char szName, IMG& img)
LoadSF(SF& sf, const char szName)
```

- Object handles that are not type safe (`IMG`, `CLF`, `RESULTS`, `SF`, … ► at the end of the day all `void*`).
- Identical concepts appearing under different names („Locality“, „Separation“, „MinDist“).
- The COM-way of doing reference counting (`ShareObject`/`ReleaseObject`) which remained an enigma for many users and is one of the most common sources of mistakes when using CVB.
- Each module using its own result containers (`PIXELLIST`, `RESULTS`, `SEARCHRESULTS`, etc.) for lack of a generic container that was usable with all supported languages.

STEMMER® IMAGING

# DESIGN GOALS FOR A NEW API

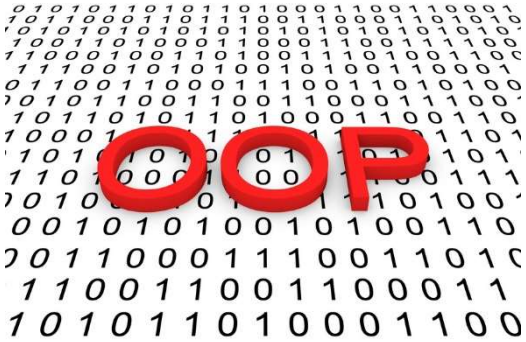
*What does an API which addresses the aforementioned items and modernizes CVB need to look like?*

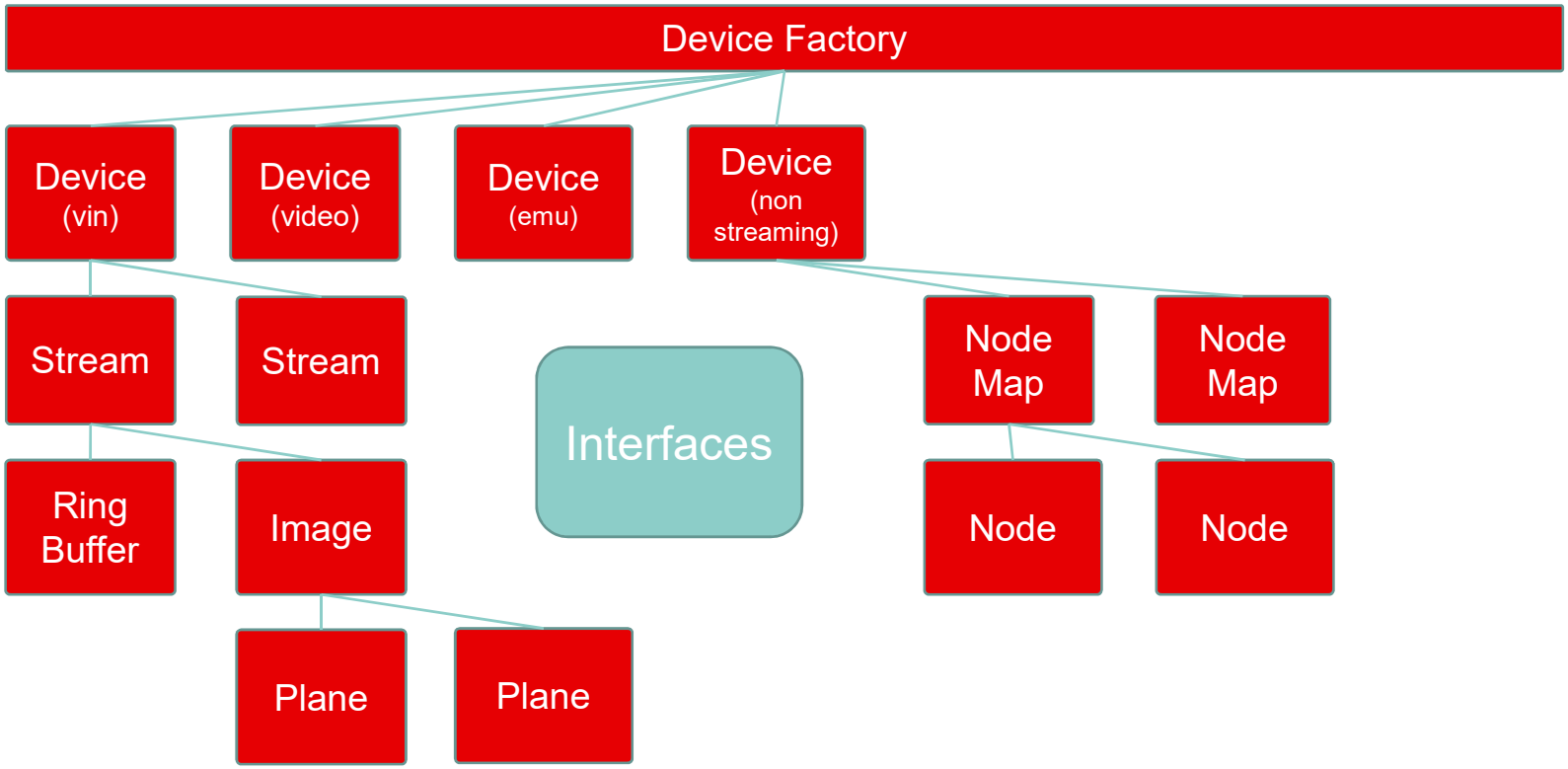
- The new API needs to be built as a lean and efficient wrapper on top of the C-API.
- It consistently uses objet oriented idioms and thereby…
  - … is easier to use and provides a simpler entry into the SDK.
  - … can be more easily integrated in common OOP patterns.
- It is to be “natively” embedded in the respective environment and its conventions:
  - Working with CVB.NET should feel no different than working with the CLR
  - Working with CVB++ should feel no different than working with STL/Boost/Qt/…
  - Working with CVBpy should feel no different than working with common Python modules.
- This may in fact have priority over preserving identical names as the C-API and between CVB.Net, CVB++ and CVBpy.
- But: Concepts that are specific to CVB will be modeled consistently between the APIs.

STEMMER® IMAGING

# OBJECT ORIENTED PHILOSOPHY

*The basic classes the wrappers provide*



STEMMER® IMAGING

# LIFE TIME MANAGEMENT

*How to actually close your device and free all depended resources?*

- Smart Pointers in C++
  - Closing a device (run out of scope)
  - … does decrease your recount – only closes if ref count is zero.
  - … dependent objects will keep the device alive and open
  - … accessing them will work as long as you own a reference.
- References in .NET and Python
  - Closing a device (dispose or with/del)
  - … you can easily make sure that the device is closed
  - … will free all dependent Objects Node Maps, Streams Images etc.
  - … accessing them will throw

STEMMER® IMAGING

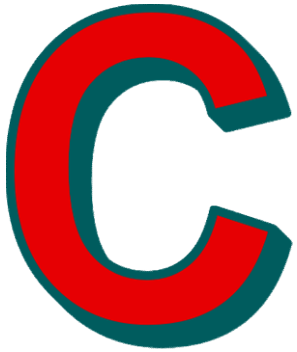
# ERROR HANDLING WITH EXCEPTIONS

*What does an API which addresses the aforementioned items and modernizes CVB need to look like?*

- No error codes, no accidental ignoring errors.
  - … only if you really want to using *Try* versions where an applicable
- Live management, destructors do cleanup
  - No leaks caused by errors
  - No gotos
- Exception carry more information
  - No lookup of error codes is required
  - Verbose error messages
- Much easier to implement…

STEMMER® IMAGING

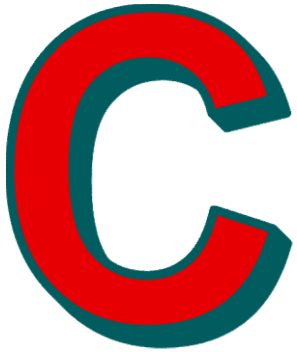
# SIMPLE ACQUISITION C-API



```c
int main(int, char* [])
{
  int i;
  IMG img;

  LoadImageFile("GenICam.vin", img);

  G2Grab(img);
  for (i = 0; i < 10; ++i)
  {
    G2Wait(img);
    printf("acquired image: %i", i);
  }

  G2Freeze(img, 0);

  ReleaseObject(img);
}
```

STEMMER® IMAGING

# SIMPLE ACQUISITION C-API



```c
int main(int, char* [])
{
  int i = 0;
  IMG img = NULL;
  cvbres_t res = 0;

  if (!LoadImageFile("GenICam.vin", img))
  {
    printf("LoadImageFile failed");
    return -1;
  }

  res = G2Grab(img);
  if (res < 0)
  {
    printf("G2Grab failed: %i", res);
    ReleaseObject(img);
    return res;
  }

  for (i = 0; i < 10; ++i)
  {
    res = G2Wait(img);
    if (res < 0)
    {
      printf("G2Wait failed: %i", res);
      G2Freeze(img, 1); // TODO find out what this means
      ReleaseObject(img);
      return res;
    }
    printf("acquired image: %i", i);
  }…
```

STEMMER® IMAGING

# SIMPLE ACQUISITION CVB++

C++

```
int main(int, char* [])
{
  auto device =
    Cvb::DeviceFactory::Open(L"GenICam.vin");
  auto stream = device->Stream();

  stream->Start();
  for (int i = 0; i < 0; ++i)
  {
    auto waitResult = stream->WaitFor(std::chrono::milliseconds(1000));
    std::cout << "acquired image: " << i << std::endl;
  }
  stream->Abort();
}
```

STEMMER® IMAGING

# SIMPLE ACQUISITION CVB++

C++

```
int main(int, char* [])
{
  try
  {
    auto device =
      Cvb::DeviceFactory::Open(L"GenICam.vin");
    auto stream = device->Stream(0);

    stream->Start();
    for (int i = 0; i < 0; ++i)
    {
      auto waitResult = stream->WaitFor(std::chrono::milliseconds(1000));
      std::cout << "acquired image: " << i << std::endl;
    }
    stream->Abort();
  }
  catch (const std::exception& err)
  {
    std::cout << err.what() << std::endl;
  }
}
```

STEMMER® IMAGING

# DESIGN GOALS FOR A NEW API

- Consistent use of smart pointers/reference types makes the “manual” reference counting obsolete, eliminating the most common source of errors.
- Consistent use of the containers typically used in the supported languages (`System.Collections.Generic.IEnumerable<>` and the likes for C#, `std::vector<>` for C++, `[]` for Python) simplifies and unifies use of result lists.
- Error handling generally happens through exceptions, complemented by alternative `Try`-functions where it makes sense ► errors in the program flow will need to be ignored *actively.*
- At least during the introductory phase there are no stability commitments with regard to the API definition – corrections and refactoring are at this stage very likely and client code will need to be adapted when they occur.

STEMMER® IMAGING

# CURRENT STATUS

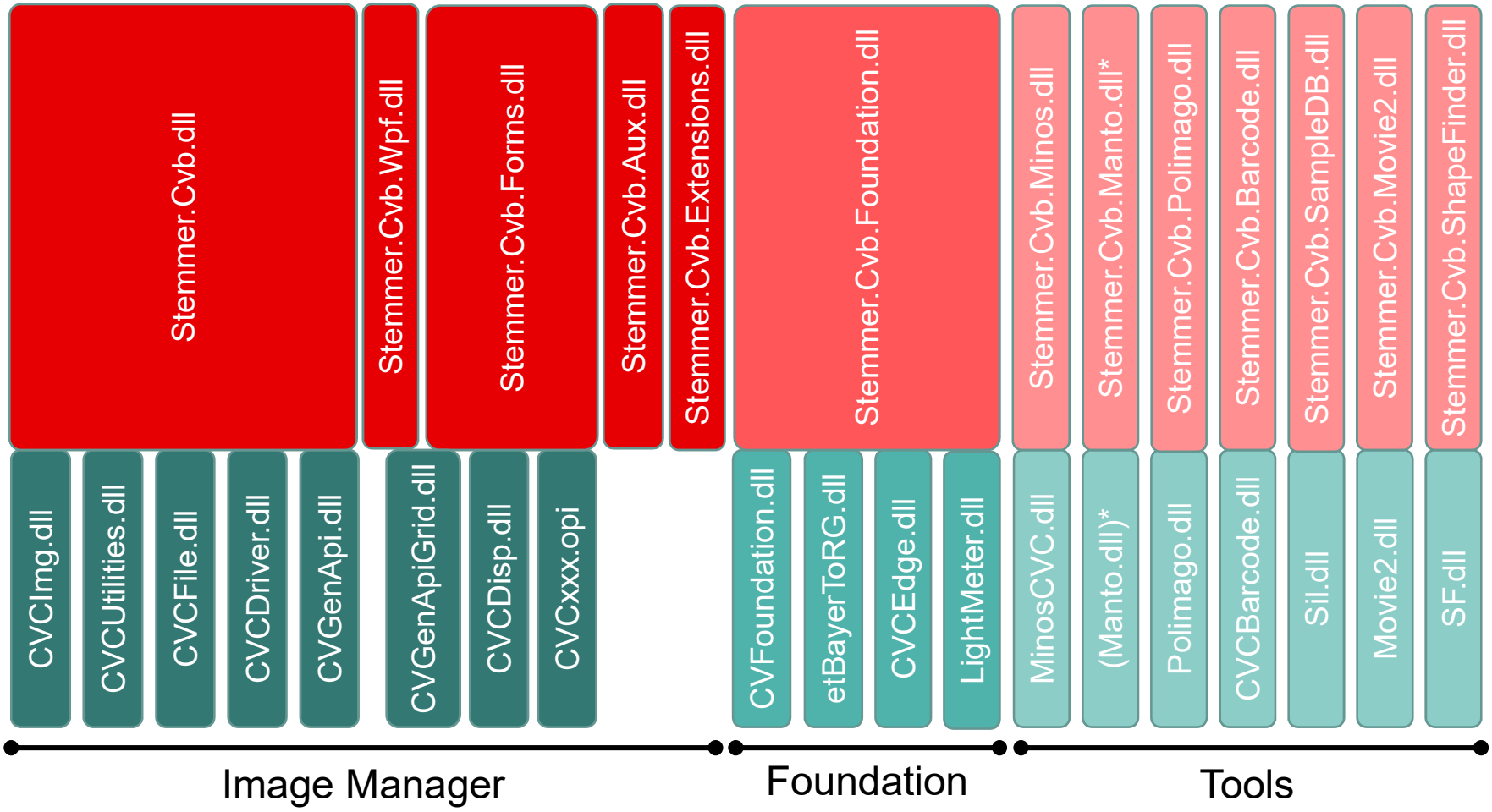
- The current state is available in the recently launched Common Vision Blox User Forum. (generally: https://forum.commonvisionblox.com CVB.NET, CVB++ and CVBpy: https://goo.gl/hSiF3h)
- Installers for CVB.NET, CVB++ and CVBpy including documentation and tutorials will updated approximately on a weekly basis.
- Support: Currently only via the User Forum.
- Requirements:

| Category | CVBNet | CVB++ | CVBpy |
|---|---|---|---|
| Image Manager | ✅ + | ✅ o | ✅ o |
| Foundation Package* | ✅ o | ❌ | ❌ |
| Barcode | ✅ o | ❌ | ❌ |
| Blob | ❌ | ❌ | ❌ |
| Color | ❌ | ❌ | ❌ |
| GEVServer | ❌ | ❌ | ❌ |
| GPUprocessing | ❌ | ❌ | ❌ |
| Manto | ✅ - | ❌ | ❌ |
| Minos | ✅ + | ❌ | ❌ |
| Movie2 | ✅ + | ❌ | ❌ |
| Polimago | ✅ + | ❌ | ❌ |
| Sil | ✅ + | ❌ | ❌ |
| ShapeFinder | ✅ - | ❌ | ❌ |

| BESCHREIBUNG | ANFORDERUNGEN |
|---|---|
| CVB.Net | .NET Framework 4.0 or Core 2.0 |
| CVB++ | C++11-capable compiler (vc14, gcc5) |
| CVBpy | Python (3.5, 3.6) |

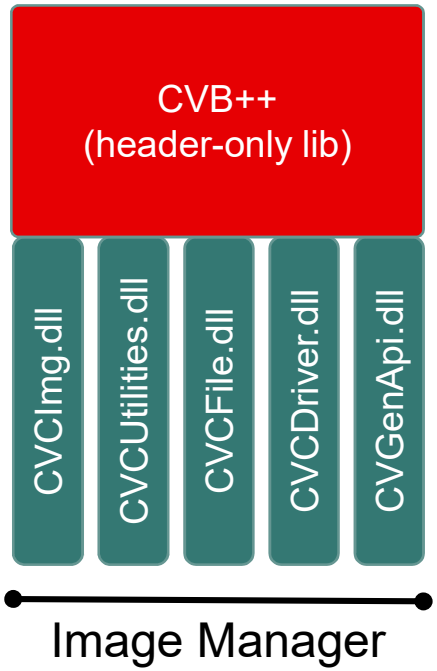
# CURRENT STATUS

- Structure of CVB.ET



# CURRENT STATUS

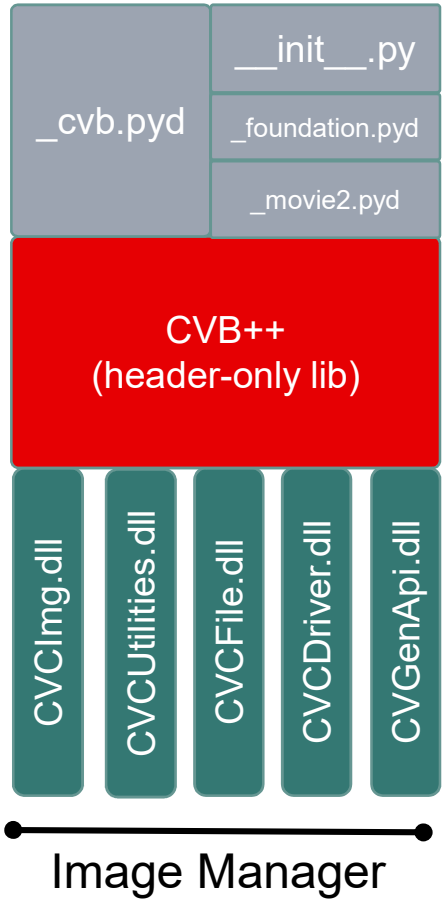
- Structure of CVB++




| | |
|---|---|
| N Cvb | Root namespace for the Image Manager interface |
| N Async | Convenience classes for asynchronous image acquisition |
| N Driver | Namespace for driver or device related operations |
| N ErrorCodes | Classic API error codes |
| N GenApi | Namespace for GenApi based device configuration |
| N UI | Namespace for user interface components |
| N Utilities | Namespace for helpers and utilities, which are not directly related to image processing |
| N SystemInfo | Namespace for helper functions related to system or CVB installation information |

# CURRENT STATUS

- Structure of CVBpy




**a**
AccessMode (cvb)
AcquisitionInterface (cvb)

**b**
BooleanNode (cvb)

**c**
CacheMode (cvb)
ColorModel (cvb)
CommandNode (cvb)

**d**
DataType (cvb)
Device (cvb)
DeviceControlOperation (cvb)
DeviceFactory (cvb)
DeviceImage (cvb)
DeviceUpdateMode (cvb)
DigitalIO (cvb)
DiscoverFlags (cvb)
DiscoveryInformation (cvb)
DiscoveryProperties (cvb)

**e**
EmuDevice (cvb)
EnumEntryNode (cvb)
EnumerationNode (cvb)

**f**
FloatNode (cvb)
FloatRegNode (cvb)

**i**
Image (cvb)
ImagePlane (cvb)
ImageRect (cvb)
IndexedStream (cvb)
IntegerNode (cvb)
IntRegNode (cvb)

**m**
ModuleLayer (cvb)

**n**
Node (cvb)
NodeMap (cvb)
NumberRepersentation (cvb)

**p**
PixelDataType (cvb)
PlaneNormalization (cvb)
PlaybackMode (cvb)
Point2D (cvb)
PortNode (cvb)

**r**
ReadWriteVerify (cvb)
Rect (cvb)
RegisterNode (cvb)
RingBuffer (cvb)
RingBufferImage (cvb)
RingBufferLockMode (cvb)

**s**
Size2D (cvb)
SoftwareTrigger (cvb)
StopWatch (cvb)
StopWatchMode (cvb)
Stream (cvb)
StreamImage (cvb)
StreamInfo (cvb)
StringNode (cvb)
StringRegNode (cvb)

**v**
ValueNode (cvb)
VideoDevice (cvb)
VinConnectionInformation (cvb)
VinDevice (cvb)
Visibility (cvb)

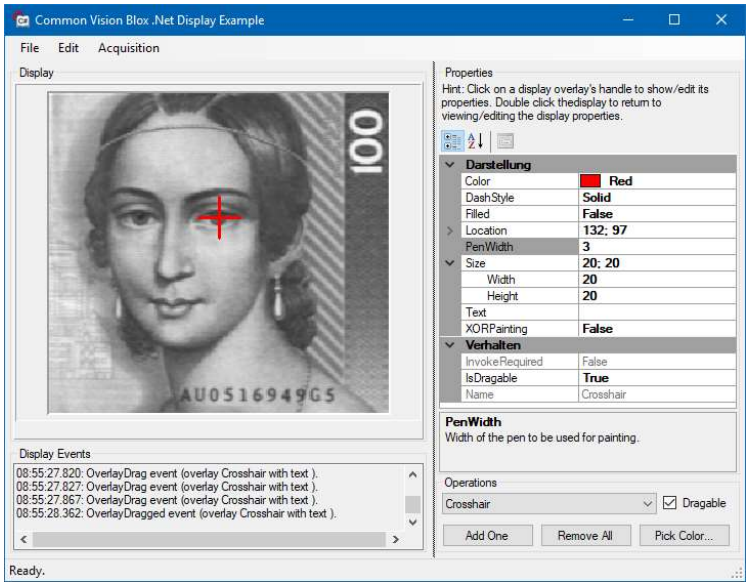
**w**
WaitStatus (cvb)
WrappedImage (cvb)

# EXAMPLES

- **CVB.NET**: Crosshair Overlay Plugin on Forms-Display (i.e. based on the *CVCDisp.dll*)

C-API (using CVDisplay.ocx):

```
axCVImage1.Load(@"%CVB%\Tutorial\Clara.bmp");
axCVDisplay1.Image = axCVImage1.Image;
Cvb.Image.PIXELLIST pixelList = Cvb.Image.CreatePixelList(3);
var tmp = new double[3];
tmp[0] = x;
tmp[1] = y;
Cvb.Image.AddPixel(pixelList, tmp);
tmp[0] = 20;
tmp[1] = 20;
Cvb.Image.AddPixel(pixelList, tmp);
Cvb.Plugin.TPenStylePlugInData tpenStylePlugInData =
   new Cvb.Plugin.TPenStylePlugInData(Cvb.Plugin.TPenStyle.SOLID, 3);
axCVdisplay1.AddOverlayObjectNET("Crosshair",
   "Hello world", canDrag, xorOnly,
   Cvb.Plugin.ColorToInt32(Color.Red),
   Cvb.Plugin.ColorToInt32(Color.Yellow), filled, id,
   pixelList.ToInt64(), tpenStylePlugInData.ToIntPtr());
Cvb.Image.ReleaseObj(pixelList);
```

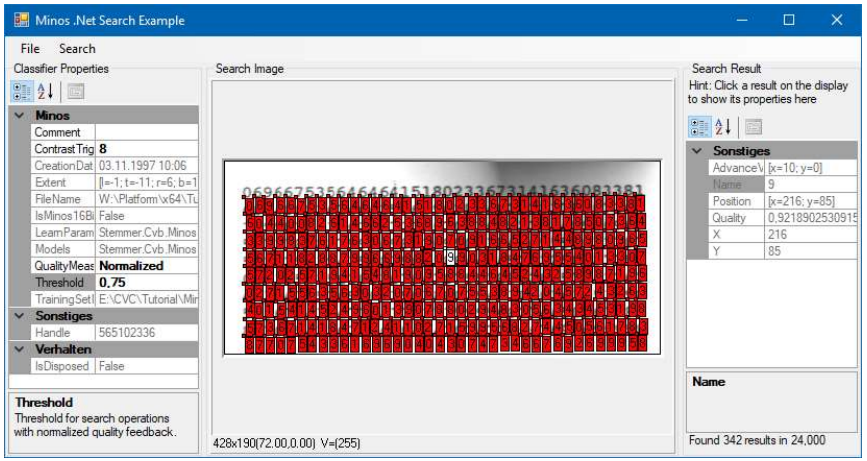



CVB.NET:

```
display.Image = new Stemmer.Cvb.Image(@"%CVB%\Tutorial\Clara.bmp");
display.Overlays.Add(
   new CrossHairOverlay("Hello world", canDrag, Color.Red,
   xorOnly, new Point(x, y), new Size(20, 20), DashStyle.Solid, 3));
```

# EXAMPLES

- **CVB.NET**: Show Minos search results




C-API (using CVDisplay.ocx):

```
TArea aoi;
MaxImageArea(reinterpret_cast<IMG>(m_cvDisp.GetImage()), aoi);
RESULTS results = nullptr;
SearchAll(clf, reinterpret_cast<IMG>(m_cvDisp.GetImage()), 0,
    density, aoi, locality, 255, results);
m_cvDisp.RemoveAllLabels();
for (int i = 0; i < SearchResultsCount(results); i++)
{
  double dQual, dXPos, dYPos, dDX, dDY;
  char* pstrName;
  cvbval_t lAID;
  SearchResult(results, i, dQual, dXPos, dYPos, dDX, dDY, pstrName, lAID);
  m_cvDisp.AddLabel(pstrName, FALSE, RGB(255, 0, 0), i,
      static_cast<long>(dXPos), static_cast<long>(dYPos));
}
ReleaseResults(results);
```

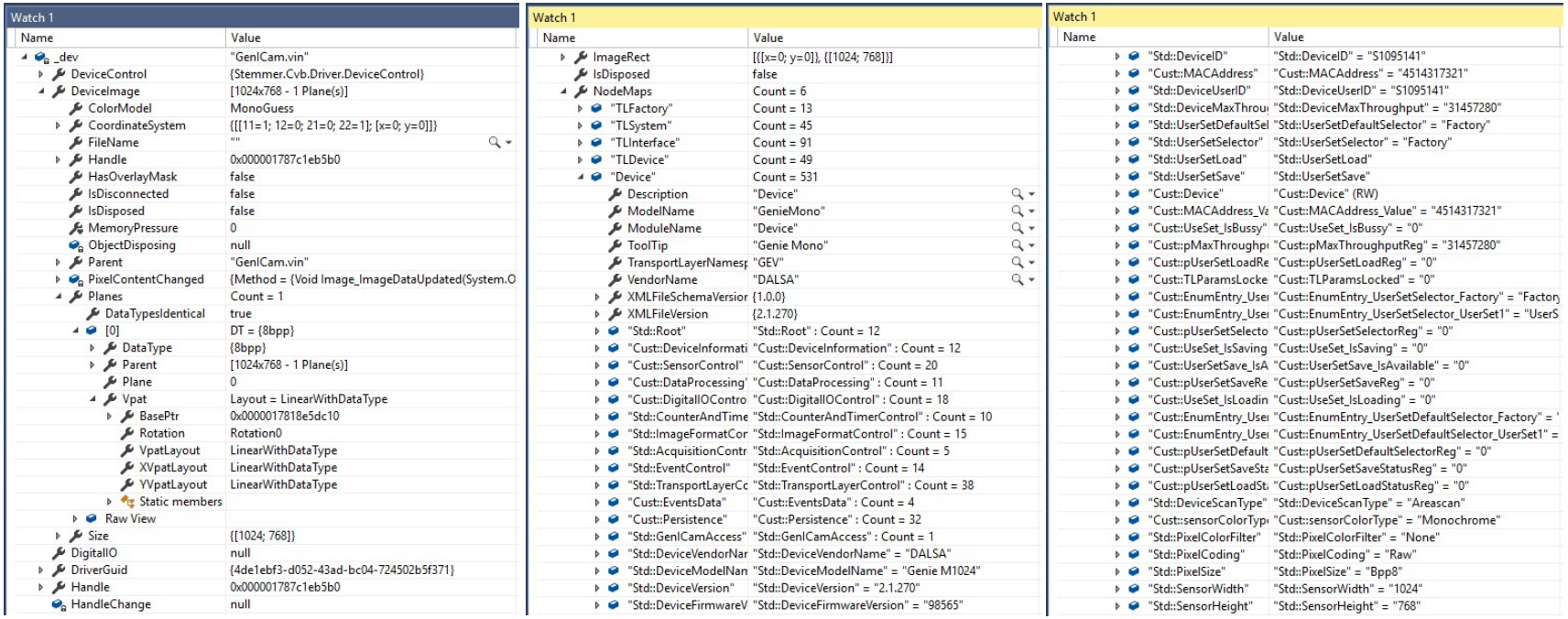
Additionally: All search result properties are directly visible in the debugger or e.g. in a property grid.

CVB.NET:

```
var results = clf.SearchAll(display.Image.Planes[0], locality);
display.Overlays.Clear();
foreach (var res in results)
  display.Overlays.Add(new DisplayLabel(res.Name, false,
      Color.Red, res.Position.ToPoint(), res));
```

# EXAMPLES

- **CVB.NET**: Detailed information about image and device properties (including all possibly available node maps) is accessible in the debugger (read- and writeable!)

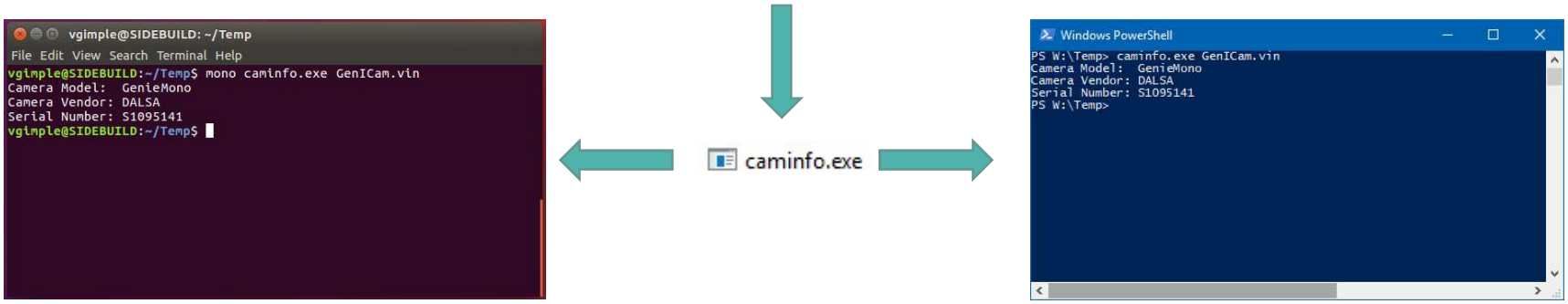



For comparison: With the C-API only *_dev.Handle = 0x0000017818e5dc10* would be visible.

# EXAMPLES

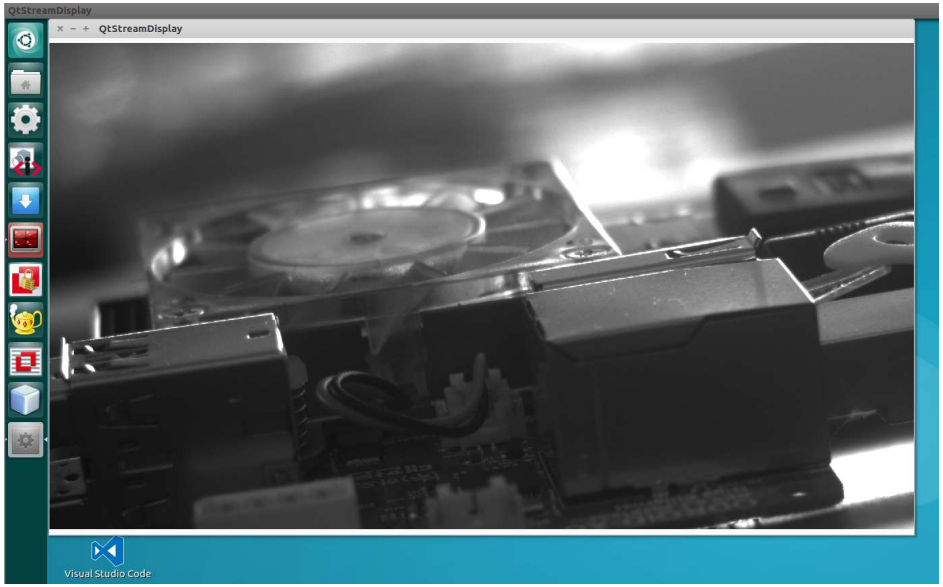
- **CVB.NET**: Binaries are usable cross platform as long as they do not need a GUI. Currently this feature is limited to Windows (Win32 and x64) and 32 bit Linux (Intel & ARM architecture); extension to 64 Bit (Intel & ARM) to follow later on…

```
static void Main(string[] args)
{
  try {
    var dev = DeviceFactory.Open(args[0]);
    if (dev.NodeMaps.Count == 0) {
      Console.WriteLine("No node map.");
      return;
    }
    var nodeMap = dev.NodeMaps[NodeMapNames.Device];
    Console.WriteLine("Camera Model:  " + nodeMap["ModelName"]);
    Console.WriteLine("Camera Vendor: " + nodeMap["VendorName"]);
    Console.WriteLine("Serial Number: " + nodeMap["Std::DeviceID"]);
    dev.Dispose();
  }
  catch (Exception ex) {
    Console.WriteLine("Operation failed. Hint: " + ex.Message);
  }
}
```



# EXAMPLES

- **CVB++ & Qt**: Cross platform capable applications with GUI become possible (all currently supported platforms)

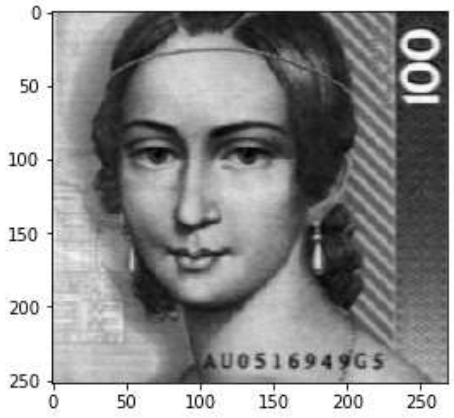



```cpp
int main(int argc, char* argv[])
{
  try
  {
    QApplication app(argc, argv);

    Cvb::String path(CVB_LIT("%CVB%/drivers/GenICam.vin"));

    // expand environment variables in path
    path = Cvb::ExpandPath(path);

    // open a device
    auto device = Cvb::DeviceFactory::OpenDevice(path);

    // connect the device image to the UI
    Cvb::ImageView view;
    view.Refresh(device->DeviceImage(), Cvb::AutoRefresh::On);
    view.show();

    // get the first stream of the device
    auto stream = device->Stream();

    // create an acquisition handler for that stream and run it
    auto streamHandler = Cvb::SingleStreamHandler::Create(stream);
    streamHandler->Run();

    return app.exec();
  }
  catch (const std::exception& error)
  {
    std::cout << error.what() << std::endl;
  }
}
```

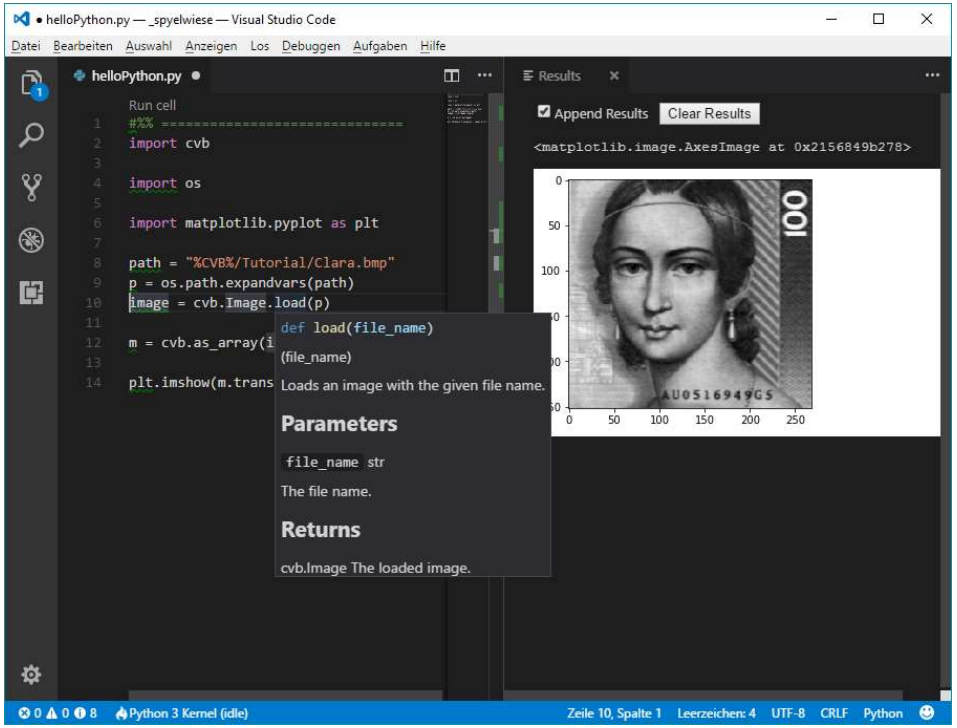
# EXAMPLES

- **CVBpy**: Usable in all commonly used Python environments. Visualization via pyplot.

```
In [1]: import cvb

In [2]: import os

In [3]: image = cvb.Image.load(os.environ["CVB"] + "/tutorial/Clara.bmp")

In [4]: matrix = cvb.as_array(image)

In [5]: import matplotlib.pyplot as plt

In [6]: plt.imshow(matrix.transpose(), cmap="gray")
Out[6]: <matplotlib.image.AxesImage at 0x8299470>

In [7]: plt.show()
```

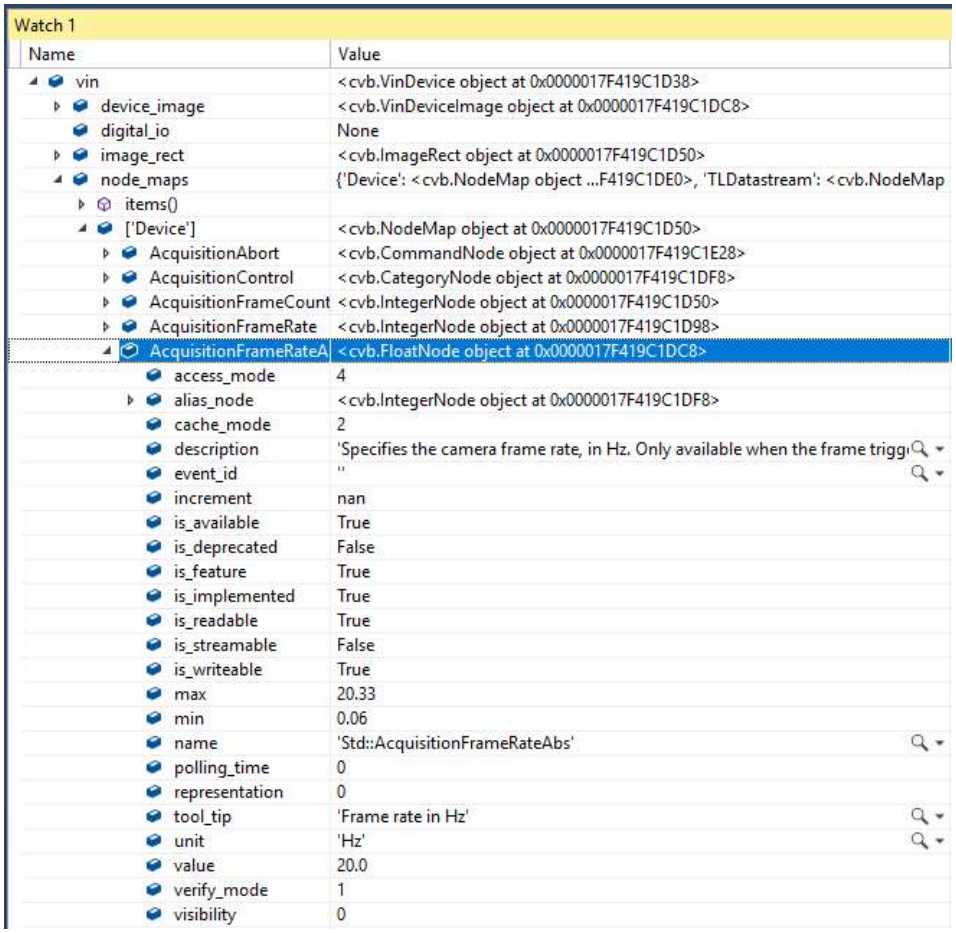



```
In [8]:
```



# EXAMPLES

- **CVBpy**: Similar to CVB.Net the image and device information is directly visible in the debugger.



STEMMER® IMAGING

# DEMO

- **CVBpy**: IPython
  REPL Read-eval-print-loop

# FUTURE PROSPECTS

**The next steps for CVB.Net, CVB++ and CVBpy**


- Increase the feature and test coverage for all three programming languages.
- Extend the binary compatibility for CVB.Net to the 64 Bit Linux platforms (probably in CVB 14) and adapt to the .NET Core 2.0 Standard.
- Establish a cross platform available display possibility for CVB.NET.

► Enough work to be done until the next CVB User Group Meeting

- *The C-API will **not** be made obsolete by these efforts! It rather will continue to be the basis for CVB.NET, CVB++ and CVBpy und will be maintained and supported accordingly!*

# THANK YOU VERY MUCH FOR YOUR ATTENTION

**Your contact:**
Andreas Rittinger

STEMMER IMAGING GmbH
+49 89 80902-745
a.rittinger@stemmer-imaging.de
https://www.stemmer-imaging.de
https://forum.commonvisionblox.com