Unibrain

Fire-i.net SDK manual

Programmer's Manual & Reference

Table of Contents

Introduction

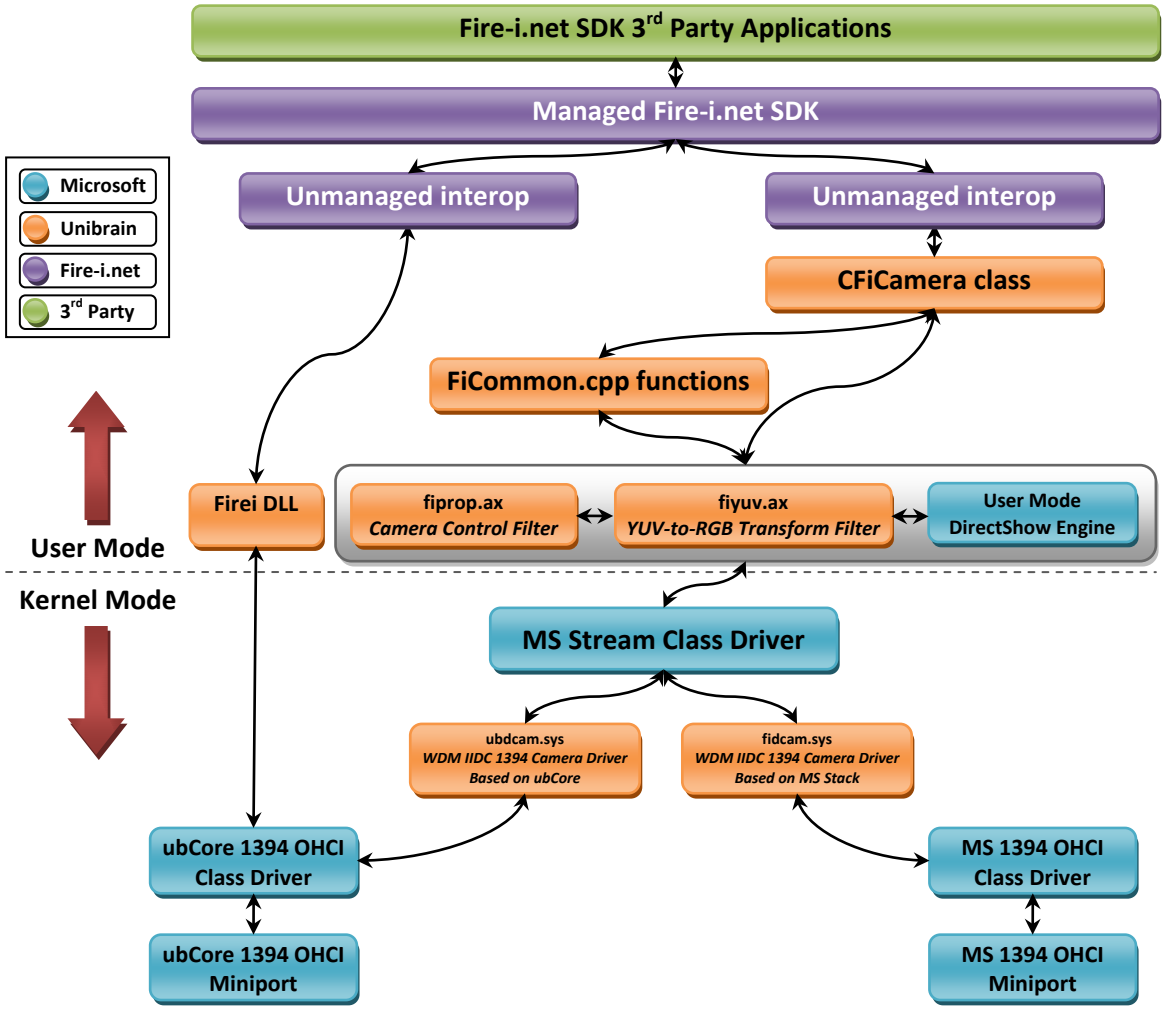
The Fire-i.net SDK is the latest addition to ubCore™ and the Unibrain APIs. It provides an entirely new and comprehensive way to interact with multiple IEEE1394 IIDC compatible cameras, including video and feature manipulation, direct register access, etc. Even though it is designed to follow in the footsteps of the Unibrain Fire-iX SDK, it is based on Microsoft™ .net™ technology, written entirely in C#, making a perfect fit for any managed .net project.

The Fire-i.net SDK can operate in two modes; ubCore and DirectShow. The SDK feature set is almost identical and transparent under both modes of operation. It does not require familiarity with either of the two SDKs, so it is an ideal fit in situations where the programmer does not need/want to delve into IEEE1394 or IIDC details in order to perform simple camera tasks, such as setting a few features and capturing a few frames. This is only on the surface though; the more powerful features are still there for anyone to use, they are just not required for the simpler tasks.

Technical Details

Architecture

The following illustration shows how the various Unibrain APIs interact with each other, and how they are abstracted from the programmer by using the Fire-i.net SDK.



Performance

The performance of the program at runtime depends on which underlying API is selected (Firei.dll, DirectShow/ubCore or DirectShow/MS Stack). There is no specific set of circumstances where selecting one API over another will produce better results. The ease of changing between all three, which requires minimal changes[1], allows the programmer to test through all three different cases and compare the performance. Please keep in mind however, the performance using the MS Stack DirectShow drivers will be affected by the

[1] A single parameter indicating whether Firei.dll or DirectShow is going to be used is passed to the FireiManager constructor; no code difference to choose between DirectShow/ubCore and MS Stack (ubSwitch is used in the latter case).

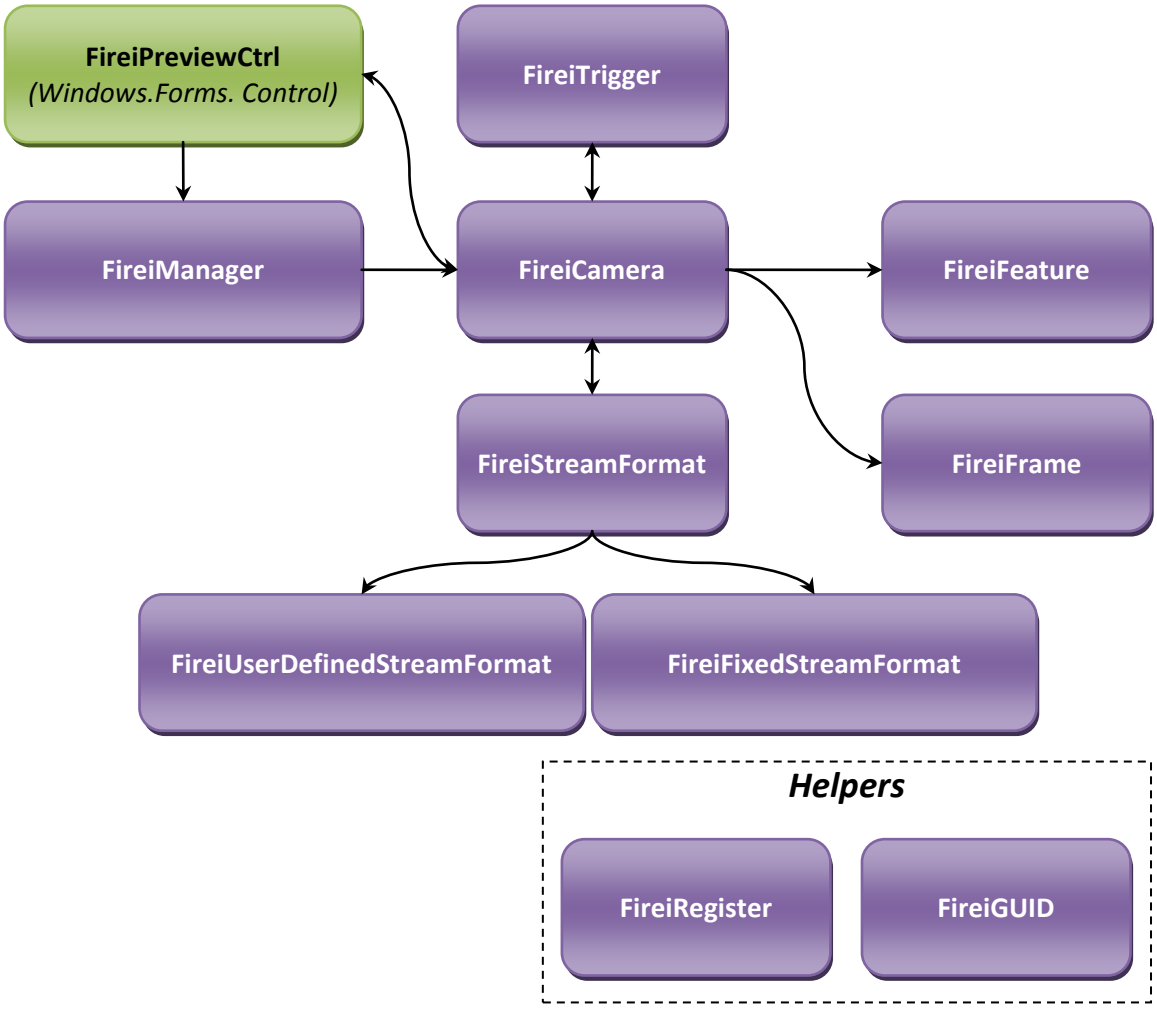
MS 1394.sys driver, which is known to have various issues with popular operating systems (Windows XP Service Pack 2 and Windows Vista included).

Installation

No specific installation of Fire-i.net SDK is necessary. It is included with both ubCore Setup and Firei MS Stack Setup. The unmanaged portion of the API is automatically installed along with the other unmanaged dlls of ubCore of Firei. It is therefore necessary to have either ubCore or Fire-i MS Stack installed and in working order for Fire-i.net SDK to operate. The (managed) Firei.net.dll module (which contains the full functionality of the Fire-i.net SDK) is registered with the GAC (Global Assembly Cache) automatically during these installations. Also during installation, a registry entry is inserted making the Fire-i.net API directly available in Microsoft Visual Studio™ for use in .net projects.

Design

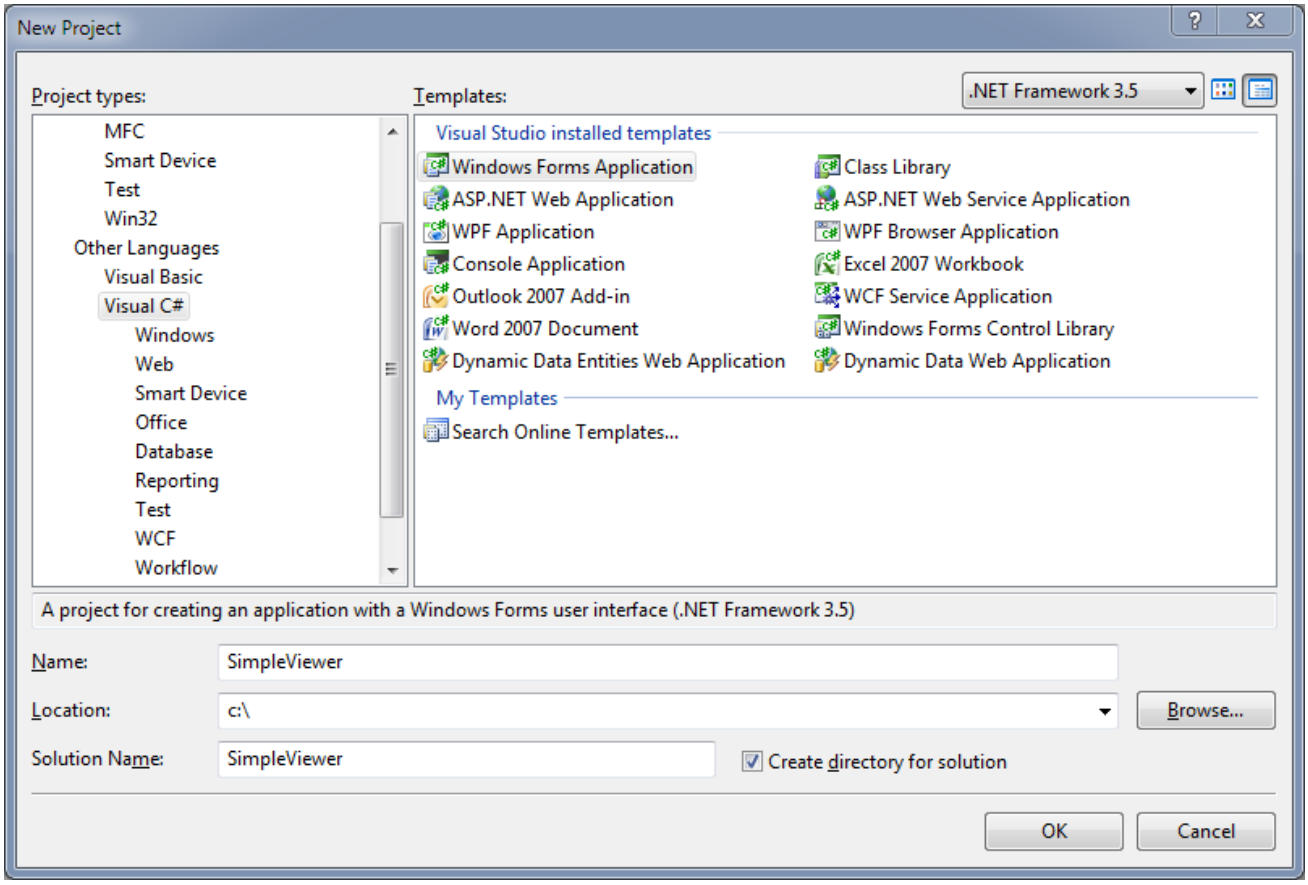
The Fire-i.net SDK is designed as a Windows control, bundled with a few managed objects that help control the action, each with its own interface, properties and methods. The programmer either begins with the control (for use on a GUI application) and then uses it to create the other objects, or directly constructs the objects (for use on a command-line based application), omitting the Windows control altogether. The following diagram attempts to depict how the various objects interact with the Windows control and with each other:



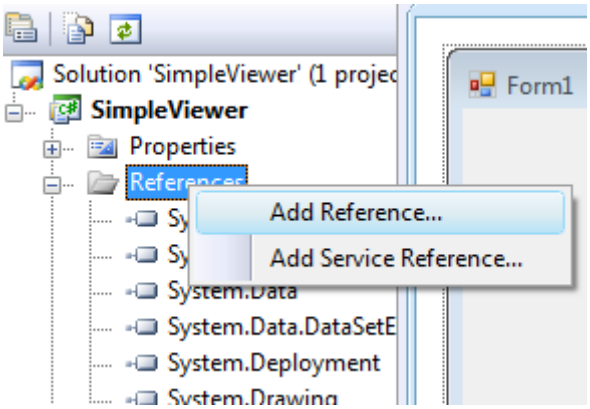
Usage

Scenario I – Simple video viewer

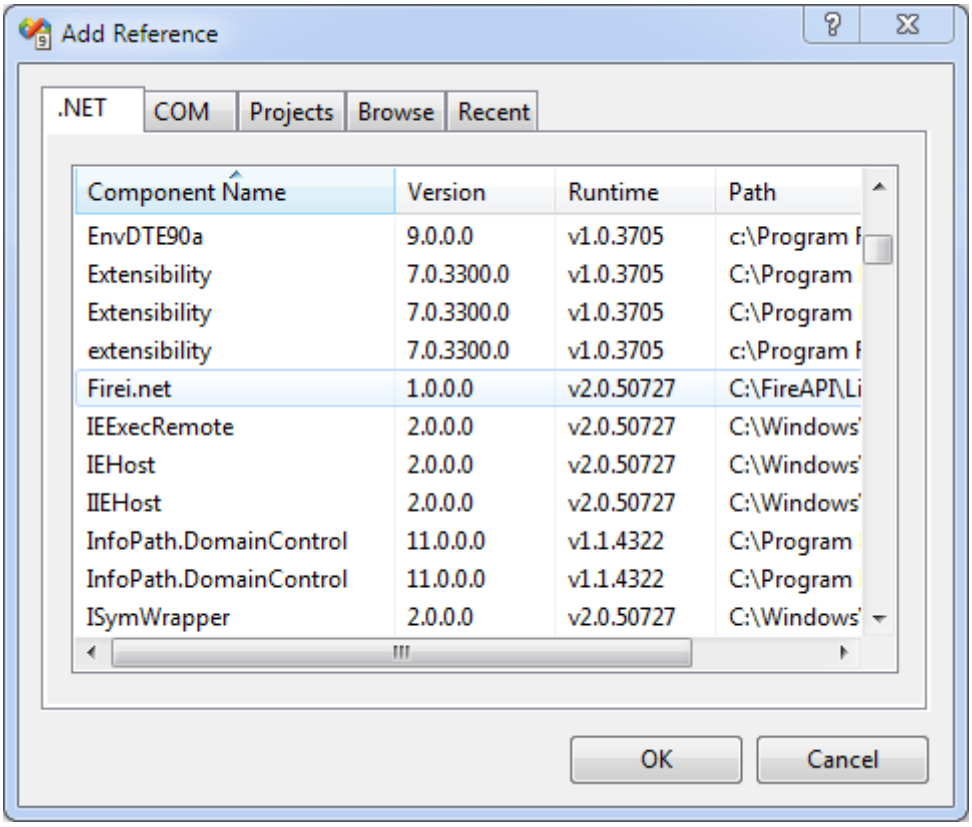
Let's start Microsoft Visual Studio 2008. Create a new empty project, of the "Windows Forms Application" variety:



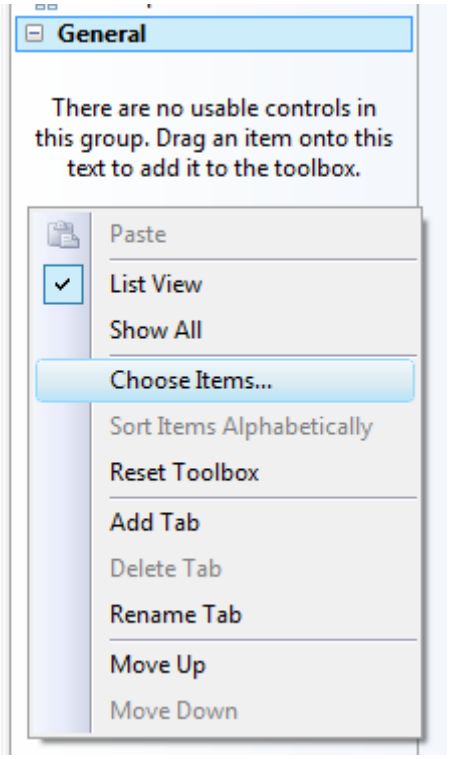
This should produce a new empty project, with a single form (empty as well). Now, right-click on the References folder of the project and select "Add Reference…":



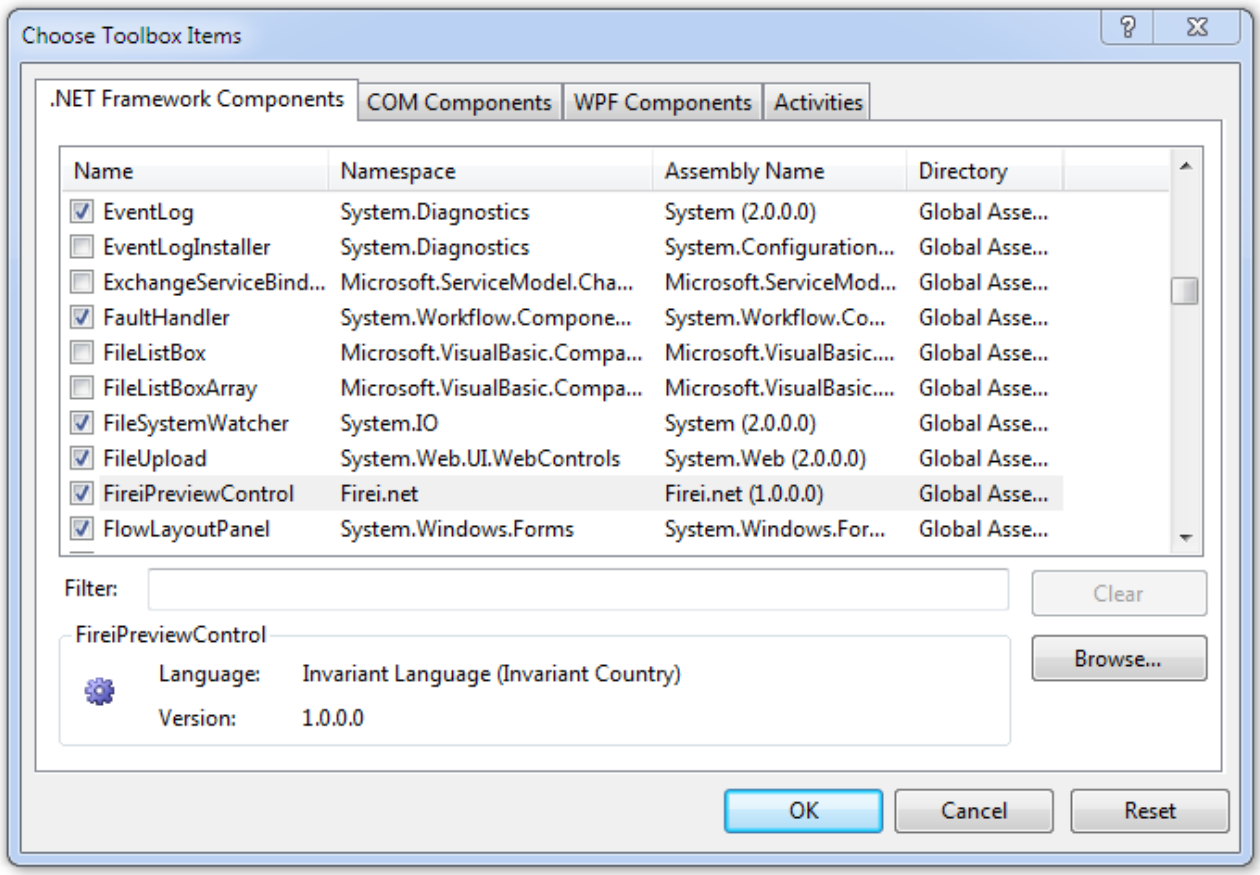
This will bring up the reference selection dialog of VS. In the ".NET" tab, select "Firei.net" and click on OK:



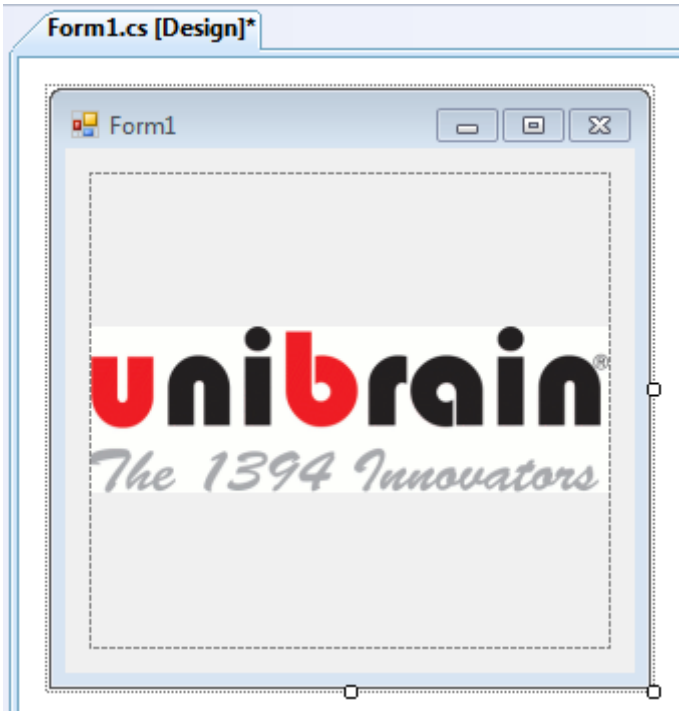
This will add the Fire-i.net assembly to the project references. Next, we need to add the FireiPreviewControl in our toolbox. To do that, right-click on the empty space of the current toolbox, and select "Choose Items…":



This will bring up the “Choose Control Items” dialog of Visual Studio. On the “.NET Framework Components” tab, select and mark the FireiPreviewControl entry:



As soon as OK is pressed, the FireiPreviewControl is added to the toolbar. Let’s use it on our empty form:



Visual Studio automatically sets the name of the control to be “fireiPreviewControl1”, which for our purpose will do nicely.

Now it’s time to edit the form code. First, we need two variables, so we add them at the very top:

```
private FireiManager Manager;
private FireiCamera Camera;
```

The `FireiManager` object will be used to construct our camera object. The `FireiCamera` object exposes all the features that our camera supports. We need to initialize our Manager object before using it. This can be done by calling:

```
Manager = new FireiManager(true);
```

The constructor parameter indicates whether DirectShow (true) or DLL (false) mode will be employed internally. It is highly recommended that only one instance of a FireiManager object is constructed during the program lifetime, ensuring that either the DirectShow or the DLL mode will be used throughout. Mixed mode is not supported.

We override the OnShown method to place our initialization code:

```
protected override void OnShown(System.EventArgs e)
{
      base.OnShown(e);
      Manager = new FireiManager(true);
}
```

Now we need to construct our camera object. `FireiManager` provides two methods for doing just that; `GetCameraFromIndex` and `GetCameraFromGUID`. The former is useful when a) we have only one camera connected to our system or b) we want to programmatically iterate between all the connected cameras, and select the one we want[2]. To simply construct the first (or only) camera connected to our system, a call

```
Camera = Manager.GetCameraFromIndex(0);
```

will suffice. To iterate between all cameras, we can use `GetConnectedCamerasCount` and proceed with a regular for-loop:

```
for (int i = 0; i < Manager.GetConnectedCamerasCount(); ++i)
{
    Camera = Manager.GetCameraFromIndex(i);
    if (Camera.Vendor == "Unibrain")
        break;
}
if (Camera == null || Camera.Vendor != "Unibrain")
    Close();
```

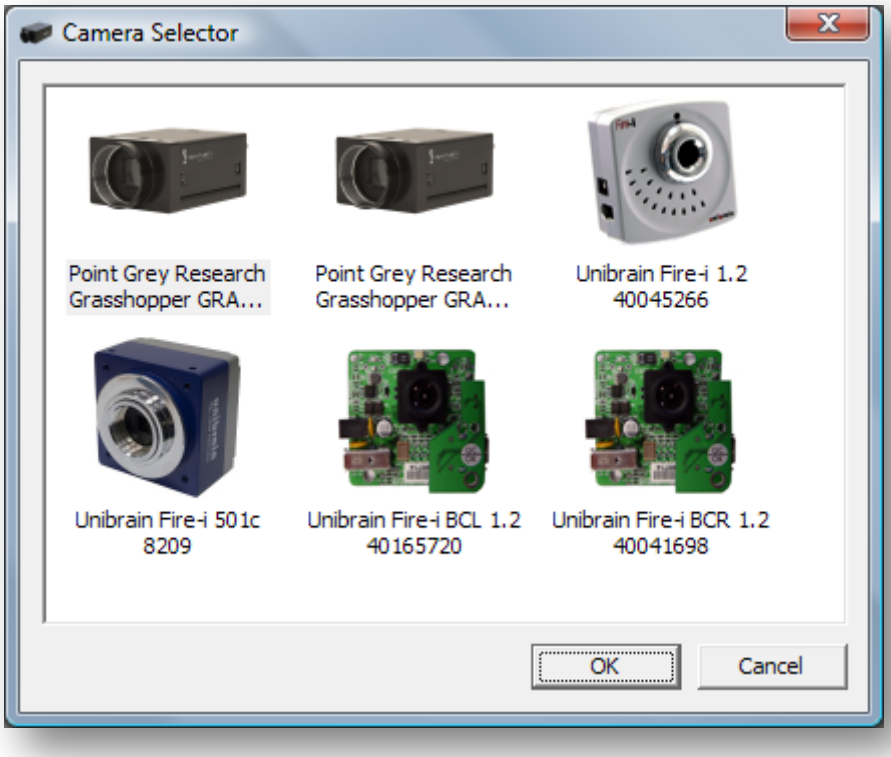
[2] Please keep in mind that the order in which the cameras will be indexed is non-deterministic. It can be considered to be random even on the same system with the exact same configuration between reboots.

The above code fragment will iterate through the connected cameras, and stop at the first camera that is made by Unibrain. With the additional check at the end, it will effectively select the first available Unibrain camera.

If the camera GUID (unique identifier) is known beforehand, we can use the `GetCameraFromGUID` method of `FireiManager`. We construct a `FireiGUID` object, initialize it with our known GUID, and then construct the camera with `GetCameraFromGUID`:

```
FireiGUID guid = new FireiGUID();
guid.InitFromString("08:14:43:61:02:63:0A:D2");
Camera = Manager.GetCameraFromGUID(guid);
```

Additionally, `FireiManager` allows choosing a camera using a GUI through the `SelectCamera` method. This method will present a camera selection dialog, and if the user selects one, it will return the camera's GUID:



For simplicity, we will follow this path in this example. So, after our `FireiManager` construction, we add:

```
FireiGUID guid;
if (!Manager.SelectCamera(out guid))
    Close();
```

If the user didn't select a camera, the `SelectCamera` call will return `false`, in which case we exit the application. The `FireiGUID` object is constructed inside the `SelectCamera` method, and it is then used to create the camera object:

```
Camera = Manager.GetCameraFromGUID(guid);
```

Next, we need to connect the `Camera` object to our `FireiPreviewControl` object. This is done with a single call:

```
Camera.AttachPreviewCtrl(fireiPreviewControl1);
```

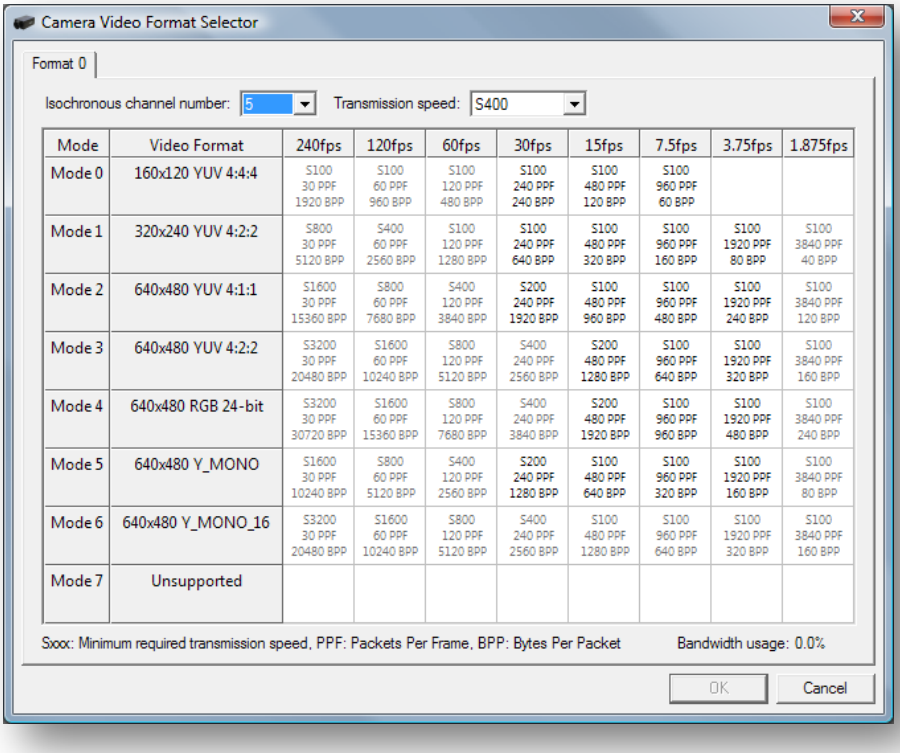
Effectively it means that any video this camera will produce will end up being shown in this preview control.

Now we need to select which streaming format our camera will run at. This can be done programmatically by either accessing the currently selected streaming mode (through the `StreamFormat` property of `FireiCamera`), or by accessing the supported streaming formats generic list, through the `GetStreamFormats` method. For the purposes of our simple example though, we will go down a different path, the GUI-driven one.

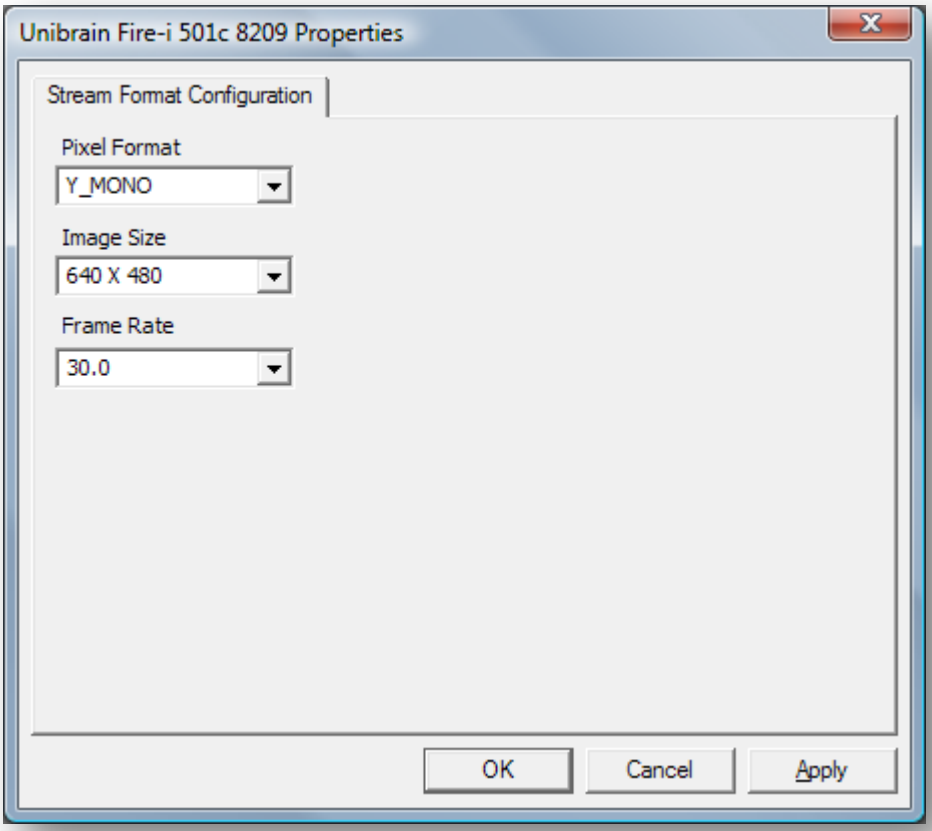
We add a call to `SelectStreamFormat`:

```
Camera.SelectStreamFormat();
```

This will allow selecting and setting a streaming format through the API provided GUI:



The available choice will of course vary depending on which camera this is called upon; only supported formats will be presented. Also, the above selection dialog shown is the one provided by the Firei.dll API. Using the DirectShow API will show a format selection dialog similar to:



Now that the streaming format is selected, the only part remaining is actually starting the camera. This requires a simple call:

```
Camera.Run();
```

Some cleanup is also necessary though, according to best programming practices. Before exiting, we'll stop the camera and dispose of our two constructed objects. This can be done at the `OnClosed` override:

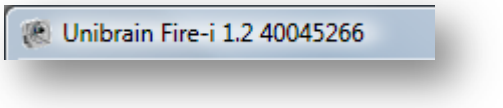
```
protected override void OnClosed(EventArgs e)
{
    if (Camera != null)
    {
        Camera.Stop();
        Camera.Dispose();
    }
    Manager.Dispose();
    base.OnClosed(e);
}
```

Running the program will show live video on our control on the form, after selecting the desired camera and streaming format. If only one camera is connected on the system, it is selected without presenting a selection dialog at all (and the `SelectCamera` method returns `true` immediately).

A nice touch to our program: Set the caption and the form icon to match our selected camera:

```
Text = Camera.ToString();
Icon = Camera.Icon;
```

This will change the form caption to look like:



NOTE: With DirectShow enabled and ubCore installed, by using ubSwitch (the ubCore tool) we can toggle between ubCore DirectShow support and MS-Stack DirectShow support.

Scenario II – Controlling camera features

Now suppose we need to change some camera features, like brightness or shutter speed. Again, there are more than one ways of achieving that. Each feature for example can be accessed individually through its `FireiFeature` derived object; we can access that object by a simple direct call to the corresponding camera property, like:

```
FireiFeature feature = Camera.Shutter;
```

Another way to access a feature would be by name, using the `GetFeature` method of `FireiCamera`:

```
FireiFeature feature = Camera.GetFeature("Shutter");
```

Suppose we want to change the shutter value to its maximum supported. Now that we have the Shutter feature object, this can be done by:

```
if (feature.IsSupported)
    feature.Value = feature.MaxValue;
```

Note that the above code fragment checks whether the camera supports the feature before accessing any of its properties. Failure to do so would throw an exception if the feature is not supported. Also, if the Shutter feature is set to Auto, we'll need to turn that off before setting the value, or an exception will be thrown as well.

Yet another way to access the camera features is through the `GetFeatures` method provided by `FireiCamera`. This allows iterating through all the supported camera features, like:

```
Camera.
GetFeatures(true, FireiFeatureGroup.All).
FindAll(f => f.HasAuto).
ForEach(f => f.AutoMode = true);
```

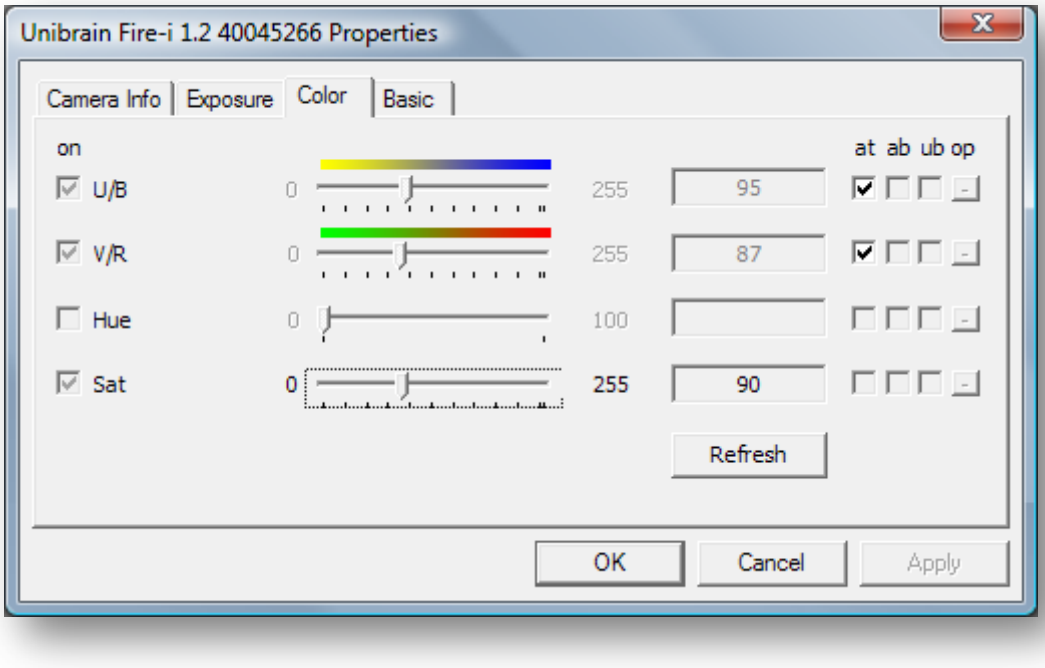
As can be seen from the above example, advanced .net features like lamda expressions are fully supported. The above example gets a generic list of all supported features (for additional information on `GetFeatures`,

see the reference section of this manual) of the camera, checks whether they support Auto setting and if they do, sets it.

A GUI-driven way to manipulate the camera features is also provided. A call to the `DisplayProperties` method of `FireiCamera` will suffice:

```
Camera.DisplayProperties();
```

This brings up the camera properties sheet:



NOTE: The call to `DisplayProperties` returns immediately after opening the dialog. Therefore it is possible to manipulate the camera features while observing the results in the preview window. There is no additional functionality that can be accessed through the GUI feature manipulator; everything that can be done through the GUI, can also be done programmatically. For example, the Shutter feature discussed above resides in the “Exposure” tab. The “AutoMode” setting is the one marked “at” with a checkbox. The slider control corresponds to the value setting.

For the purposes of our example, we’ll add a listbox to our Scenario I sample application, containing the names of all the features that are supported by the camera, and a button that brings up the camera properties.

Firstly, we enlarge our form, to make way for the new controls:

Then, we add the two controls. It is also a good idea to change their corresponding variable names to `FeatureList` and `Properties` respectively:

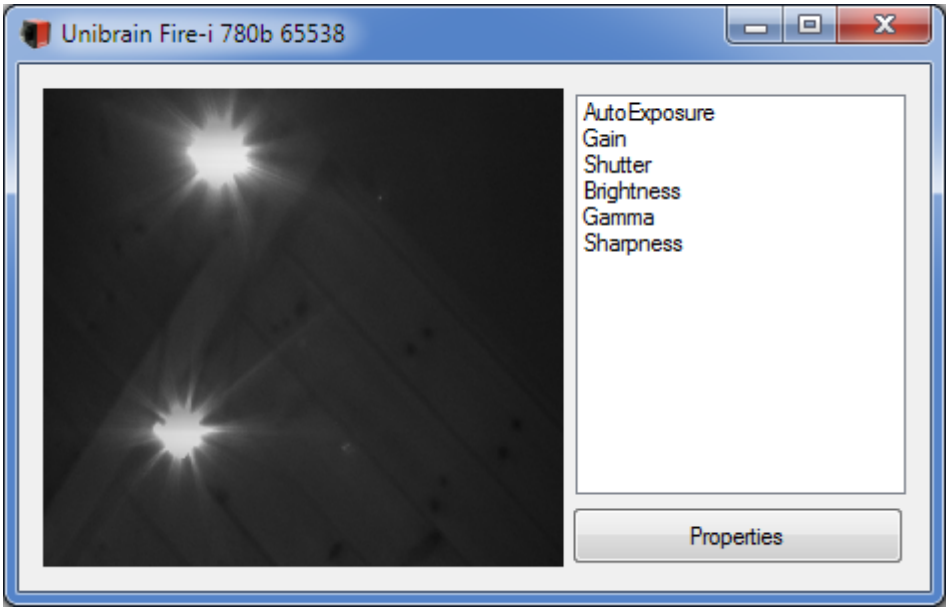
The best time to populate the listbox would be immediately after we constructed our `FireiCamera` object. As discussed above, iterating through the supported features is done with the `GetFeatures` method:

```
Camera.
GetFeatures(true, FireiFeatureGroup.All).
ForEach(f => FeaturesList.Items.Add(f.Name));
```

This takes care of the listbox population. Now for the properties button, we override the Properties_Click event:

```
private void Properties_Click(object sender, EventArgs e)
{
      if (Camera != null)
            Camera.DisplayProperties();
}
```

Running the program produces the desired result:



Most cameras will reset to their default feature values if powered off. The SDK provides two ways to save and restore them; to/from an XML file through the `LoadFromXML` and `SaveToXML` methods of `FireiCamera` and, if the camera supports memory presets, through the `SaveToMemory` and `LoadFromMemory` methods. Note that the XML formatting of the file is compatible with the other tools of ubCore. So for instance, the programmer can use Fire-i Application or FireIIDC to select and then save a feature set, and then load it programmatically in a Fire-i.net program.

Scenario III – Manipulating captured frames

Now that we have our simple viewer program up and running, watching the live video preview, let's take it a step further: we can manipulate the camera frames as they arrive.

In order to do that, we need to override the `FrameReceived` event of `FireiCamera`. First, we create the event callback method:

```
void Camera_FrameReceived(object sender, FrameReceivedEventArgs e)
{
}
```

Next, we assign that method to the FrameReceived event. A good place for that would be immediately before calling Camera.Run:

```
Camera.FrameReceived +=
      new EventHandler<FrameReceivedEventArgs>(Camera_FrameReceived);
```

As shown above, the event provides a FireiFrame object for our perusal (via the Frame property of the FrameReceivedEventArgs). Through this object we have absolute access to the frame buffer, pixel by pixel, with the GetPixel/SetPixel and GetRGB/SetRGB methods. The only difference between the two sets is the way the color value of the pixel is passed; in the former it is passed as a Windows.Drawing.Color object, in the latter it is passed with R, G and B values separately. The same designation holds true for the other methods of FireiFrame; whenever "RGB" is in their name, it means separate R, G and B values.

Besides direct per-pixel manipulation, a few additional methods are implemented in FireiFrame. Suppose we'd like to draw a blue rectangular box, along the edges of the frame, and a big red X inside it. We can use the DrawRectangleRGB and DrawLineRGB methods to achieve that:

```
short x1 = 10;
short y1 = 10;
short x2 = (short)(Camera.StreamFormat.Width - x1);
short y2 = (short)(Camera.StreamFormat.Height - y1);
e.Frame.DrawLineRGB(x1, y1, x2, y2, 255, 0, 0);
e.Frame.DrawLineRGB(x2, y1, x1, y2, 255, 0, 0);
e.Frame.DrawRectangleRGB(x1, y1, (short)(x2 - x1), (short)(y2 - y1), 0, 0,
255, false);
```

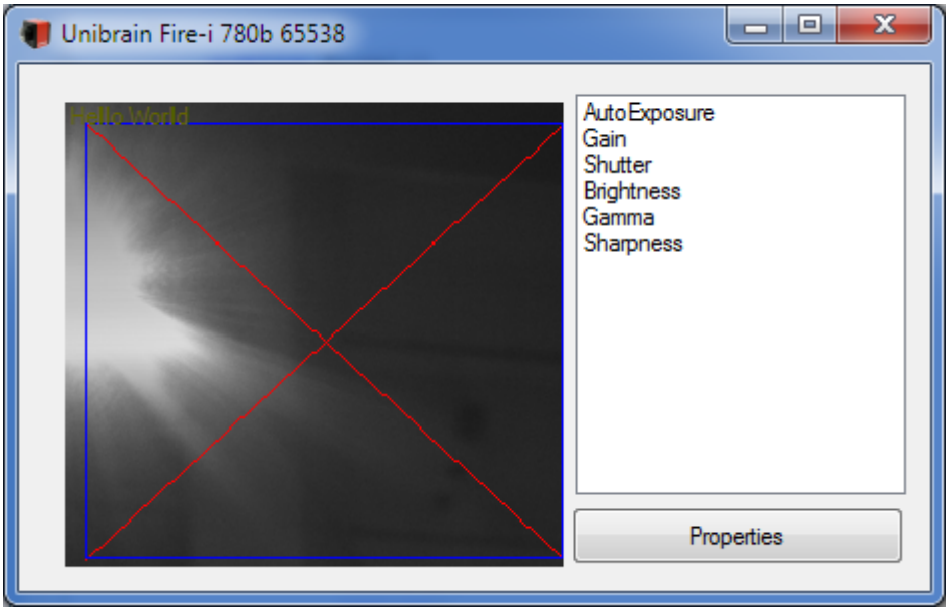
Note the way x2 and y2 are calculated: we use the Width and Height properties of the selected streaming format to ascertain the edges of our frame. The "false" in the last parameter of DrawRectangleRGB, means "empty" as opposed to "filled" box.

Now suppose we'd like to write some text on the frame as well. We can use the DrawStringRGB method for this:

```
e.Frame.DrawStringRGB("Hello World", 0, 0, Font, 100, 100, 0);
```

A yellowish color is chosen this time, using the default font of our form, placed at the top left corner of the frame.

This is what the outcome looks like:



NOTE: The aliasing of the drawn lines and text is due to the enlargement of the picture in order for it to fit in our selected preview control. The actual streaming format that was selected for the above screenshot was 252×232, which is more than the size of the window. Similarly, if the format was smaller than the screen, the image would have been enlarged to fit accordingly. Also, to nitpick, there is no test made whether the image is actually larger than 20×20, as would be required for the box and lines to fit. In case the image was smaller than 20×20, an exception would be thrown trying to call `DrawLineRGB` with positions out-of-bounds.

Additionally, we can save the frame at any time to a Windows BMP compatible file on disk. To do that, simply call the `SaveToFile` method of `FireiFrame`:

```
e.Frame.SaveToFile("C:\\1.bmp");
```

The placement of the `SaveToFile` call is important. If it is placed before any changes on the frame, the file will contain the frame as it is, whereas if it is placed after any changes, it will contain them too. The resulting file will have the Width × Height size as sent by the camera, not as shown on the screen. So for instance, in this case it would be 252×232, regardless of the size of the window showing the video.

NOTE: Please exercise caution when using the `SaveToFile` method. If a high frame rate streaming format is selected, especially with a high resolution, the overhead on the system saving individual uncompressed BMP files in quick succession can be extremely taxing. For example, a 1280×960×24bit image has a size on the disk of roughly 3.52MB. At 30 frames per second, the throughput required would be above 100MB per second, which is prohibitive in most real-world cases. In such instances it is preferable not to save *every* frame captured but every *n*-th frame to reduce overhead – or simply use a lower frame rate.

Scenario IV – Working with a ROI (a.k.a. Format 7)

Some cameras support capturing a specific ROI (Region Of Interest), through what is defined in the IIDC specification as “Format 7” or “Partial Image Format”. The Fire-i.net SDK has full support for those modes of

operation, by using a derivative of the FireiStreamFormat object, called FireiUserDefinedStreamFormat (as opposed to the FireiFixedStreamFormat).

The SelectFormat method discussed in Scenario I allows such selection, but this time we'll go a different way, and manipulate the streaming format programmatically.

First, we need to iterate through all available streaming formats of the camera. We do that through the GetStreamFormats method, in a similar way to iterating through the camera features:

```
FireiStreamFormat streamFormat = Camera.
      GetStreamFormats().
      Find(s => s.IsUserDefined);
if (streamFormat == null)
{
    Close();
    return;
}
ushort w = streamFormat.UserDefined.MaxWidth;
ushort h = streamFormat.UserDefined.Height;
streamFormat.UserDefined.SetROI(0, 0, w, h);
Camera.StreamFormat = streamFormat;
```

In the above code fragment, a generic list of supported streaming formats is requested from the camera, and the first User Defined stream format is selected through the Find method. If the camera doesn't support any User Defined formats, Find will return null – in which case we exit the program. Since we ascertained that StreamFormat in fact is valid, and is of User Defined type, we can use the UserDefined property of FireiStreamFormat to access the User Defined properties of the format. In this case we assign in the w and h ushort variables the maximum allowed width and height of the format respectively and then set it as our selection, using the SetROI method. If an invalid rectangle was specified as the User Defined size the SetROI method would throw an exception. Finally, we set the now updated StreamFormat to the Camera, via its StreamFormat property – we could also have done this through the Save method of the FireiStreamFormat itself.

In fact, the UserDefined property of FireiStreamFormat returns a variable of type FireiUserDefinedStreamFormat. In case the FireiStreamFormat is Fixed instead of User Defined, the corresponding property is called Fixed accordingly and returns a variable of type FireiFixedStreamFormat. Even though a Fixed format carries that name, it contains a single variable property – the frame rate (it can be set using its FrameRate property).

All formats, whether Fixed or User Defined, have two attributes in common: their FireiResolution and their FireiPixelFormat. Both of these attributes are enumerated values – in the case of FireiResolution, the name is sort of a misnomer; if the stream format is User Defined it is actually valued from Variable to Variable_7 which are roughly equivalent to the Format_7 Modes of the camera (unlike for Fixed formats, which are valued from res160x120 to res1600x1200). This pair of values is unique to a stream format, so it is sufficient to define the format.

Since the above situation is not always easy to work with, another way to identify a streaming format exists: a unique identifier, which can be stored in a uint value. This is accessed through the Identifier property

of `FireiStreamFormat`. This identifier can be used to retrieve this specific stream format from the camera, using its `RetrieveStreamFormatFromIdentifier` method. Keep in mind though that any user selected settings are not stored – it is the stream format that is retrieved, not its contents.

Reference

In the following reference of all interfaces implemented, “Long” is defined as a 32-bit signed integer value, and “Integer” is defined as a 16-bit signed integer value. “Byte” on the other hand is an 8-bit unsigned integer value. “Single” is a single-precision floating-point number.

FireiPreviewControl

The FireiPreviewControl is a Windows.Forms.Control-derived control, which purpose is to show the video output of the underlying API at work. It can be resized at will while running, and can be attached on any Windows form.

FireiGUID

The FireiGUID object is a helper object, making the camera GUID easier to manipulate, store and retrieve.

It can be constructed directly, or through the `SelectCamera` method of `FireiManager`. It supports a number of properties and methods through its interface:

[] property

Prototype

```
public byte this[int index]
```

Comments

The `[]` property is a read/write property, enabling the direct manipulation of any of the 8 bytes in a `FireiGUID` object.

Visual Basic syntax

```
Dim Byte0 As Byte
Byte0 = Guid.Item(0)                    'get
Guid.Item(0) = Byte0                    'set
```

C# syntax

```
byte Byte0 = Guid[0];                   //get
Guid[0] = Byte0;                        //set
```

ToString method

Prototype

```
String ToString()
```

Comments

The `ToString` method converts and returns the bytes contained in the `FireiGUID` object to a regular string, with each of the 8 bytes separated with the ‘:’ character for better clarity. Each byte is represented in 2-character hexadecimal form.

Visual Basic syntax

```
Dim StringGUID As String
StringGUID = Guid.ToString
```

C# syntax

```
string StringGUID = Guid.ToString();
```

InitFromString method

Prototype

```
public void InitFromString(string guidString)
```

Comments

The `InitFromString` method can be used to initialize a `FireiGUID` object with an 8-byte GUID represented as a string, with each byte being represented as a 2-character hexadecimal value. The bytes can be separated by any single character (must be the same all the way) or not separated at all.

The output of the `ToString` method can be fed to the `InitFromString` method directly if desired.

Visual Basic syntax

```
Dim StringGUID As String
StringGUID = "XX:XX:XX:XX:XX:XX:XX:XX"
Guid.InitFromString StringGUID
```

C# syntax

```
string StringGUID = "XX:XX:XX:XX:XX:XX:XX:XX";
FireiGUID Guid = new FireiGUID();
Guid.InitFromString(StringGUID);
```

FireiManager

The `FireiManager` object is the starting point for the selection and construction of cameras.

It itself can be constructed directly. By constructing it directly it is ensured that, a Fire-i.net program can be completely detached from any UI, existing entirely in command line (provided no video preview is necessary, just image processing and/or manipulation).

There is no default constructor. The single constructor available requires a parameter to be passed denoting whether DirectShow or DLL mode will be utilized.

It has a number of methods implemented, all having to do with the selection and construction of `FireiCamera` objects.

GetConnectedCamerasCount method

Prototype

```
public uint GetConnectedCamerasCount()
```

Comments

This method will return the number of connected cameras on the bus.

Please keep in mind that since cameras can be freely plugged in and unplugged, this call can return different results at different times, even during runtime. Also, the number of cameras connected to the bus is not the same as the number of cameras directly connected to the PC running the Fire-i.net program. The 1394 bus networks any connected cameras and PCs together, and the call to `GetConnectedCamerasCount` will return *all* the accessible cameras from this PC.

This call is not light on system resources (it produces a lot of traffic on the 1394 bus searching for cameras) and care should be exercised when using it.

Visual Basic syntax

```
Dim NumOfCameras As UInteger
NumOfCameras = Manager.GetConnectedCamerasCount
```

C# syntax

```
uint NumOfCameras;
Manager.GetConnectedCamerasCount();
```

SelectCamera method

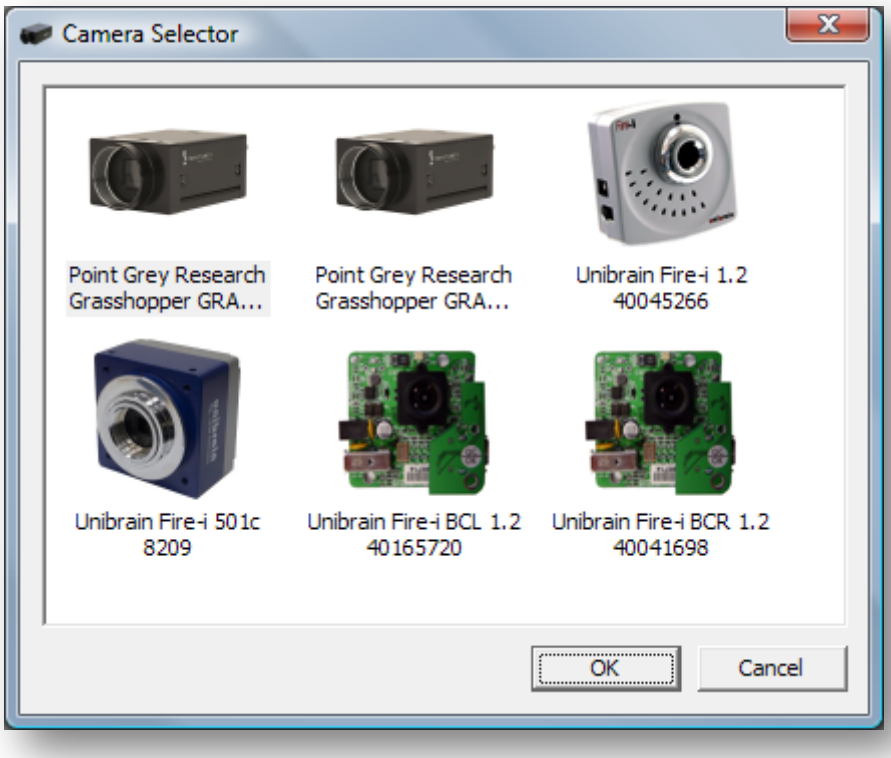
Prototype

```
public bool SelectCamera(out FireiGUID cameraGuid)
```

Comments

This method when called will bring up a “Camera Selector” dialog, as constructed and maintained internally by the APIs. Through this dialog, user selection of a camera is possible. The user can double-click on a camera to select it, or click on one and then “OK”.

The dialog presented looks similar to:



The method returns true if a selection was made by the user and false if "Cancel" was pressed (no selection made). If the result is true, then a `FireiGUID` object is constructed and returned through the `cameraGuid` parameter. This `FireiGUID` can then be used by other methods, such as the `GetCameraFromGUID` method.

If only one camera is connected to the system, the method returns true immediately, without presenting the selector dialog. The `FireiGUID` then represents the GUID of the single camera.

Visual Basic syntax

```
Dim GUID As FireiGUID
Dim Selected As Boolean
Selected = Manager.SelectCamera(GUID)
```

C# syntax

```
FireiGUID GUID;
bool Selected = Manager.SelectCamera(out GUID);
```

GetCameraFromIndex method

Prototype

```
public FireiCamera GetCameraFromIndex(uint index)
```

Comments

This method constructs a `FireiCamera` object and returns it, given an Index number. This number must be less than `GetConnectedCamerasCount`, starting from 0.

It is most useful when:

- A single camera is connected (a call with 0 index is sufficient).
- The order in which the cameras are created is not important, and all camera objects must be created (a for-loop from 0 to `GetConnectedCamerasCount – 1` is sufficient).
- A search for a specific camera, following specific criteria is required (the same for-loop, this time looking at the properties of each camera and stopping at the first that meets the criteria).

Visual Basic syntax

```
Dim Camera As FireiCamera
Dim i, NumOfCameras As Byte
NumOfCameras = Manager.GetConnectedCamerasCount
For i = 0 To NumOfCameras - 1
      Camera = Manager.GetCameraFromIndex(i)
      If Camera.Vendor = "Unibrain" Then Exit For
Next
```

C# syntax

```
for (int i = 0; i < Manager.GetConnectedCamerasCount(); ++i)
{
    Camera = Manager.GetCameraFromIndex(i);
    if (Camera.Vendor == "Unibrain")
        break;
}
```

GetCameraFromGUID method

Prototype

```
public FireiCamera GetCameraFromGUID(FireiGUID guid)
```

Comments

This method constructs a `FireiCamera` object and returns it, given a valid `FireiGUID` object. The GUID represented by this `FireiGUID` must belong to a camera connected on the system; otherwise an exception will be thrown.

It is most useful when the camera GUID is known beforehand, and provides a more direct way of accessing the camera as opposed to `GetCameraFromIndex`, because the positioning of the camera on the bus is irrelevant.

It is also mated to the `SelectCamera` method, as the `FireiGUID` returned by that method can be passed directly in `GetCameraFromGUID` to create the selected camera.

Visual Basic syntax

```
Dim Camera As FireiCamera
Dim GUID As FireiGUID
If Manager.SelectCamera(GUID) = True Then
      Camera = Manager.GetCameraFromGUID(GUID)
End If
```

C# syntax

```
FireiGUID guid;
if (Manager.SelectCamera(out guid))
      Camera = Manager.GetCameraFromGUID(guid);
```

FireiRegister

The `FireiRegister` object is a helper object designed to make reading and writing values contained in a 4-byte register easy. It can be used to read/write any single bit of the register (represented as a Boolean value), or read/write any field of the register (represented as a uint value, and defined as from one bit to another bit).

It can be used in conjunction with the `ReadRegister/WriteRegister` and `ReadCommand/WriteCommand` methods of the camera, as those methods use `FireiRegister` objects.

GetBit method

Prototype

```
public bool GetBit(byte index)
```

Comments

The `GetBit` method is used for reading a single bit of the register, and it returns a Boolean value; it is true when the bit is 1 and false when the bit is 0 in the `FireiRegister` object.

The Index parameter can carry any value from 0 to 31, with 0 being the Most Significant Bit (represented in little-endian form, the same way cameras store their register values in them). If a value greater than 31 is passed, an error will occur.

It is the equivalent of using the `GetField` method with the FromIndex and ToIndex parameters being equal to Index.

Visual Basic syntax

```
Dim Bit5 As Boolean
Bit5 = Reg.GetBit(5)
```

C# syntax

```
bool Bit5 = Reg.GetBit(5);
```

SetBit method

Prototype

```
public void SetBit(byte index, bool value)
```

Comments

The `SetBit` method is used for writing a single bit to the register, using a Boolean value; true to set the bit to 1 and false to set the bit to 0 in the `FireiRegister` object.

The Index parameter can carry any value from 0 to 31, with 0 being the Most Significant Bit (represented in little-endian form, the same way cameras store their register values in them). If a value greater than 31 is passed, an error will occur.

It is the equivalent of using the `SetField` method with the FromIndex and ToIndex parameters being equal to Index.

Visual Basic syntax

```
Reg.SetBit(5, True)
```

C# syntax

```
Reg.SetBit(5, true);
```

GetField method

Prototype

```
public uint GetField(byte fromIndex, byte toIndex)
```

Comments

The `GetField` method can be used to read a sequence of bits in the register, from a given bit up to and including another bit. It returns a uint value, so the entire register can be returned if desired.

The two Index parameters must both be from 0 to 31, and the ToIndex value must be greater than or equal to the FromIndex value. In any other case, an exception is thrown.

If FromIndex and ToIndex are equal, it is the same as calling the `GetBit` method with the same value (albeit return 0 or 1 as uint, instead of false or true as Boolean).

Visual Basic syntax

```
Dim Value As UInteger
Value = Reg.GetField(5, 7)
```

C# syntax

```
uint Value = Reg.GetField(5, 7);
```

SetField method

Prototype

```
public void SetField(byte fromIndex, byte toIndex, uint value)
```

Comments

The `SetField` method can be used to write a sequence of bits to the register, from a given bit up to and including another bit. It uses a uint value, so the entire register can be written if desired.

The two Index parameters must both be from 0 to 31, and the ToIndex value must be greater than or equal to the FromIndex value. In any other case, an exception is thrown. Also, the uint value passed must be less than or equal to the maximum value that can be stored in the number of bits comprising the field. So for instance, the examples below, which define a field of length of 3 bits can store a value from 0 to 7. If a value of 8 or greater was tried, an exception would be thrown.

If FromIndex and ToIndex are equal, it is the same as calling the `SetBit` method with the same value (albeit passing 0 or 1 as uint, instead of false or true as Boolean).

Visual Basic syntax

```
Reg.SetField(5, 7, 6)
```

C# syntax

```
Reg.SetField(5, 7, 6);
```

GetFieldLen method

Prototype

```
public uint GetFieldLen(byte fromIndex, byte fieldLength)
```

Comments

This method is similar to `GetField`, the only difference being the second parameter; here it denotes the total bit-length of the field, instead of the ending bit.

Visual Basic syntax

```
Dim Value As UInteger
Value = Reg.GetFieldLen(5, 3)
```

C# syntax

```
uint Value = Reg.GetFieldLen(5, 3);
```

SetFieldLen method

Prototype

```
public void SetFieldLen(byte fromIndex, byte fieldLength, uint value)
```

Comments

This method is similar to `SetField`, the only difference being the second parameter; here it denotes the total bit-length of the field, instead of the ending bit.

Visual Basic syntax

```
Reg.SetFieldLen(5, 3, 2)
```

C# syntax

```
uint Value = Reg.GetFieldLen(5, 3, 2);
```

SwapEndianess method

Prototype

```
public void SwapEndianess()
```

Comments

This method does a 32-bit in-place endianness swap in the `FireiRegister` object internally.

Please note that the entire value is swapped, meaning that any subsequent call to the `FireiRegister` methods will yield different results. Also, two subsequent calls to `SwapEndianess` have no effect on the value.

Visual Basic syntax

```
Reg.SwapEndianess()
```

C# syntax

```
Reg.SwapEndianess();
```

FireiTrigger

The `FireiTrigger` object encapsulates all the Trigger functionality that the camera may support. An instance cannot be constructed directly; it is rather accessed through the `Trigger` property of the camera. When no longer in use however, it should be disposed of properly (through its Dispose() method.)

Please note that the following properties have no additional functionality hidden inside them, besides what the camera itself supports. In essence, the entire `FireiTrigger` object could have been two simple `FireiRegister` objects (one for inquiry, one for control); it exits merely as a helping hand, in effect "naming" the bits and fields of the two registers as they map to the Trigger inquiry and control registers of the camera.

In order to call any of the following properties and methods, the `IsSupported` property must be true, otherwise each call will throw an exception. Additionally, in order to read the value of any of the following properties, the `CanRead` property must also be true.

AbsControl property

Prototype

```
public bool AbsControl
```

Comments

This property reads/sets the `Abs_Control` field of the camera trigger control register (bit 1). In order to access this property, the property `HasAbsolute` of the `FireiTrigger` object must be true, otherwise an error will occur.

Visual Basic syntax

```
Dim Abs As Boolean
Abs = Trigger.AbsControl                    'get
Trigger.AbsControl = True                   'set
```

C# syntax

```
bool Abs = Trigger.AbsControl;              //get
Trigger.AbsControl = true;                  //set
```

Enabled property

Prototype

```
public bool Enabled
```

Comments

This property reads/sets the `ON_OFF` field of the camera trigger control register (bit 6). In order to access this property, the property `HasOnOff` of the `FireiTrigger` object must be true, otherwise an exception will be thrown.

Visual Basic syntax

```
Dim On As Boolean
On = Trigger.Enabled                                    'get
Trigger. Enabled = True                                 'set
```

C# syntax

```
bool On;
On = Trigger.Enabled;                                   //get
Trigger.Enabled = true;                                 //set
```

Polarity property

Prototype

```
public bool Polarity
```

Comments

This property reads/sets the `Trigger_Polarity` field of the camera trigger control register (bit 7). In order to access this property, the property `HasPolarity` of the `FireiTrigger` object must be true, otherwise an exception will be thrown.

Visual Basic syntax

```
Dim Polarity As Boolean
Polarity = Trigger.Polarity                             'get
Trigger.Polarity = True                                 'set
```

C# syntax

```
bool Polarity;
On = Trigger. Polarity;                                 //get
Trigger. Polarity = true;                               //set
```

Source property

Prototype

```
FireiTriggerSource Source
```

Comments

This property reads/sets the `Trigger_Source` field of the camera trigger control register (bits 8 to 10). In order to set this property, a call to method `IsSourceSupported` of the `FireiTrigger` object should first be performed, to check if the value being set is supported in this camera.

`FireiTriggerSource` is an enumerated value, with possible values being `None, Source_0, Source_1, Source_2, Source_3` and `Source_SW`.

Visual Basic syntax

```
Dim TriggerSource As FireiTriggerSource
TriggerSource = Trigger.Source                          'get
Trigger.Source = FireiTriggerSource.Source_0            'set
```

C# syntax

```
FireiTriggerSource Source = Trigger.Source;             //get
Trigger.Source = FireiTriggerSource.Source_0;           //set
```

Value property

Prototype

```
public bool Value
```

Comments

This property reads/sets the `Trigger_Value` field of the camera trigger control register (bit 11). In order to read this property, the property `CanReadRaw` of the `FireiTrigger` object must be true, otherwise an exception will be thrown.

The resulting value can be considered as false being low and true being high signal value.

Visual Basic syntax

```
Dim Value As Boolean
Value = Trigger.Value                                   'get
Trigger.Value = True                                    'set
```

C# syntax

```
bool Value = Trigger.Value;                             //get
Trigger.Value = true;                                   //set
```

Mode property

Prototype

```
public FireiTriggerMode Mode
```

Comments

This property reads/sets the `Trigger_Mode` field of the camera trigger control register (bits 12 to 15). In order to set this property, a call to method `IsModeSupported` of the `FireiTrigger` object should first be performed, to check if the value being set is supported in this camera.

`FireiTriggerMode` is an enumerated value, with possible values being `None`, `Mode_0`, `Mode_1`, `Mode_2`, `Mode_3`, `Mode_4`, `Mode_5`, `Mode14` and `Mode15`.

Visual Basic syntax

```
Dim TriggerMode As FireiTriggerMode
TriggerMode = Trigger.Mode                              'get
Trigger.Mode = FireiTriggerMode.Mode_0                  'set
```

C# syntax

```
FireiTriggerMode Mode = Trigger.Mode;                   //get
Trigger.Mode = FireiTriggerMode.Mode_0;                 //set
```

IsSupported property

Prototype

```
public bool IsSupported
```

Comments

This is a read only property that reflects the `Presence_Inq` field of the camera trigger inquiry register (bit 0). This property can always be read, and it is a necessary requirement for it to be true for all the other methods and properties to have their intended behavior.

Visual Basic syntax

```
Dim Supported As Boolean
Supported = Trigger.IsSupported
```

C# syntax

```
bool Supported = Trigger.IsSupported;
```

HasAbsolute property

Prototype

```
public bool HasAbsolute
```

Comments

This is a read only property that reflects the `Abs_Control_Inq` field of the camera trigger inquiry register (bit 1). It is a necessary requirement for it to be true for the `AbsControl` property to be accessible.

Visual Basic syntax

```
Dim HasAbs As Boolean
HasAbs = Trigger.HasAbsolute
```

C# syntax

```
bool HasAbs = Trigger.HasAbsolute;
```

CanRead property

Prototype

```
public bool CanRead
```

Comments

This is a read only property that reflects the `ReadOut_Inq` field of the camera trigger inquiry register (bit 4). It is a necessary requirement for it to be true for any of the control properties to be readable.

Visual Basic syntax

```
Dim CanRead As Boolean
CanRead = Trigger.CanRead
```

C# syntax

```
bool CanRead = Trigger.CanRead;
```

HasOnOff property

Prototype

```
public bool HasOnOff
```

Comments

This is a read only property that reflects the `On/Off_Inq` field of the camera trigger inquiry register (bit 5). It is a necessary requirement for it to be true for the `Enable` property to be accessible.

Visual Basic syntax

```
Dim OnOff As Boolean
OnOff = Trigger.HasOnOff
```

C# syntax

```
bool OnOff = Trigger.HasOnOff;
```

HasPolarity property

Prototype

```
Boolean HasPolarity
```

Comments

This is a read only property that reflects the `Polarity_Inq` field of the camera trigger inquiry register (bit 6). It is a necessary requirement for it to be true for the `Polarity` property to be accessible.

Visual Basic syntax

```
Dim HasPol As Boolean
HasPol = Trigger.HasPolarity
```

C# syntax

```
Bool HasPol = Trigger.HasPolarity;
```

CanReadRaw property

Prototype

```
Boolean CanReadRaw
```

Comments

This is a read only property that reflects the `Value_Read_Inq` field of the camera trigger inquiry register (bit 7). It is a necessary requirement for it to be true for the `Value` property to be accessible.

Visual Basic syntax

```
Dim CanReadRaw As Boolean
CanReadRaw = Trigger.CanReadRaw
```

C# syntax

```
CanReadRaw = Trigger.CanReadRaw;
```

Parameter property

Prototype

```
ushort Parameter
```

Comments

This property reads/sets the `Parameter` field of the camera trigger control register (bits 20 to 31). Since this is an optional field for the trigger, there is no equivalent inquiry to be made whether it is available or not. However, acceptable values are integers greater than or equal to zero and less than 4096.

Visual Basic syntax

```
Dim Parameter As UShort
Parameter = Trigger.Parameter                    'get
Trigger.Parameter = 1000                         'set
```

C# syntax

```
ushort Parameter;
Parameter = Trigger.Parameter;                   //get
Trigger.Parameter = 1000;                        //set
```

IsSourceSupported method

Prototype

```
Boolean IsSourceSupported(FireiTriggerSource source)
```

Comments

This method checks whether a given `FireiTriggerSource` value is supported by the camera. The programmer should typically test for the support of a trigger source value, before trying to set it using the `Source` property.

Visual Basic syntax

```
Dim Supported As Boolean
Supported = Trigger.IsSourceSupported(FireiTriggerSource.Source_0)
```

C# syntax

```
bool Supported = Trigger.IsSourceSupported(FireiTriggerSource.Source_0);
```

IsModeSupported method

Prototype

```
Boolean IsModeSupported(FireiTriggerMode Mode)
```

Comments

This method checks whether a given `FireiTriggerMode` value is supported by the camera. The programmer should typically test for the support of a trigger mode value, before trying to set it using the `Mode` property.

Visual Basic syntax

```
Dim Supported As Boolean
Supported = Trigger.IsModeSupported(FireiTriggerMode.Mode_0)
```

C# syntax

```
bool Supported = Trigger.IsModeSupported(FireiTriggerMode.Mode_0);
```

Reload method

Prototype

```
Reload()
```

Comments

This method reloads the values of the trigger control register from the camera, so effectively it acts as an "undo" function for any changes made through the various properties, as long as `Save` hasn't yet been called.

The inquiry register on the other hand is not reloaded, since it contains read-only values which cannot be changed anyway.

Visual Basic syntax

```
Trigger.Reload()
```

C# syntax

```
Trigger.Reload();
```

PullSoftwareTrigger method

Prorotype

```
PullSoftwareTrigger()
```

Comments

This method sets the software_trigger register of the camera for acquiring a frame. If the source property of the trigger is not set to Source_SW, an exception will be thrown.

You can call PullSoftwareTrigger method subsequently for acquiring new frames.

Visual Basic syntax

```
Trigger.PullSoftwareTrigger()
```

C# syntax

```
Trigger.PullSoftwareTrigger();
```

Save method

Prototype

```
Save()
```

Comments

This method saves any changes made to the trigger control register of the camera. If for some reason this cannot be completed (i.e., the camera refuses the write request), an exception will be thrown.

Any subsequent calls to the `Reload` method will revert to this saved state.

Visual Basic syntax

```
Trigger.Save()
```

C# syntax

```
Trigger.Save();
```

FireiStreamFormat

The `FireiStreamFormat` object encapsulates the video format settings relevant to a given camera. A `FireiStreamFormat` object instance cannot and should not be created directly; its functionality is paired closely with the specific camera, so the camera is responsible for constructing the object.

To obtain a `FireiStreamFormat`, the programmer can use the `StreamFormat` property of `FireiCamera`. Additionally, a list of all available to the camera `FireiStreamFormats` can be obtained through the `GetStreamFormats` method; all the `FireiStreamFormats` contained in it are connected with that camera automatically. It can also be obtained directly through the two "retrieve" methods, `RetrieveStreamFormat` and `RetrieveStreamFormatFromIdentifier`.

The following properties and methods are not final; changes are kept in the `FireiStreamFormat` object and only passed to the camera as the current settings when the `Save` method is called (or the `FireiStreamFormat` is set to the camera using the `StreamFormat` property).

PixelFormatString property

Prototype

```
String PixelFormatString
```

Comments

This is a read only property that returns the pixel format of the `FireiStreamFormat` as a string. A pixel format is not unique to a `FireiStreamFormat`. A pair of a pixel format and a resolution however is.

Visual Basic syntax

```
Dim PixelFormat As String
PixelFormat = StreamFormat.PixelFormatString
```

C# syntax

```
string PixelFormat = StreamFormat.PixelFormatString;
```

PixelFormat property

Prototype

```
FireiPixelFormat PixelFormat
```

Comments

This is a read only property that returns the pixel format of the `FireiStreamFormat` as a `FireiPixelFormat` value. A pixel format is not unique to a `FireiStreamFormat`. A pair of a pixel format and a resolution however is.

Possible values in the `FireiPixelFormat` enumerated value are:

```
None,
```

```
Y_MONO,
YUV_411,
YUV_422,
YUV_444,
RGB_24,
Y_MONO_16,
RGB_48,
S_Y_MONO_16,
S_RGB_48,
RAW_8,
RAW_16
```

Visual Basic syntax

```
Dim PixelFormat As FireiPixelFormat
PixelFormat = StreamFormat.PixelFormat
```

C# syntax

```
FireiPixelFormat PixelFormat = StreamFormat.PixelFormat;
```

Resolution property

Prototype

```
FireiResolution Resolution
```

Comments

This is a read only property that returns the resolution of the `FireiStreamFormat` as a `FireiResolution` value. A resolution is not unique to a `FireiStreamFormat`. A pair of a pixel format and a resolution however is.

Possible values in the `FireiResolution` enumerated value are:

```
None,
res160x120,
res320x240,
res640x480,
res800x600,
res1024x768,
res1280x960,
res1600x1200,
Variable,
Variable_1,
Variable_2,
Variable_3,
Variable_4,
Variable_5,
Variable_6,
Variable_7
```

The values from `res160x120` up to `res1600x1200` represent Fixed formats, and from `Variable` to `Variable_7` represent User Defined formats.

Visual Basic syntax

```
Dim Resolution As FireiResolution
Resolution = StreamFormat.Resolution
```

C# syntax

```
FireiResolution Resolution = StreamFormat.Resolution;
```

IsUserDefined property

Prototype

```
Boolean IsUserDefined
```

Comments

This is a read only property that returns whether the given `FireiStreamFormat` is User Defined. If true, the `UserDefined` property of `FireiStreamFormat` can be used. Otherwise, the `Fixed` property can be used.

Visual Basic syntax

```
Dim UserDefined As Boolean
UserDefined = StreamFormat.IsUserDefined
```

C# syntax

```
bool IsUserDefined = StreamFormat.IsUserDefined;
```

IsCurrent property

Prototype

```
Boolean IsCurrent
```

Comments

This is a read only property that returns whether the given `FireiStreamFormat` is the currently selected streaming format of the camera. Please note that it does not necessarily mean that all parameters of the stream format are saved on the camera; it merely means it has the same pixel format and resolution.

If the `FireiStreamFormat` was obtained through the `StreamFormat` property of `FireiCamera` (which returns the currently selected `FireiStreamFormat`) this property should always be true. It is therefore only useful if the `FireiStreamFormat` object was obtained through other means.

Visual Basic syntax

```
Dim Current As Boolean
Current = StreamFormat.IsCurrent
```

C# syntax

```
bool IsCurrent = StreamFormat.IsCurrent;
```

Identifier property

Prototype

```
uint Identifier
```

Comments

This is a read only property that returns a unique identifier, in the form of a uint value that can be stored for later usage (e.g., to retrieve this format directly from the camera using `RetrieveStreamFormatFromIdentifier`).

This identifier is ideal for storing the stream format selection to a windows control, such as a Combo Box or a List Box (as each entry's data). It should be noted however, that the variable settings of `FireiStreamFormat` are not transferred through this unique identifier; only the constant attributes of a `FireiStreamFormat`, i.e., the pixel format and resolution.

Visual Basic syntax

```
Dim ID As UInteger
ID = StreamFormat.Identifier
```

C# syntax

```
uint ID = StreamFormat.Identifier;
```

Fixed property

Prototype

```
FireiFixedStreamFormat Fixed
```

Comments

This is a read only property that returns the associated `FireiFixedStreamFormat` object with this `FireiStreamFormat`. It is only valid if the `IsUserDefined` property is false (a null value will be returned if the `IsUserDefined` property is true).

Visual Basic syntax

```
Dim Fixed As FireiFixedStreamFormat
Set Fixed = StreamFormat.Fixed
```

C# syntax

```
FireiFixedStreamFormat Fixed = StreamFormat.Fixed;
```

UserDefined property

Prototype

```
FireiUserDefinedStreamFormat UserDefined
```

Comments

This is a read only property that returns the associated `FireiUserDefinedStreamFormat` object with this `FireiStreamFormat`. It is only valid if the `IsUserDefined` property is true (a null value will be returned if the `IsUserDefined` property is false).

Visual Basic syntax

```
Dim UserDefined As FireiUserDefinedStreamFormat
Set UserDefined = StreamFormat.UserDefined
```

C# syntax

```
FireiUserDefinedStreamFormat UserDefined = StreamFormat.UserDefined;
```

RawModeOverride property

Prototype

```
FireiRawMode RawModeOverride
```

Comments

This is a read/write property that allows the programmer to establish if a bayer conversion will be performed on the image data coming from the camera and if so, the color filter that the camera is using.

The default setting is `Auto`, which means the SDK will try to determine the correct conversion necessary. This is not always possible, since many raw-mode cameras sent raw-data as regular Y_MONO. In those cases, an override is necessary if a conversion to RGB is desired, hence this property.

Per the IIDC specification, only under User Defined modes (Format_7) a camera is allowed to send raw unconverted data – in that case, the color filter is supplied by the camera. However, many camera manufacturers disguise raw data as Y_MONO, sent either through Fixed or User Defined streaming formats.

The `RawModeOverride` property carries enumerated values; possibilities are:

```
Auto,
None,
RGGB,
GRBG,
GBRG,
BGGR
```

Visual Basic syntax

```
Dim RawMode As FireiRawMode
RawMode = StreamFormat.RawModeOverride                    'get
StreamFormat.RawModeOverride = FireiRawMode.RGGB          'set
```

C# syntax

```
FireiRawMode RawMode = StreamFormat.RawModeOverride;      //get
StreamFormat.RawModeOverride = FireiRawMode.RGGB;         //set
```

Width property

Prototype

```
ushort Width
```

Comments

This is a read only property that returns the resolution width of the streaming format. This is constant for a Fixed streaming format and variable on a User Defined.

Visual Basic syntax

```
Dim Width As UShort
Width = StreamFormat.Width
```

C# syntax

```
ushort Width = StreamFormat.Width;
```

Height property

Prototype

```
ushort Height
```

Comments

This is a read only property that returns the resolution height of the streaming format. This is constant for a Fixed streaming format and variable on a User Defined.

Visual Basic syntax

```
Dim Height As UShort
Height = StreamFormat.Height
```

C# syntax

```
ushort Height = StreamFormat.Height;
```

ToString method

Prototype

```
String ToString()
```

Comments

This method returns a textual representation of the `FireiStreamFormat`. The text contains the pixel format and the resolution of the streaming format. A typical result for a Fixed format would look like:

```
Y_MONO, 160 X 120
```

In case of a User Defined format the string contains the maximum resolution instead, similar to:

```
Y_MONO, Max 160 X 120
```

Visual Basic syntax

```
Dim Description As String
Description = StreamFormat.ToString()
```

C# syntax

```
string Description = StreamFormat.ToString();
```

Save method

Prototype

```
Save()
```

Comments

This method sets this `FireiStreamFormat` to the camera, including any selected attributes, whether Fixed or User Defined. If for some reason this cannot be completed (i.e., the camera refuses the write request), an exception will be thrown.

The effect of this method is the same as using the `StreamFormat` property of `FireiCamera`, to set this `FireiStreamFormat`.

Visual Basic syntax

```
StreamFormat.Save()
```

C# syntax

```
StreamFormat.Save();
```

FireiFixedStreamFormat

The `FireiFixedStreamFormat` object is derived from `FireiStreamFormat`. It carries any additional functionality that pertains to Fixed streaming formats. It cannot be constructed directly; it is only ever obtained through the `Fixed` property of `FireiStreamFormat`.

Its properties are mostly read-only, except for the `FrameRate`, which can be set by the programmer.

ResolutionString property

Prototype

```
String ResolutionString
```

Comments

This is a read only property that returns the resolution of the `FireiFixedStreamFormat` as a string. A resolution is not unique to a `FireiStreamFormat`. A pair of a pixel format and a resolution however is.

Visual Basic syntax

```
Dim Resolution As String
Resolution = Fixed.ResolutionString
```

C# syntax

```
BSTR Resolution;
HRESULT hr = pIFixed->get_ResolutionString(&Resolution);
```

FrameRateString property

Prototype

```
String FrameRateString
```

Comments

This is a read/write property that can be used to retrieve a textual representation of the frame rate of the streaming format and then set it back.

Since the format of the text is specific, setting the property can be safely done only with values got from it.

Visual Basic syntax

```
Dim FrameRate As String
FrameRate = Fixed.FrameRateString                    'get
Fixed.FrameRateString = FrameRate                    'set
```

C# syntax

```
string FrameRate = Fixed.FrameRateString;            //get
Fixed.FrameRateString = FrameRate;                   //set
```

FrameRate property

Prototype

```
FireiFrameRate FrameRate
```

Comments

This is a read/write property that can be used to retrieve or set the frame rate of the `FireiFixedStreamFormat`. It is represented by an enumerated value (`FireiFrameRate`), with the following possibilities:

```
None,
fps1_875,
fps3_75,
fps7_5,
fps15,
fps30,
fps60,
fps120,
fps240
```

The programmer can test whether a given frame rate is valid with this Fixed format and acceptable by the camera, a call to `IsFrameRateSupported` is sufficient.

Visual Basic syntax

```
Dim FrameRate As FireiFrameRate
FrameRate = Fixed.FrameRate                    'get
Fixed.FrameRate = FireiFrameRate.fps30         'set
```

C# syntax

```
FireiFrameRate FrameRate = Fixed.FrameRate;    //get
Fixed.FrameRate = FireiFrameRate.fps30;        //set
```

PacketSize property

Prototype

```
uint PacketSize
```

Comments

This is a read only property that returns the size in bytes of the packets for this streaming format. This will be constant between successive reads, since the packet size is always the same on a given Fixed streaming format.

Visual Basic syntax

```
Dim PacketSize As UInteger
PacketSize = Fixed.PacketSize
```

C# syntax

```
uint PacketSize = Fixed.PacketSize;
```

PacketsPerFrame property

Prototype

```
uint PacketsPerFrame
```

Comments

This is a read only property that returns the number of packets per frame for this streaming format. This will be constant between successive reads, since the number of packets per frame is always the same on a given Fixed streaming format.

Visual Basic syntax

```
Dim PacketsPerFrame As UInteger
PacketsPerFrame = Fixed.PacketsPerFrame
```

C# syntax

```
uint PacketsPerFrame = Fixed.PacketsPerFrame;
```

ToString method

Prototype

```
String ToString()
```

Comments

This method returns a textual representation of the `FireiFixedStreamFormat`. The text contains the pixel format and the resolution of the streaming format. A typical result would look like:

```
Y_MONO, 160 X 120
```

It is also the same value returned by the `ToString` method of the `FireiStreamFormat` from which this `FireiFixedStreamFormat` was derived from.

Visual Basic syntax

```
Dim Description As String
Description = Fixed.ToString()
```

C# syntax

```
string Description = Fixed.ToString();
```

IsFrameRateSupported method

Prototype

```
Boolean IsFrameRateSupported(FireiFrameRate frameRate)
```

Comments

This method will return true if the supplied `FireiFrameRate` is both available in this `FireiFixedStreamFormat` and acceptable by the camera and the current conditions of the 1394 bus.

A frame rate may not be available on a given Fixed streaming format, as defined in the IIDC spec. It may also not be supported by the camera; these are static conditions. There are dynamic conditions though, such as the isochronous speed of the bus the camera is connected to, limited by both hardware and software. Some frame rates may require a higher isochronous speed than what is available currently, and even though the camera may support the frame rate, it might not be currently achievable. In all those cases, `IsFrameRateSupported` will return false.

Visual Basic syntax

```
Dim Supported As Boolean
Supported = Fixed.IsFrameRateSupported(FireiFrameRate.fps30)
```

C# syntax

```
bool Supported = Fixed.IsFrameRateSupported(FireiFrameRate.fps30);
```

FireiUserDefinedStreamFormat

The `FireiUserDefinedStreamFormat` object is derived from `FireiStreamFormat`. It carries any additional functionality that pertains to User Defined streaming formats. It cannot be constructed directly; it is only ever obtained through the `UserDefined` property of `FireiStreamFormat`.

Unlike the `FireiFixedStreamFormat` object, there many selectable attributes in `FireiUserDefinedStreamFormat`.

Left property

Prototype

```
ushort Left
```

Comments

This is a read only property that can be used to read the currently set left coordinate of the Region of Interest set in the `FireiUserDefinedStreamFormat`. This can be set through the `SetROI` method.

Visual Basic syntax

```
Dim Left As UShort
Left = UserDefined.Left
```

C# syntax

```
ushort Left = UserDefined.Left;
```

Right property

Prototype

```
ushort Right
```

Comments

This is a read only property that can be used to read the currently set right coordinate of the Region of Interest set in the `FireiUserDefinedStreamFormat`. This can be set through the `SetROI` method.

Visual Basic syntax

```
Dim Right As UShort
Right = UserDefined.Right
```

C# syntax

```
ushort Right = UserDefined.Right;
```

Top property

Prototype

```
ushort Top
```

Comments

This is a read only property that can be used to read the currently set top coordinate of the Region of Interest set in the `FireiUserDefinedStreamFormat`. This can be set through the `SetROI` method.

Visual Basic syntax

```
Dim Top As UShort
Top = UserDefined.Top
```

C# syntax

```
ushort Top = UserDefined.Top;
```

Bottom property

Prototype

```
ushort Bottom
```

Comments

This is a read only property that can be used to read the currently set bottom coordinate of the Region of Interest set in the `FireiUserDefinedStreamFormat`. This can be set through the `SetROI` method.

Visual Basic syntax

```
Dim Bottom As UShort
Bottom = UserDefined.Bottom
```

C# syntax

```
ushort Bottom = UserDefined.Bottom;
```

Width property

Prototype

```
ushort Width
```

Comments

This is a read only property that can be used to read the currently set width of the Region of Interest set in the `FireiUserDefinedStreamFormat`. This can be set through the `SetROI` method and it is equal to `Right – Left`. It is also the same value returned by the `Width` property of the `FireiStreamFormat` from which this `FireiUserDefinedStreamFormat` was derived from.

Visual Basic syntax

```
Dim Width As UShort
Width = UserDefined.Width
```

C# syntax

```
ushort Width = UserDefined.Width;
```

Height property

Prototype

```
ushort Height
```

Comments

This is a read only property that can be used to read the currently set height of the Region of Interest set in the `FireiUserDefinedStreamFormat`. This can be set through the `SetROI` method and it is equal to `Bottom – Top`. It is also the same value returned by the `Height` property of the `FireiStreamFormat` from which this `FireiUserDefinedStreamFormat` was derived from.

Visual Basic syntax

```
Dim Height As UShort
Height = UserDefined.Height
```

C# syntax

```
ushort Height = UserDefined.Height;
```

MaxWidth property

Prototype

```
ushort MaxWidth
```

Comments

This is a read only property that can be used to read the maximum possible width for the Region of Interest of the `FireiUserDefinedStreamFormat`.

Visual Basic syntax

```
Dim MaxWidth As UShort
MaxWidth = UserDefined.MaxWidth
```

C# syntax

```
ushort MaxWidth = UserDefined.MaxWidth;
```

MaxHeight property

Prototype

```
ushort MaxHeight
```

Comments

This is a read only property that can be used to read the maximum possible height for the Region of Interest of the `FireiUserDefinedStreamFormat`.

Visual Basic syntax

```
Dim MaxHeight As UShort
MaxHeight = UserDefined.MaxHeight
```

C# syntax

```
ushort MaxHeight = UserDefined.MaxHeight;
```

HorizontalPositionUnit property

Prototype

```
ushort HorizontalPositionUnit
```

Comments

This is a read only property that can be used to read the horizontal unit for the position of the Region of Interest of the `FireiUserDefinedStreamFormat`. The left, right, top and bottom x-coordinates of the ROI must be perfectly divisible by this value.

Visual Basic syntax

```
Dim HPosUnit As UShort
```

```
HPosUnit = UserDefined.HorizontalPositionUnit
```

C# syntax

```
ushort HPosUnit = UserDefined.HorizontalPositionUnit;
```

VerticalPositionUnit property

Prototype

```
ushort VerticalPositionUnit
```

Comments

This is a read only property that can be used to read the vertical unit for the position of the Region of Interest of the `FireiUserDefinedStreamFormat`. The left, right, top and bottom y-coordinates of the ROI must be perfectly divisible by this value.

Visual Basic syntax

```
Dim VPosUnit As UShort
VPosUnit = UserDefined.VerticalPositionUnit
```

C# syntax

```
ushort VPosUnit = UserDefined.VerticalPositionUnit;
```

WidthUnit property

Prototype

```
ushort WidthUnit
```

Comments

This is a read only property that can be used to read the width unit for the Region of Interest of the `FireiUserDefinedStreamFormat`. The total width of the ROI must be perfectly divisible by this value.

Visual Basic syntax

```
Dim WUnit As UShort
WUnit = UserDefined.WidthUnit
```

C# syntax

```
ushort WUnit = UserDefined.WidthUnit;
```

HeightUnit property

Prototype

```
ushort HeightUnit
```

Comments

This is a read only property that can be used to read the height unit for the Region of Interest of the `FireiUserDefinedStreamFormat`. The total height of the ROI must be perfectly divisible by this value.

Visual Basic syntax

```
Dim HUnit As UShort
HUnit = UserDefined.HeightUnit
```

C# syntax

```
ushort HUnit = UserDefined.HeightUnit;
```

MaxPacketSize property

Prototype

```
ushort MaxPacketSize
```

Comments

This is a read only property that can be used to read the maximum allowed packet size in bytes of the `FireiUserDefinedStreamFormat`. The `PacketSize` property cannot be set to a value greater than this value.

Visual Basic syntax

```
Dim MaxPacketSize As UShort
MaxPacketSize = UserDefined.MaxPacketSize
```

C# syntax

```
ushort MaxPacketSize = UserDefined.MaxPacketSize;
```

PacketSizeUnit property

Prototype

```
ushort PacketSizeUnit
```

Comments

This is a read only property that can be used to read the packet size unit of the `FireiUserDefinedStreamFormat`. The `PacketSize` property must be set to a value perfectly divisibly by this value.

Visual Basic syntax

```
Dim PacketSizeUnit As UShort
PacketSizeUnit = UserDefined.PacketSizeUnit
```

C# syntax

```
ushort PacketSizeUnit = UserDefined.PacketSizeUnit;
```

PacketSize property

Prototype

```
ushort PacketSize
```

Comments

This is a read/write property that can be used to retrieve or set the size in bytes of each packet the camera will transmit for this format.

When setting it, acceptable values are greater than or equal to 0 and less than or equal to the `MaxPacketSize` property. The value must also be perfectly divisible by the value of `PacketSizeUnit` property.

Setting this value to 0 means the SDK will try and determine the correct packet size automatically, either by setting it to the maximum possible, or by querying the camera for it.

Visual Basic syntax

```
Dim PacketSize As UShort
PacketSize = UserDefined.PacketSize                    'get
UserDefined.PacketSize = 0                             'set
```

C# syntax

```
ushort PacketSize = UserDefined.PacketSize;            //get
UserDefined.PacketSize = 0;                            //set
```

F7Offset property

Prototype

```
uint F7Offset
```

Comments

This is a read only property that can be used to read the register offset of this `FireiUserDefinedStreamFormat`. This value can then be used in the `ReadCommand` and `WriteCommand` methods of `FireiCamera` in order to access the streaming format directly through its register.

Visual Basic syntax

```
Dim F7Offset As UInteger
F7Offset = UserDefined.F7Offset
```

C# syntax

```
uint F7Offset = UserDefined.F7Offset;
```

ToString method

Prototype

```
String ToString()
```

Comments

This method returns a textual representation of the `FireiUserDefinedStreamFormat`. The text contains the pixel format and the maximum resolution of the streaming format. A typical result would look like:

```
Y_MONO, Max 160 X 120
```

It is also the same value returned by the `ToString` method of the `FireiStreamFormat` from which this `FireiUserDefinedStreamFormat` was derived from.

Visual Basic syntax

```
Dim Description As String
Description = UserDefined.ToString()
```

C# syntax

```
string Description = UserDefined.ToString();
```

SetROI method

Prototype

```
SetROI(ushort left, ushort top, ushort width, ushort height)
```

Comments

This method is used to set the desired coordinates and size of the Region of Interest for the `FireiUserDefinedStreamFormat`.

The left and top values must be greater than or equal to 0 and less than or equal to the `MaxWidth` and `MaxHeight` properties respectively. They must also be perfectly divisible by `HorizontalPositionUnit` and `VerticalPositionUnit` respectively.

The width and height values must be greater than or equal to `WidthUnit` and `HeightUnit` respectively and less than or equal to `MaxWidth` and `MaxHeight` respectively. They must also be perfectly divisible by `WidthUnit` and `HeightUnit` respectively. Additionally, they must not be greater than the available rectangle as set by the left and top values (i.e., the sum of width and left must not be greater than `MaxWidth` and the sum of height and top must not be greater than `MaxHeight`).

In case the above conditions are not met an exception will be thrown, containing an informative message. The method will not query the camera for validity; any checks are made inside the SDK, according to the reported parameters of the camera.

Visual Basic syntax

```
UserDefined.SetROI(0, 0, 160, 120)
```

C# syntax

```
UserDefined.SetROI(0, 0, 160, 120);
```

FireiFeature

The `FireiFeature` object encapsulates all functionality related to a camera feature. It contains all the information available through the feature's inquiry register, and all the capabilities available through the feature's control register in one concise package.

The `FireiFeature` object cannot be constructed directly by the programmer; instead, it is accessible through the camera properties at any time. The `FireiCamera` object offers a specialized read-only property for accessing each individual feature. For example, it has a `Shutter` property that returns the `FireiFeature` object pertaining to the shutter feature. If a more generic way to access features is required, the `Feature` property can be used instead; it takes a string of the name of the feature as a parameter.

Additionally, a full list of all features can be retrieved from the camera, through its `GetFeatures` method.

The `FireiFeature` object implements various properties and methods to its interface. Unlike the `FireiStreamFormat` and `FireiTrigger` objects, any change made to the `FireiFeature` instance is directly passed to the camera, as there is no specific `Save` method.

Name property

Prototype

```
String Name
```

Comments

This is a read-only property that returns the name of the object as a string. This name is the exact same name that can be used to retrieve this `FireiFeature` object, through the `GetFeature` method of `FireiCamera`.

Visual Basic syntax

```
Dim Name As String
Name = Feature.Name
```

C# syntax

```
string Name = Feature.Name;
```

IsSupported property

Prototype

```
Boolean IsSupported
```

Comments

This is a read-only property that will return whether this `FireiFeature` represents a feature supported by the camera.

If this property is false, no other property or method of `FireiFeature` should be called, otherwise an exception will be thrown.

Visual Basic syntax

```
Dim Supported As Boolean
```

```
Supported = Feature.IsSupported
```

C# syntax

```
bool Supported = Feature.IsSupported;
```

HasAbsolute property

Prototype

```
Boolean HasAbsolute
```

Comments

This is a read-only property that will return whether this `FireiFeature` supports setting its value through "Absolute" values.

If this property is false, the `Absolute` property of `FireiFeature` should not be accessed, otherwise an exception will be thrown.

Visual Basic syntax

```
Dim HasAbsolute As Boolean
HasAbsolute = Feature.HasAbsolute
```

C# syntax

```
bool HasAbsolute = Feature.HasAbsolute;
```

HasOnePush property

Prototype

```
Boolean HasOnePush
```

Comments

This is a read-only property that will return whether this `FireiFeature` supports setting its value through a "One Push" operation (as defined by the IIDC specification).

If this property is false, the `OnePush` method of `FireiFeature` should not be called, otherwise an exception will be thrown.

Visual Basic syntax

```
Dim HasOnePush As Boolean
HasOnePush = Feature.HasOnePush
```

C# syntax

```
bool HasOnePush = Feature.HasOnePush;
```

CanRead property

Prototype

```
Boolean CanRead
```

Comments

This is a read-only property that will return whether this `FireiFeature` supports reading its value.

If this property is false, the `Value` property of `FireiFeature` should not be read, otherwise an exception would be thrown. This does not mean it cannot be set however; the `HasManual` property can be used to determine that.

Visual Basic syntax

```
Dim CanRead As Boolean
CanRead = Feature.CanRead
```

C# syntax

```
bool CanRead = Feature.CanRead;
```

HasOnOff property

Prototype

```
Boolean HasOnOff
```

Comments

This is a read-only property that will return whether this `FireiFeature` supports turning it on and off entirely.

If this property is false, the `Enable` property of `FireiFeature` should not be accessed, otherwise an exception will be thrown.

Visual Basic syntax

```
Dim HasOnOff As Boolean
HasOnOff = Feature.HasOnOff
```

C# syntax

```
bool HasOnOff = Feature.HasOnOff;
```

HasAuto property

Prototype

```
Boolean HasAuto
```

Comments

This is a read-only property that will return whether this `FireiFeature` supports setting its value automatically, according to the camera shooting circumstances.

If this property is false, the `AutoMode` property of `FireiFeature` should not be accessed, otherwise an exception will be thrown.

Visual Basic syntax

```
Dim HasAuto As Boolean
HasAuto = Feature.HasAuto
```

C# syntax

```
bool HasAuto = Feature.HasAuto;
```

HasManual property

Prototype

```
Boolean HasManual
```

Comments

This is a read-only property that will return whether this `FireiFeature` supports setting its value manually, through the `Value` property.

If this property is false the `Value` property of `FireiFeature` should not be set, otherwise an exception would be thrown. It could still be read though; the `CanRead` property can be used to determine that.

Visual Basic syntax

```
Dim HasManual As Boolean
HasManual = Feature.HasManual
```

C# syntax

```
bool HasManual = Feature.HasManual;
```

MinValue property

Prototype

```
uint MinValue
```

Comments

This is a read-only property that will return the minimum value that can be set to this `FireiFeature`, through the `Value` property.

Visual Basic syntax

```
Dim MinValue As UInteger
MinValue = Feature.MinValue
```

C# syntax

```
uint MinValue = Feature.MinValue;
```

MaxValue property

Prototype

```
uint MaxValue
```

Comments

This is a read-only property that will return the maximum value that can be set to this `FireiFeature`, through the `Value` property.

Visual Basic syntax

```
Dim MaxValue As UInteger
MaxValue = Feature.MaxValue
```

C# syntax

```
uint MaxValue = Feature.MaxValue;
```

Absolute property

Prototype

```
Boolean Absolute
```

Comments

This is a read/write property that will retrieve or set whether this `FireiFeature` will use “absolute” values when setting its value. If set to true, the value of the feature is then read and set through the `AbsoluteValue` property, instead of the `Value` property.

This property is not available if `HasAbsolute` is false.

Visual Basic syntax

```
Dim Absolute As Boolean
Absolute = Feature.Absolute                    'get
Feature.Absolute = True                        'set
```

C# syntax

```
bool Absolute = Feature.Absolute;              //get
Feature.Absolute = true;                       //set
```

Enabled property

Prototype

```
Boolean Enabled
```

Comments

This is a read/write property that will retrieve or set whether this `FireiFeature` is on or off. If set to false a call to any of its properties or methods could result in an exception.

This property is not available if `HasOnOff` is false.

Visual Basic syntax

```
Dim Enabled As Boolean
Enabled = Feature.Enabled                      'get
Feature.Enabled = True                         'set
```

C# syntax

```
bool Enabled = Feature.Enabled;                //get
Feature.Enabled = true;                        //set
```

AutoMode property

Prototype

```
Boolean AutoMode
```

Comments

This is a read/write property that will retrieve or set whether this `FireiFeature` will be setting its value automatically or not. If it is set to true, the `Value` property cannot be set, or an exception will be thrown.

This property is not available if `HasAuto` is false.

Visual Basic syntax

```
Dim Auto As Boolean
Auto = Feature.AutoMode                    'get
Feature.AutoMode = True                    'set
```

C# syntax

```
bool Auto = Feature.AutoMode;              //get
Feature.AutoMode = true;                   //set
```

Value property

Prototype

```
uint Value
```

Comments

This is a read/write property that will retrieve or set the value of this `FireiFeature`. In order to read the value, the `CanRead` property must be true. In order to set the value, the `HasManual` property must be true, and the `AutoMode` property must be false. The value being set must also reside inside the boundaries set by the `MinValue` and `MaxValue` properties.

Visual Basic syntax

```
Dim Value As UInteger
Value = Feature.Value                      'get
Feature.Value = Feature.MaxValue           'set
```

C# syntax

```
uint Value = Feature.Value;                //get
Feature.Value = Feature.MaxValue;          //set
```

HasSoftAbsolute property

Prototype

```
Boolean HasSoftAbsolute
```

Comments

This is a read-only property that will return whether this `FireiFeature` supports setting its value through "absolute" values, calculated through the API[3].

If this property is false, the `MinAbsoluteValue`, `MaxAbsoluteValue` and `SoftAbsolute` properties are inaccessible.

Visual Basic syntax

```
Dim HasSoftAbsolute As Boolean
HasSoftAbsolute = Feature.HasSoftAbsolute
```

[3] This feature is called by the various Unibrain APIs "SoftAbsolute". It is only available on specific cameras, mostly Unibrain models. The camera in effect works in its regular manual values mode; the interface with the program operates in absolute mode, and the underlying API takes care of the value adaptation.

C# syntax

```
bool HasSoftAbsolute = Feature.HasSoftAbsolute;
```

SoftAbsolute property

Prototype

```
Boolean SoftAbsolute
```

Comments

This is a read/write property that will turn the SoftAbsolute feature on and off, if supported. Trying to set this feature when `HasSoftAbsolute` is false will result in an exception.

If this property is true, the `AbsoluteValue` property is used to set the feature value instead of the `Value` property.

Visual Basic syntax

```
Dim SoftAbsolute As Boolean
SoftAbsolute = Feature.SoftAbsolute                    'get
Feature.SoftAbsolute = True                            'set
```

C# syntax

```
bool SoftAbsolute = Feature.SoftAbsolute;              //get
Feature.SoftAbsolute = True;                           //set
```

ValueString property

Prototype

```
String ValueString
```

Comments

This is a read/write property that will retrieve or set the value of the feature through a textual representation of it. The string will include the unit of the feature if in absolute mode.

Through this property the value can be read or set either in Absolute, SoftAbsolute or Relative mode, albeit with different formats.

The same limitations as in the `Value` and `AbsoluteValue` properties apply.

Visual Basic syntax

```
Dim ValueString As String
ValueString = Feature.ValueString                      'get
Feature.ValueString = ValueString                      'set
```

C# syntax

```
string ValueString = Feature.ValueString;              //get
Feature.ValueString = ValueString;                     //set
```

MinValueString property

Prototype

```
String MinValueString
```

Comments

This is a read-only property that will retrieve the minimum allowed value of the feature as a textual representation. The string will include the unit of the feature if in absolute mode.

The same limitations as in the `MinValue` and `MinAbsoluteValue` properties apply.

Visual Basic syntax

```
Dim MinValueString As String
MinValueString = Feature.MinValueString
```

C# syntax

```
string MinValueString = Feature.MinValueString;
```

MaxValueString property

Prototype

```
String MaxValueString
```

Comments

This is a read-only property that will retrieve the maximum allowed value of the feature as a textual representation. The string will include the unit of the feature if in absolute mode.

The same limitations as in the `MaxValue` and `MaxAbsoluteValue` properties apply.

Visual Basic syntax

```
Dim MaxValueString As String
MaxValueString = Feature.MaxValueString
```

C# syntax

```
string MaxValueString = Feature.MaxValueString;
```

Unit property

Prototype

```
FireiFeatureUnit Unit
```

Comments

This is a read-only property that will retrieve the unit of the value of the feature as `FireiFeatureUnit` enumerated values.

Possible values returned are:

```
None,
FractionPercent,
ExposureValue,
Kelvin,
Degree,
Time,
Decibel,
FStops,
Distance,
ActualPercent
```

Visual Basic syntax

```
Dim Unit As FireiFeatureUnit
Unit = Feature.Unit
```

C# syntax

```
FireiFeatureUnit Unit = Feature.Unit;
```

AbsoluteValue property

Prototype

```
float AbsoluteValue
```

Comments

This is a read/write property that will retrieve or set the absolute value of this `FireiFeature`. For this property to be enabled instead of the `Value` property, the `Absolute` or `SoftAbsolute` properties must be true. Additionally, the value must reside in the boundaries set by the `MinAbsoluteValue` and `MaxAbsoluteValue` properties.

Since absolute values have fractional parts, this value is represented through a single-precision floating point number.

Visual Basic syntax

```
Dim Value As Single
Value = Feature.AbsoluteValue                       'get
Feature.AboluteValue = Feature.MaxAbsoluteValue     'set
```

C# syntax

```
float Value = Feature.AbsoluteValue;                //get
Feature.AboluteValue = Feature.MaxAbsoluteValue;    //set
```

MinAbsoluteValue property

Prototype

```
float MinAbsoluteValue
```

Comments

This is a read-only property that will return the minimum value that can be set to this `FireiFeature`, through the `AbsoluteValue` property.

Since absolute values have fractional parts, this value is represented through a single-precision floating point number.

Visual Basic syntax

```
Dim MinValue As Single
MinValue = Feature.MinAbsoluteValue
```

C# syntax

```
float MinValue = Feature.MinAbsoluteValue;
```

MaxAbsoluteValue property

Prototype

```
float MaxAbsoluteValue
```

Comments

This is a read-only property that will return the maximum value that can be set to this `FireiFeature`, through the `AbsoluteValue` property.

Since absolute values have fractional parts, this value is represented through a single-precision floating point number.

Visual Basic syntax

```
Dim MaxValue As Single
MaxValue = Feature.MaxAbsoluteValue
```

C# syntax

```
float MaxValue = Feature.MaxAbsoluteValue;
```

Reload method

Prototype

```
Reload()
```

Comments

This method can be used to read the current value and absolute value (if applicable, along with the minimum and maximum absolute values) of the camera feature.

It is particularly useful after a call to `OnePush` or if the camera is in `AutoMode`, as reading the value through the Value property will not return the current value on the camera, rather the value stored in the `FireiFeature` object. This is done for bus bandwidth conservation.

Visual Basic syntax

```
Feature.Reload()
```

C# syntax

```
Feature.Reload();
```

OnePush method

Prototype

```
OnePush()
```

Comments

This method, when called, will perform a complete one-push operation on the camera for the feature[4].

[4] A one push operation is two-step: first the register for the one push is set, and then the acknowledge register is repeatedly read until the operation is finished. The OnePush method does these two steps in one call and returns when the one push operation is finished.

This method is available only if `HasOnePush` is true; calling it otherwise will result in an exception.

Visual Basic syntax

```
Feature.OnePush()
```

C# syntax

```
Feature.OnePush();
```

FireiFrame

The `FireiFrame` object contains a single camera frame, allowing the programmer to read and write the image data pixel by pixel. It also provides some useful methods that help manipulate the image, e.g., draw lines, rectangles or print text on the image.

This object cannot be constructed directly; it is constructed and maintained internally by the SDK, and passed through the `FrameReceived` event of `FireiCamera`. It can be considered valid throughout the context of the event handler – but it cannot be stored in memory for later use.

GetPixel method

Prototype

```
Color GetPixel(ushort x, ushort y)
```

Comments

This method returns the color value as a `System.Drawing.Color` object of a given by coordinates pixel of the image buffer.

x and y must be within the boundaries of the image, starting from 0. The maximum width can be supplied either by the camera (`GetCurrentResolution` method), or the `FireiStreamFormat` properties `Width` and `Height`.

Visual Basic syntax

```
Dim Color As System.Drawing.Color
Color = Frame.GetPixel(0, 0)
```

C# syntax

```
System.Drawing.Color Color = Frame.GetPixel(0, 0);
```

SetPixel method

Prototype

```
SetPixel(ushort x, ushort y, System.Drawing.Color color)
```

Comments

This method sets the color value from a `System.Drawing.Color` object to a given by coordinates pixel of the image buffer.

x and y must be within the boundaries of the image, starting from 0. The maximum width can be supplied either by the camera (`GetCurrentResolution` method), or the `FireiStreamFormat` properties `Width` and `Height`.

Visual Basic syntax

```
Frame.SetPixel(0, 0, Drawing.Color.Black)
```

C# syntax

```
Frame.SetPixel(0, 0, Drawing.Color.Black);
```

GetRGB method

Prototype

```
GetRGB(ushort x, ushort y, out byte red, out byte green, out byte blue)
```

Comments

This method returns the color value as a red, green and blue components of a given by coordinates pixel of the image buffer.

x and y must be within the boundaries of the image, starting from 0. The maximum width can be supplied either by the camera (`GetCurrentResolution` method), or the `FireiStreamFormat` properties `Width` and `Height`.

Visual Basic syntax

```
Dim Red, Green, Blue As Byte
Frame.GetRGB(0, 0, Red, Green, Blue)
```

C# syntax

```
byte Red, Green, Blue;
Frame.GetRGB(0, 0, out Red, out Green, out Blue);
```

SetRGB method

Prototype

```
SetRGB(ushort x, ushort y, byte red, byte green, byte blue)
```

Comments

This method sets the color value of a given by coordinates pixel of the image buffer, using red, green and blue components.

x and y must be within the boundaries of the image, starting from 0. The maximum width can be supplied either by the camera (`GetCurrentResolution` method), or the `FireiStreamFormat` properties `Width` and `Height`.

Visual Basic syntax

```
Frame.SetRGB(0, 0, 0, 0, 0)
```

C# syntax

```
Frame.SetRGB(0, 0, 0, 0, 0);
```

SaveToFile method

Prototype

```
SaveToFile(string filename)
```

Comments

This method will save the contents of the `FireiFrame` to disk, as a regular Windows uncompressed BMP file. The filename must be supplied, with or without the .bmp extension (the SDK will add it if it was omitted).

If for some reason the operation is unsuccessful, an exception will be thrown.

Visual Basic syntax

```
Frame.SaveToFile("c:\Frame1.bmp")
```

C# syntax

```
Frame.SaveToFile("c:\Frame1.bmp");
```

FlipHorizontally method

Prototype

```
FlipHorizontally()
```

Comments

This method will invert the frame in-place along the y-axis (horizontally).

Visual Basic syntax

```
Frame.FlipHorizontally()
```

C# syntax

```
Frame.FlipHorizontally();
```

FlipVertically method

Prototype

```
FlipVertically()
```

Comments

This method will invert the frame in-place along the x-axis (vertically).

Visual Basic syntax

```
Frame.FlipVertically()
```

C# syntax

```
Frame.FlipVertically();
```

Negative method

Prototype

```
Negative()
```

Comments

This method will invert the color information of the frame. The result will be similar to a photographic negative.

Visual Basic syntax

```
Frame.Negative()
```

C# syntax

```
Frame.Negative();
```

ToBitmap method

Prototype

```
System.Drawing.Bitmap ToBitmap()
```

Comments

This method will return the frame data as a `System.Drawing.Bitmap` object. Any subsequent alterations on the returned object will not be reflected on the image of the Frame. This object can be stored for later use and it is the programmer's responsibility to dispose of.

Visual Basic syntax

```
Dim Picture As System.Drawing.Bitmap
Picture = Frame.ToBitmap()
```

C# syntax

```
System.Drawing.Bitmap Picture = Frame.ToBitmap();
```

GetGraphics method

Prototype

```
System.Drawing.Graphics GetGraphics()
```

Comments

This method will return the frame data as a `System.Drawing.Graphics` object. Any subsequent alterations on this object will be reflected on the image of the frame, so it should be considered volatile and not stored for later use or disposed of.

Visual Basic syntax

```
Dim Picture As System.Drawing.Graphics
Picture = Frame.GetGraphics()
```

C# syntax

```
System.Drawing.Graphics Picture = Frame.GetGraphics();
```

DrawLine method

Prototype

```
DrawLine(
      short fromX,
      short FromY,
      short toX,
      short ToY,
      System.Drawing.Color color
      )
```

Comments

This method will draw a line from a point defined by a set of coordinates (fromX, fromY) to another point defined by a set of coordinates(toX, toY), using the color defined by a `System.Drawing.Color` object.

The coordinates given as the 4 parameters must reside in the frame boundaries, otherwise an exception will be thrown.

Visual Basic syntax

```
Frame.DrawLine(0, 0, 50, 50, Drawing.Color.Black)
```

C# syntax

```
Frame.DrawLine(0, 0, 50, 50, Drawing.Color.Black);
```

DrawString method

Prototype

```
DrawString(
      string text,
      short x,
      short y,
      System.Drawing.Font font,
      System.Drawing.Color color
      )
```

Comments

This method will draw a string of text on a point on the frame defined by a set of coordinates (x, y), using the color defined by a `System.Drawing.Color` object and a valid `System.Drawing.Font` object.

The coordinates given as the 2 parameters must reside in the frame boundaries, otherwise an exception will be thrown.

Visual Basic syntax

```
Frame.DrawString("Foo", 0, 0, Me.Font, Drawing.Color.Black)
```

C# syntax

```
Frame.DrawString("Foo", 0, 0, this.Font, Drawing.Color.Black);
```

DrawRectangle method

Prototype

```
DrawRectangle(
      short x,
      short y,
      short width,
      short height,
      System.Drawing.Color color,
      bool filled
      )
```

Comments

This method will draw a rectangle, with its top-left corner being at a point defined by a set of coordinates (x, y) having a specific width and height and using the color defined by a `System.Drawing.Color` object. Additionally, it can be specified whether this rectangle will be empty (just its outline drawn) or filled.

The coordinates defined by the 4 parameters given must reside in the frame boundaries, otherwise an exception will be thrown.

Visual Basic syntax

```
Frame.DrawRectangle(0, 0, 50, 50, Drawing.Color.Black, False)
```

C# syntax

```
Frame.DrawRectangle(0, 0, 50, 50, Drawing.Color.Black, false);
```

DrawLineRGB method

Prototype

```
DrawLineRGB(
      short fromX,
      short fromY,
      short toX,
      short toY,
      byte red,
      byte green,
      byte blue
      )
```

Comments

This method will draw a line from a point defined by a set of coordinates (fromX, fromY) to another point defined by a set of coordinates(toX, toY), using the color defined by its red, green and blue components.

The coordinates given as the 4 parameters must reside in the frame boundaries, otherwise an exception will be thrown.

Visual Basic syntax

```
Frame.DrawLineRGB(0, 0, 50, 50, 0, 0, 0)
```

C# syntax

```
Frame.DrawLineRGB(0, 0, 50, 50, 0, 0, 0);
```

DrawStringRGB method

Prototype

```
DrawString(
      string text,
      short x,
      short y,
      System.Drawing.Font font,
      byte red,
      byte green,
      byte blue
      )
```

Comments

This method will draw a string of text on a point on the frame defined by a set of coordinates (x, y), using the color defined by its red, green and blue components, and a valid `System.Drawing.Font` object.

The coordinates given as the 2 parameters must reside in the frame boundaries, otherwise an exception will be thrown.

Visual Basic syntax

```
Frame.DrawStringRGB("Foo", 0, 0, Me.Font, 0, 0, 0)
```

C# syntax

```
Frame.DrawStringRGB("Foo", 0, 0, this.Font, 0, 0, 0);
```

DrawRectangleRGB method

Prototype

```
DrawRectangle(
      short x,
      short y,
      short width,
      short height,
      byte red,
      byte green,
      byte blue,
      bool filled
      )
```

Comments

This method will draw a rectangle, with its top-left corner being at a point defined by a set of coordinates (x, y) having a specific width and height and using the color defined by its red, green and blue components. Additionally, it can be specified whether this rectangle will be empty (just its outline drawn) or filled.

The coordinates defined by the 4 parameters given must reside in the frame boundaries, otherwise an exception will be thrown.

Visual Basic syntax

```
Frame.DrawRectangleRGB(0, 0, 50, 50, 0, 0, 0, False)
```

C# syntax

```
Frame.DrawRectangleRGB(0, 0, 50, 50, 0, 0, 0, false);
```

FireiCamera

The `FireiCamera` object encapsulates all functionality pertaining to the camera itself; therefore it could be considered to be the "central" object of the SDK.

It acts as a factory for many other objects: `FireiFeature`, `FireiStreamFormat` and `FireiTrigger`.

It cannot be constructed directly however; the `FireiManager` object is responsible for constructing a `FireiCamera` instance.

Besides having a great number of properties and methods, it also implements three events: `FrameReceived`, BufferReceived, and `CameraRemoved`. The first, if handled, will be fired every time a new frame is received by the SDK; the second will be fired every time a new frame is received by the SDK, but before converting it to a display-compatible format; and the third will be fired if the camera is physically removed from the system.

GUID property

Prototype

```
FireiGUID GUID
```

Comments

This is a read-only property that can be used to retrieve the GUID of the camera. It constructs and initializes a `FireiGUID` object internally and then returns it.

Visual Basic syntax

```
Dim GUID As FireiGUID
Set GUID = Camera.GUID
```

C# syntax

```
FireiGUID Set GUID = Camera.GUID;
```

Vendor property

Prototype

```
string Vendor
```

Comments

This is a read-only property that can be used to retrieve the vendor name of the camera, as a string.

Visual Basic syntax

```
Dim VendorName As String
VendorName = Camera.Vendor
```

C# syntax

```
string VendorName = Camera.Vendor;
```

Model property

Prototype

```
string Model
```

Comments

This is a read-only property that can be used to retrieve the model name of the camera, as a string.

Visual Basic syntax

```
Dim ModelName As String
ModelName = Camera.Model
```

C# syntax

```
string ModelName = Camera.Model;
```

Serial property

Prototype

```
uint Serial
```

Comments

This is a read-only property that can be used to retrieve the serial number of the camera, as a uint.

Visual Basic syntax

```
Dim SerialNo As UInteger
SerialNo = Camera.Serial
```

C# syntax

```
uint SerialNo = Camera.Serial;
```

StreamFormat property

Prototype

```
FireiStreamFormat StreamFormat
```

Comments

This is a read/write property that will retrieve or set the currently selected `FireiStreamFormat` of this `FireiCamera`. Through this `FireiStreamFormat`, all the streaming format functionality of the camera is exposed to the programmer.

The "set" operation of this property is the equivalent of calling the `Save` method of `FireiStreamFormat`.

Visual Basic syntax

```
Dim StreamFormat As FireiStreamFormat
StreamFormat = Camera.StreamFormat                    'get
Camera.StreamFormat = StreamFormat                    'set
```

C# syntax

```
FireiStreamFormat StreamFormat = Camera.StreamFormat;       //get
Camera.StreamFormat = StreamFormat;                         //set
```

Trigger property

Prototype

```
FireiTrigger Trigger
```

Comments

This is a read-only property that will retrieve the currently selected `FireiTrigger` of this `FireiCamera`. Through this `FireiTrigger`, all the trigger functionality of the camera is exposed to the programmer.

Visual Basic syntax

```
Dim Trigger As FireiTrigger
Trigger = Camera.Trigger                                    'get
Camera.Trigger = Trigger                                    'set
```

C# syntax

```
FireiTrigger Trigger = Camera.Trigger;                      //get
Camera.Trigger = Trigger;                                   //set
```

AutoExposure property

Prototype

```
FireiFeature AutoExposure
```

Comments

This is a read-only property that can be used to retrieve directly the `FireiFeature` object for the AutoExposure feature of the camera.

Visual Basic syntax

```
Dim AutoExposure As FireiFeature
AutoExposure = Camera.AutoExposure
```

C# syntax

```
FireiFeature AutoExposure = Camera.AutoExposure;
```

Shutter property

Prototype

```
FireiFeature Shutter
```

Comments

This is a read-only property that can be used to retrieve directly the `FireiFeature` object for the Shutter feature of the camera.

Visual Basic syntax

```
Dim Shutter As FireiFeature
Shutter = Camera.Shutter
```

C# syntax

```
FireiFeature Shutter = Camera.Shutter;
```

Gain property

Prototype

```
FireiFeature Gain
```

Comments

This is a read-only property that can be used to retrieve directly the `FireiFeature` object for the Gain feature of the camera.

Visual Basic syntax

```
Dim Gain As FireiFeature
Gain = Camera.Gain
```

C# syntax

```
FireiFeature Gain = Camera.Gain;
```

Iris property

Prototype

```
FireiFeature Iris
```

Comments

This is a read-only property that can be used to retrieve directly the `FireiFeature` object for the Iris feature of the camera.

Visual Basic syntax

```
Dim Iris As FireiFeature
Iris = Camera.Iris
```

C# syntax

```
FireiFeature Iris = Camera.Iris;
```

ColorUB property

Prototype

```
FireiFeature ColorUB
```

Comments

This is a read-only property that can be used to retrieve directly the `FireiFeature` object for the U/B component of the Color feature of the camera.

Visual Basic syntax

```
Dim ColorUB As FireiFeature
ColorUB = Camera.ColorUB
```

C# syntax

```
FireiFeature ColorUB = Camera.ColorUB;
```

ColorVR property

Prototype

```
FireiFeature ColorVR
```

Comments

This is a read-only property that can be used to retrieve directly the `FireiFeature` object for the V/R component of the Color feature of the camera.

Visual Basic syntax

```
Dim ColorVR As FireiFeature
ColorVR = Camera.ColorVR
```

C# syntax

```
FireiFeature ColorVR = Camera.ColorVR;
```

Hue property

Prototype

```
FireiFeature Hue
```

Comments

This is a read-only property that can be used to retrieve directly the `FireiFeature` object for the Hue feature of the camera.

Visual Basic syntax

```
Dim Hue As FireiFeature
Hue = Camera.Hue
```

C# syntax

```
FireiFeature Hue = Camera.Hue;
```

Saturation property

Prototype

```
FireiFeature Saturation
```

Comments

This is a read-only property that can be used to retrieve directly the `FireiFeature` object for the Saturation feature of the camera.

Visual Basic syntax

```
Dim Saturation As FireiFeature
Saturation = Camera.Saturation
```

C# syntax

```
FireiFeature Saturation = Camera.Saturation;
```

Focus property

Prototype

```
FireiFeature Focus
```

Comments

This is a read-only property that can be used to retrieve directly the `FireiFeature` object for the Focus feature of the camera.

Visual Basic syntax

```
Dim Focus As FireiFeature
Focus = Camera.Focus
```

C# syntax

```
FireiFeature Focus = Camera.Focus;
```

Zoom property

Prototype

```
FireiFeature Zoom
```

Comments

This is a read-only property that can be used to retrieve directly the `FireiFeature` object for the Zoom feature of the camera.

Visual Basic syntax

```
Dim Zoom As FireiFeature
Zoom = Camera.Zoom
```

C# syntax

```
FireiFeature Zoom = Camera.Zoom;
```

Brightness property

Prototype

```
FireiFeature Brightness
```

Comments

This is a read-only property that can be used to retrieve directly the `FireiFeature` object for the Brightness feature of the camera.

Visual Basic syntax

```
Dim Brightness As FireiFeature
Brightness = Camera.Brightness
```

C# syntax

```
FireiFeature Brightness = Camera.Brightness;
```

Sharpness property

Prototype

```
FireiFeature Sharpness
```

Comments

This is a read-only property that can be used to retrieve directly the `FireiFeature` object for the Sharpness feature of the camera.

Visual Basic syntax

```
Dim Sharpness As FireiFeature
Sharpness = Camera.Sharpness
```

C# syntax

```
FireiFeature Sharpness = Camera.Sharpness;
```

Gamma property

Prototype

```
FireiFeature Gamma
```

Comments

This is a read-only property that can be used to retrieve directly the `FireiFeature` object for the Gamma feature of the camera.

Visual Basic syntax

```
Dim Gamma As FireiFeature
Gamma = Camera.Gamma
```

C# syntax

```
FireiFeature Gamma = Camera.Gamma;
```

RawConversion property

Prototype

```
FireiRawConversion RawConversion
```

Comments

This is a read/write property that can be used to retrieve or set the Bayer conversion method that will be employed in case it is set on the streaming format.

The Fire-i.net SDK uses the Bayer conversion algorithms implemented in the Firei.dll and DirectShow APIs as they are, benefitting automatically from future performance updates in them.

Possible values of the `FireiRawConversion` enumerated value are:

```
NearestNeighbor,
BilinearInterpolation,
SmoothHueTransition
```

The three implemented algorithms are ordered “Best Performance → Best Quality”. `BilinearInterpolation` is the default setting.

Please note that the `SmoothHueTransition` algorithm is only implemented in the Firei.dll API; if set while using the DirectShow API, an exception will be thrown.

Visual Basic syntax

```
Dim RawConversion As FireiRawConversion
RawConversion = Camera.RawConversion                        'get
Camera.RawConversion = FireiRawConversion.NearestNeighbor   'set
```

C# syntax

```
FireiRawConversion RawConversion = Camera.RawConversion;    //get
Camera.RawConversion = FireiRawConversion.NearestNeighbor;  //set
```

Icon property

Prototype

```
System.Drawing.Icon Icon
```

Comments

This is a read-only property that can be used to retrieve a `System.Drawing.Icon` object, depicting the camera itself. This picture is derived from an internal database of camera pictures that the Unibrain APIs maintain and is selected automatically, using the camera vendor and model.

Visual Basic syntax

```
Dim Icon As System.Drawing.Icon
Icon = Camera.Icon
```

C# syntax

```
System.Drawing.Icon Icon = Camera.Icon;
```

ReadRegister method

Prototype

```
FireiRegister ReadRegister(uint offset)
```

Comments

This is a method that can be used to retrieve any register of the camera, given its offset as a parameter.

It will read the value from the camera and initialize a `FireiFeature` object, returning it to the programmer. If for some reason the camera rejects the read request, an exception will be thrown (the exception information will contain the nature of the error).

Please note that the offset parameter is based at 0 – for reading the command registers of the camera (normally based at hex F0000000) the `ReadCommand` method can be used instead.

Visual Basic syntax

```
Dim Register As FireiRegister
Register = Camera.ReadRegister(&H404)
```

C# syntax

```
FireiRegister Register = Camera.ReadRegister(0x404);
```

WriteRegister method

Prototype

```
void WriteRegister(uint offset, FireiRegister register)
```

Comments

This is a method that can be used to set any register of the camera, given its offset as a parameter.

It will write the value to the camera given a valid `FireiFeature` object. If for some reason the camera rejects the write request, an exception will be thrown (the exception information will contain the nature of the error).

Please note that the offset parameter is based at 0 – for writing the command registers of the camera (normally based at hex F0000000) the `WriteCommand` method can be used instead.

Visual Basic syntax

```
Dim Register As FireiRegister
Register = new FireiRegister(0)
Camera.WriteRegister(&H404, Register)
```

C# syntax

```
FireiRegister Register = new FireiRegister(0);
Camera.WriteRegister(0x404, Register);
```

ReadCommand method

Prototype

```
FireiRegister ReadCommand(uint offset)
```

Comments

This is a method that can be used to retrieve any command register of the camera, given its offset as a parameter. As “command register” is defined a register with its offset based on the command-base register offset (normally hex F0000000).

It will read the value from the camera and initialize a `FireiFeature` object, returning it to the programmer. If for some reason the camera rejects the read request, an exception will be thrown (the exception information will contain the nature of the error).

The `ReadCommand` method has the exact same effect as the `ReadRegister` method, if the command-base offset is added to the offset parameter. Since this offset is theoretically variable (camera-specific), for command registers it is best to use the `ReadCommand` method.

Visual Basic syntax

```
Dim Register As FireiRegister
Register = Camera.ReadCommand(&H504)
```

C# syntax

```
FireiRegister Register = Camera.ReadCommand(&H504);
```

WriteCommand method

Prototype

```
void WriteCommand(uint offset, FireiRegister register)
```

Comments

This is method that can be used to set any command register of the camera, given its offset as a parameter. As "command register" is defined a register with its offset based on the command-base register offset (normally hex F0000000).

It will write the value to the camera, given a valid `FireiFeature` object. If for some reason the camera rejects the write request, an exception will be thrown (the exception information will contain the nature of the error).

The `WriteCommand` method has the exact same effect as the `WriteRegister` method, if the command-base offset is added to the offset parameter. Since this offset is theoretically variable (camera-specific), for command registers it is best to use the `WriteCommand` method.

Visual Basic syntax

```
Dim Register As FireiRegister
Register = new FireiRegister(0)
Camera.WriteCommand(&H504, Register)
```

C# syntax

```
FireiRegister Register = new FireiRegister(0);
Camera.WriteCommand(&H504, Register);
```

ReadBlock method

Prototype

```
byte[] ReadBlock(uint offset, uint size)
```

Comments

This is a method that can be used to retrieve a block of registers (quadlets) on the camera, given its offset and the number of bytes to read (size) as parameters. The size must be a multiple of 4.

Visual Basic syntax

```
Dim buffer() As Byte
buffer = Camera.ReadBlock(&HF0F00508, 12)
```

C# syntax

```
Byte[] buffer = Camera.ReadBlock(0xF0F00508, 12);
```

WriteBlock method

Prototype

```
void WriteBlock(uint offset, byte[] buffer)
```

Comments

This is a method that can be used to write a block of registers (quadlets) on the camera, given its offset and a byte buffer as parameters. The size of the buffer must be a multiple of 4.

Visual Basic syntax

```
Dim buffer() As Byte = New Byte(7) {&H82, &H0, &H0, &H10, &H82, &H0, &H0,
&H3A}
Camera.WriteBlock(&HF0F00508, buffer)
```

C# syntax

```
byte[] buffer = new byte[8] {0x82, 0, 0, 0x10, 0x82, 0, 0, 0x3A};
Camera.WriteBlock(0xF0F0081C, buffer);
```

GetMemoryPresetsCount method

Prototype

```
uint GetMemoryPresetsCount()
```

Comments

This method that can be used to retrieve the number of available memory presets of the camera.

If this number is 0, the camera does not support memory presets (rendering the `SaveToMemory` and `LoadFromMemory` methods inaccessible).

Visual Basic syntax

```
Dim NumOfMemoryPresets As UInteger
NumOfMemoryPresets = Camera.GetMemoryPresetsCount()
```

C# syntax

```
uint NumOfMemoryPresets = Camera.GetMemoryPresetsCount();
```

ToString method

Prototype

```
string ToString()
```

Comments

This method can be used to retrieve an SDK-constructed “friendly name” for the camera, as a string. This “friendly name” is a combination of the vendor name, the model name and the serial number of the camera.

A typical result is similar to:

```
Unibrain Fire-i 1.2 40045266
```

Visual Basic syntax

```
Dim FriendlyName As String
FriendlyName = Camera.ToString()
```

C# syntax

```
string FriendlyName = Camera.ToString();
```

GetFeature method

Prototype

```
FireiFeature GetFeature(string featureName)
```

Comments

This is a method that can be used to retrieve the `FireiFeature` object of any of features of the camera.

It takes the name (as a string) of the feature to locate the desired `FireiFeature` object. This name is the same as the one returned by the `Name` property of `FireiFeature`. If the parameter passed is not a recognized feature name, an exception will be thrown.

Visual Basic syntax

```
Dim Feature As FireiFeature
Feature = Camera.GetFeature("Shutter")
```

C# syntax

```
FireiFeature Feature = Camera.GetFeature("Shutter");
```

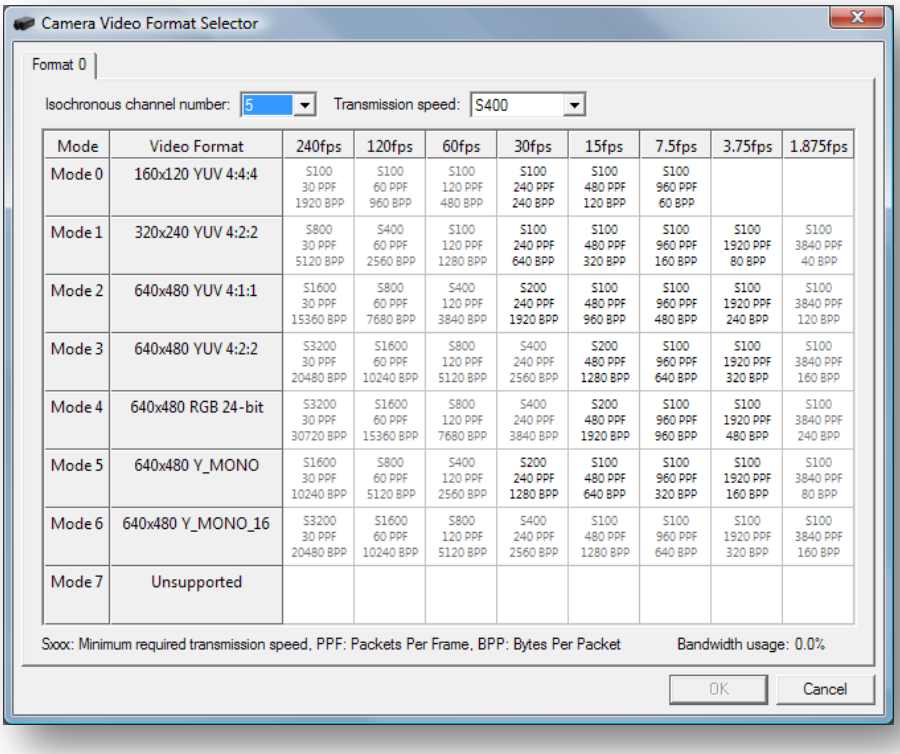
SelectStreamFormat method

Prototype

```
SelectStreamFormat()
```

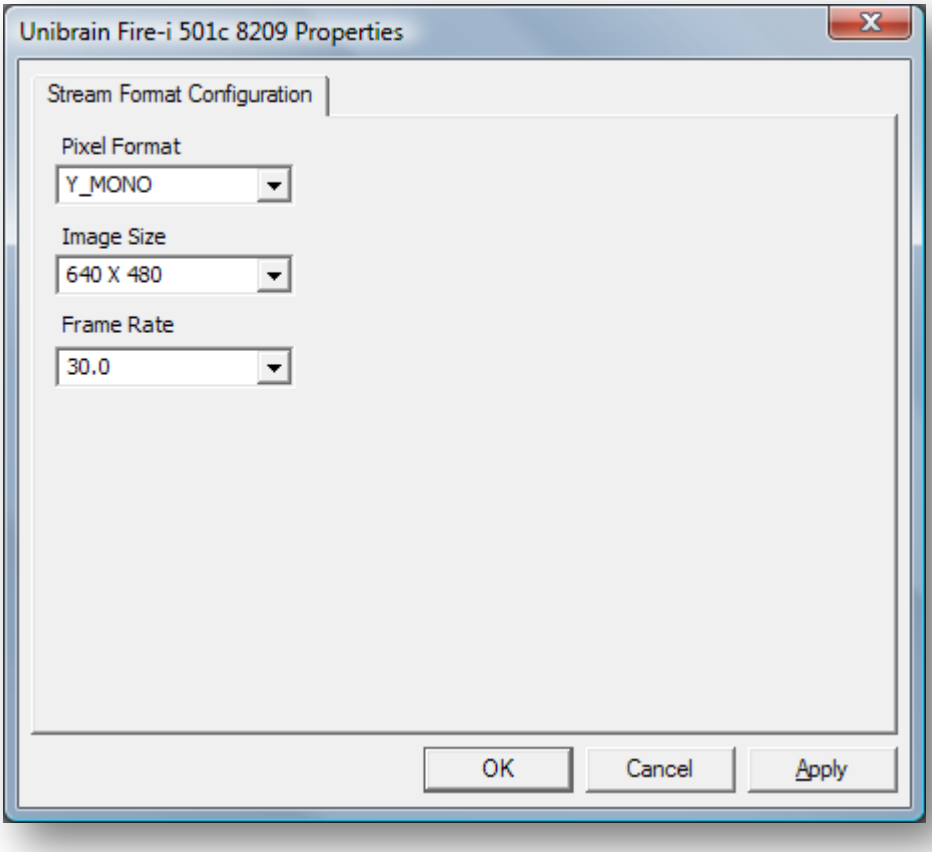
Comments

This method when called will bring up a “Stream Format Selector” dialog, as constructed and maintained internally by the APIs. Through this dialog, user selection of a streaming format is possible. The appearance of this dialog varies depending on the underlying API selected. If DLL mode is currently running, the dialog will look similar to:



Mode	Video Format	240fps	120fps	60fps	30fps	15fps	7.5fps	3.75fps	1.875fps
Mode 0	160x120 YUV 4:4:4	S100 30 PPF 1920 BPP	S100 60 PPF 960 BPP	S100 120 PPF 480 BPP	S100 240 PPF 240 BPP	S100 480 PPF 120 BPP	S100 960 PPF 60 BPP		
Mode 1	320x240 YUV 4:2:2	S800 30 PPF 5120 BPP	S400 60 PPF 2560 BPP	S100 120 PPF 1280 BPP	S100 240 PPF 640 BPP	S100 480 PPF 320 BPP	S100 960 PPF 160 BPP	S100 1920 PPF 80 BPP	S100 3840 PPF 40 BPP
Mode 2	640x480 YUV 4:1:1	S1600 30 PPF 15360 BPP	S800 60 PPF 7680 BPP	S400 120 PPF 3840 BPP	S200 240 PPF 1920 BPP	S100 480 PPF 960 BPP	S100 960 PPF 480 BPP	S100 1920 PPF 240 BPP	S100 3840 PPF 120 BPP
Mode 3	640x480 YUV 4:2:2	S3200 30 PPF 20480 BPP	S1600 60 PPF 10240 BPP	S800 120 PPF 5120 BPP	S400 240 PPF 2560 BPP	S200 480 PPF 1280 BPP	S100 960 PPF 640 BPP	S100 1920 PPF 320 BPP	S100 3840 PPF 160 BPP
Mode 4	640x480 RGB 24-bit	S3200 30 PPF 30720 BPP	S1600 60 PPF 15360 BPP	S800 120 PPF 7680 BPP	S400 240 PPF 3840 BPP	S200 480 PPF 1920 BPP	S100 960 PPF 960 BPP	S100 1920 PPF 480 BPP	S100 3840 PPF 240 BPP
Mode 5	640x480 Y_MONO	S1600 30 PPF 10240 BPP	S800 60 PPF 5120 BPP	S400 120 PPF 2560 BPP	S200 240 PPF 1280 BPP	S100 480 PPF 640 BPP	S100 960 PPF 320 BPP	S100 1920 PPF 160 BPP	S100 3840 PPF 80 BPP
Mode 6	640x480 Y_MONO_16	S3200 30 PPF 20480 BPP	S1600 60 PPF 10240 BPP	S800 120 PPF 5120 BPP	S400 240 PPF 2560 BPP	S100 480 PPF 1280 BPP	S100 960 PPF 640 BPP	S100 1920 PPF 320 BPP	S100 3840 PPF 160 BPP
Mode 7	Unsupported								

If on the other hand the DirectShow mode is currently running, the dialog will look similar to:



Upon the return of the method, the streaming format that the user selected will be also set on the camera automatically. It can be retrieved by the programmer, if desired, through the `StreamFormat` property. If the user did not select a format, pressing "Cancel", there is no change made to the camera.

Visual Basic syntax

```
Camera.SelectStreamFormat()
```

C# syntax

```
Camera.SelectStreamFormat();
```

AttachPreviewCtrl method

Prototype

```
AttachPreviewCtrl(FireiPreviewControl previewControl)
```

Comments

This method can be used to attach a `FireiPreviewControl` object to this `FireiCamera`, for video preview purposes.

Visual Basic syntax

```
Camera.AttachPreviewCtrl(FireiPreviewCtrl1)
```

C# syntax

```
Camera.AttachPreviewCtrl(FireiPreviewCtrl1);
```

Run method

Prototype

```
Run()
```

Comments

This method will start the streaming of the camera.

If the `FireiCamera` object is attached to a `FireiPreviewControl`, the preview on that control will also start.

If the camera is already running, a call to `Run` will have no effect[5]. If the camera for some reason cannot start, or the preview window (if applicable) is invalid in some way, an exception will be thrown.

Visual Basic syntax

```
Camera.Run()
```

C# syntax

```
Camera.Run();
```

Stop method

Prototype

```
Stop()
```

Comments

This method will stop the streaming of the camera.

If the `FireiCamera` object is attached to a `FireiPreviewControl`, the preview on that control will also stop.

If the camera is not running, a call to `Stop` will have no effect. If the camera for some reason cannot stop, an exception will be thrown.

Visual Basic syntax

```
Camera.Stop()
```

C# syntax

```
Camera.Stop();
```

IsRunning method

Prototype

```
Boolean IsRunning()
```

[5] This is not entirely true; if the camera was started while not being attached to a `FireiPreviewControl`, then attached and Run was called again, the preview will start.

Comments

This method will return whether the camera is currently running.

Please note that `IsRunning` will return true even if the camera was not started during runtime (i.e., it was running before the program was started).

Visual Basic syntax

```
Dim Running As Boolean
Running = Camera.IsRunning()
```

C# syntax

```
bool Running = Camera.IsRunning();
```

GetStreamFormats method

Prototype

```
System.Collections.Generic.List<FireiStreamFormat>.GetStreamFormats()
```

Comments

This method can be used to retrieve a list of the supported streaming formats of the object.

The returned list contains only the supported formats of the specific camera at the time of the call to `GetStreamFormats`. Since the actual speed of the bus the camera is connected to can vary depending on various parameters, the supported formats will also vary, depending on the speed of the bus (besides the make and model of the camera).

Visual Basic syntax

```
Dim StreamFormats As System.Collections.Generic.List(Of FireiStreamFormat)
StreamFormats = Camera.GetStreamFormats()
```

C# syntax

```
System.Collections.Generic.List<FireiStreamFormat> StreamFormats;
StreamFormats = Camera.GetStreamFormats();
```

DisplayProperties method

Prototype

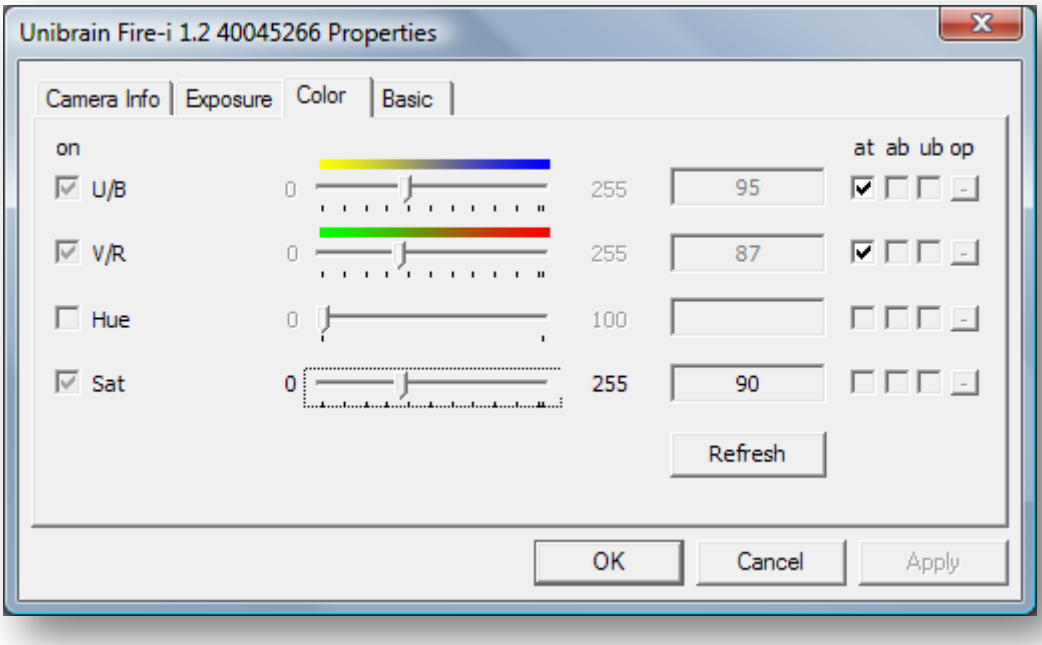
```
DisplayProperties()
```

Comments

This method when called will bring up a “Display Properties” dialog for the camera. This dialog can then be used to set the various feature values of the camera.

Unlike the other UI-driven methods of the SDK (namely `SelectCamera` and `SelectStreamFormat`), this dialog has an immediate effect on the camera as the values and settings of the features are changed. Additionally, the method opens up the dialog and returns immediately. The dialog will remain open until the user closes it, affecting the camera whether it is running or not.

The presented dialog is almost exactly the same in appearance, regardless if DirectShow or DLL is the currently selected underlying mode, and it looks similar to:



Visual Basic syntax

```
Camera.DisplayProperties()
```

C# syntax

```
Camera.DisplayProperties();
```

IsFeatureSupported method

Prototype

```
Boolean IsFeatureSupported(string featureName)
```

Comments

This method will return whether a given feature is supported. The feature parameter is defined as a string, containing the name of the feature. This name is the same as the one the `Name` property of `FireiFeature` would return.

Visual Basic syntax

```
Dim Supported As Boolean
Supported = Camera.IsFeatureSupported("Shutter")
```

C# syntax

```
bool Supported = Camera.IsFeatureSupported("Shutter");
```

GetFeaturesEnumerator method

Prototype

```
System.Collections.Generic.List<FireiFeature> GetFeatures(
```

```
bool supportedOnly,
FireiFeatureGroup featureGroup
)
```

Comments

This method will construct and return a generic list of `FireiFeatures`. It will contain the `FireiFeature` objects depending on the parameters passed. The first parameter will toggle whether all features will be returned, or only the ones supported by the camera. Additionally, since the features are divided in 3 groups (Exposure, Color, Basic), through the featureGroup parameter, further reduction on the returned set can be achieved.

The possible values for the featureGroup parameter are:

```
All,
Exposure,
Color,
Basic
```

If `FireiFeatureGroup.All` is selected, all features will be contained in the list, otherwise only the ones defined by the `FireiFeatureGroup` value.

Visual Basic syntax

```
Dim Features As System.Collections.Generic.List(Of FireiFeature)
Features = Camera.GetFeatures(true, FireiFeatureGroup.All)
```

C# syntax

```
System.Collections.Generic.List<FireiFeature> Features;
Features = Camera.GetFeatures(true, FireiFeatureGroup.All);
```

GetCurrentResolution method

Prototype

```
GetCurrentResolution(out ushort width, out ushort height)
```

Comments

This method will return (as `out` parameters) the width and height of the camera stream format resolution in pixels.

Visual Basic syntax

```
Dim Width, Height As UShort
Camera.GetCurrentResolution(width, height)
```

C# syntax

```
ushort Width, Height;
Camera.GetCurrentResolution(out width, out height);
```

GetCameraPhoto method

Prototype

```
System.Drawing.Bitmap GetCameraPhoto(
```

```
ushort edgeSize,
byte red,
byte green,
byte blue,
byte alpha
)
```

Comments

This method constructs and returns a `System.Drawing.Bitmap` object, that contains a photograph of the camera connected. This photograph is derived from an internal database that Unibrain maintains, and the SDK utilizes.

The resulting picture will be square (width equals height); the size of each edge can be defined. Additionally, the programmer can choose the background color of the picture (for rendering transparently on a control), in all 4 color components, red, green, blue and alpha.

Visual Basic syntax

```
Dim Photo As System.Drawing.Bitmap
Photo = Camera.GetCameraPhoto(80, 255, 255, 255, 255)
```

C# syntax

```
System.Drawing.Bitmap Photo;
Photo = Camera.GetCameraPhoto(80, 255, 255, 255, 255);
```

SaveToMemory method

Prototype

```
SaveToMemory(byte channelNumber)
```

Comments

This method will save all current features to a memory preset channel on the camera. Since channel 0 is reserved for the camera default values, it cannot be used as a parameter in `SaveToMemory`.

In order to ascertain what the maximum channel number allowed is, the `GetMemoryPresetsCount` method can be used: in effect, the number representing the `GetMemoryPresetsCount` value is the maximum allowed value for `SaveToMemory`.

If for any reason the camera cannot save the settings, an exception will be thrown.

Visual Basic syntax

```
Camera.SaveToMemory(1)
```

C# syntax

```
Camera.SaveToMemory(1);
```

LoadFromMemory method

Prototype

```
LoadFromMemory(byte channelNumber)
```

Comments

This method will load all current features from a memory preset channel of the camera. Since channel 0 is reserved for the camera default values, it cannot be used as a parameter in `LoadFromMemory`.

In order to ascertain what the maximum channel number allowed is, the `GetMemoryPresetsCount` method can be used: in effect, the number representing the `GetMemoryPresetsCount` value is the maximum allowed value for `LoadFromMemory`.

If for any reason the camera cannot load the settings, an exception will be thrown.

Visual Basic syntax

```
Camera.LoadFromMemory(1)
```

C# syntax

```
Camera.LoadFromMemory(1);
```

SaveToXML method

Prototype

```
SaveToXML(string filename, bool overwrite)
```

Comments

This method will save all current features to a file on disk, in XML format. The first parameter passed represents the filename to save as, and it cannot be empty. The second parameter passed will check if a file with the same filename already exists. If overwrite is false and the file already exists, an exception will be thrown.

The resulting XML file has 100% compatible format with the XML files saved and loaded from the other Unibrain tools, like FireIIDC and Fire-i Application.

If for any reason the file cannot be written to disk, an exception will be thrown.

Visual Basic syntax

```
Camera.SaveToXML("c:\settings.xml", True)
```

C# syntax

```
Camera.SaveToXML("c:\settings.xml", true);
```

LoadFromXML method

Prototype

```
LoadFromXML(string filename)
```

Comments

This method will load all features from an XML file on disk, and set them to the camera. The parameter passed represents the filename to load from, and it cannot be empty.

The XML file being loaded can be the one saved with `SaveToXML`, or any other XML files created by the other Unibrain tools, like FireIIDC and Fire-i Application, since these are 100% compatible.

If for any reason the settings cannot be loaded or set to the camera, an exception will be thrown.

Visual Basic syntax

```
Camera.LoadFromXML("c:\settings.xml")
```

C# syntax

```
Camera.LoadFromXML("c:\settings.xml");
```

RetrieveStreamFormat method

Prototype

```
FireiStreamFormat RetrieveStreamFormat(
      FireiPixelFormat pixelFormat,
      FireiResolution resolution
      )
```

Comments

This method will return a `FireiStreamFormat` object that has the given `FireiPixelFormat` and `FireiResolution` attributes, provided it is supported by the camera. If it is not supported, an exception will be thrown.

There is only a single `FireiStreamFormat` object having the same `FireiPixelFormat` and `FireiResolution` values.

Visual Basic syntax

```
Dim StreamFormat As FireiStreamFormat
StreamFormat = Camera.RetrieveStreamFormat(
      FireiPixelFormat.Y_MONO,
      FireiResolution.res320x240
      )
```

C# syntax

```
FireiStreamFormat StreamFormat = Camera.RetrieveStreamFormat(
      FireiPixelFormat.Y_MONO,
      FireiResolution.res320x240
      );
```

RetrieveStreamFormatFromIdentifier method

Prototype

```
FireiStreamFormat RetrieveStreamFormatFromIdentifier(
      uint identifier
      )
```

Comments

This method will return a `FireiStreamFormat` object that has the given unique identifier. This identifier can be retrieved by the `Identifier` property of `FireiStreamFormat` only.

If the identifier given is not valid, or if it points to a `FireiStreamFormat` that is not supported by the camera, an exception will be thrown.

Visual Basic syntax

```
Dim Identifier As UInteger
Identifier = Camera.StreamFormat.Identifier
Dim StreamFormat As FireiStreamFormat
StreamFormat = Camera.RetrieveStreamFormatFromIdentifier(Identifier)
```

C# syntax

```
uint Identifier = Camera.StreamFormat.Identifier;
FireiStreamFormat StreamFormat =
      Camera.RetrieveStreamFormatFromIdentifier(Identifier);
```

FrameReceived event

Prototype

```
event System.EventHandler<FrameReceivedEventArgs> FrameReceived
```

Comments

This event is fired whenever a frame is received from the camera, after it has been converted to RGB for viewing, but before having been actually sent to the screen (if attached to a `FireiPreviewControl`). It is therefore useful both to perform image processing using the provided `FrameReceivedEventArgs` instance and manipulation before display.

The supplied `FrameReceivedEventArgs` object can be considered valid in the context of the event handler. It contains a `Frame` property that can be used to access the `FireiFrame` object. Any changes made to it (through its methods) are then reflected on the preview screen (if applicable) upon exit from the event handler.

Syntax

Since event handlers are usually created automatically by the underlying programming environment, it would not be useful to provide sample code here.

BufferReceived event

Prototype

```
event System.EventHandler<BufferReceivedEventArgs> BufferReceived
```

Comments

This event is fired whenever a frame is received from the camera, before it has been converted to RGB for viewing. It is useful when direct access to the frame buffer being received is necessary, such as when working with Y-MONO-16 formats, since the actual buffer is available.

The supplied `BufferReceivedEventArgs` object can be considered valid in the context of the event handler. It contains a `Buffer` property that can be used to access an `IntPtr` pointing to the actual buffer. The framework-provided `System.Runtime.InteropServices.Marshal` class contains helpful methods that can be used to handle this buffer, such as `ReadByte` and `Copy`.

Syntax

Since event handlers are usually created automatically by the underlying programming environment, it would not be useful to provide sample code here.

CameraRemoved event

Prototype

```
event System.EventHandler CameraRemoved
```

Comments

This event is only ever fired once during the lifecycle of the `FireiCamera` object, if the camera is physically removed from the system.

Since calling any methods or properties of the camera after it has been removed would result in an error, this event notifies that the camera has been unplugged, giving the programmer the chance to handle the situation. Since the firing of this event might originate from a different thread than the one used while creating the camera object, the camera `Dispose` method should not be called, otherwise unexpected behavior could occur.

Syntax

Since event handlers are usually created automatically by the underlying programming environment, it would not be useful to provide sample code here.