

Approved/Godkänd
 Kjell Brunnström
 Issued by/Utfärdare
 Stéphane Junique

Distribution
 Public

Date/Datum
 2008-06-16
 Type of document/Dokumenttyp
 REPORT
 Reference/Referens

Rev
 1
 Page/Sida
 1

For information only/För kännedom

The IPTVinterface program Version 3.16

The IPTVinterface program Version 3.16.....	1
Introduction	3
Contacting us.....	3
Presentation of the program	3
License and copyright	3
IPTVInterface	4
The Matlab interface	4
The IPTVinterface class.....	5
The Init method.....	5
The RecordSequence method.....	5
The globalGraph4 class.....	5
The BuildGraph method.....	5
Using another set of filters	6
Adding support for other filters.....	7
The SampleGrabberCallback class.....	7
Overview of the execution flow of an instance	7
Use in the IPTVInterface class.....	8
The BufferCB method.....	8
The SetFileRecording method.....	8
The PreAllocateMemory method	8
The SetNbOfFrames method.....	8
The Dump_to_disk method.....	9
The CloseAvi method.....	9
The IsDone method.....	9
The GetCount method	9
The Reset method.....	9
The memManager class.....	9
The PreAllocate method.....	9
The Deallocate method	9
The AllocateFrame method.....	10
The FreeFrame method	10
Proposed design improvements.....	10
The aviWriter class	10
The Initialisation methods.....	10
The AddFrame and CloseAvi Methods.....	11
The sjdebug functions	11
The directshow_tools functions.....	11
Calling IPTVInterface from Matlab	11
IPTVcpp.....	12
The console-based program	12
Calling IPTVcpp	13
Compiling and using the code	13
Compiling the library and the standalone executable.....	13

Maturity and stability of the code.....	13
Conclusion	13
Acknowledgements	13
Future work	14
Memory management	14
Handling of codecs and filter chains	14
Appendix: the GNU General Public License v2.0	14
Preamble	14
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION.....	15
NO WARRANTY	18
END OF TERMS AND CONDITIONS	19
References.....	19

Introduction

Contacting us

The software package and all of its documentation are copyright Acreo AB, 2006-2008. If you have any question or want to provide us feedback, bug reports, bug fixes or success stories, please do not hesitate to contact us:

Kjell Brunnström (Kjell.Brunnstrom@acreo.se) or Stéphane Junique (Stephane.Junique@acreo.se)

Acreo AB

Electrum 236

SE-164 40 Kista

Sweden

www.acreo.se

Presentation of the program

IPTVInterface is a program which uses codecs installed on your Windows (2000 or later) system to receive, decode, capture and save a sequence of an IPTV channel into an uncompressed AVI file. It can be compiled either as a Matlab-callable library or as a standalone program.

IPTVInterface was originally written to work together with iVQM. iVQM is a video-quality monitoring program written by the US National Telecommunications and Information administration (NTIA) [1] and using computer-based tuner cards as the video source. We wished to use iVQM with Network-broadcast television channels (so-called IPTV). So we wrote IPTVInterface as a module providing a video files from an IPTV channel in a Matlab environment, and they were kind enough to add it as a video source in iVQM.

IPTVInterface is a Windows-only (2000 or later, tested on 2000 and XP) program. It manipulates video streams coming from the network using modules already installed on the operating system. In the Microsoft Multimedia Framework known as DirectShow, these modules are known as codecs (for coder-decoder). They are like building blocks using a standard interface and can be connected to each other to process the stream. Using this framework makes the program considerably simpler than writing all the processing blocks. Adding support for new codecs is also considerably easier than having to implement a new decoder. The downside is that the code is bound to the Microsoft platform.

License and copyright

The code of the IPTVInterface package is copyrighted by Acreo AB (2006-2008). It is licensed to you under the GNU General Public License version 2.0 (GNU GPL2), a copy of which is provided at the end of this document. It is provided to you without any warranty of any kind, not even of fitness for any particular purpose.

The codecs used in IPTVInterface as released here are from Elecard (www.elecard.com). You can install them on your system by installing their MPEG player. These codecs are used only as a demonstration of how codecs are used inside IPTVInterface. In conjunction with the media player, they are only licensed for individual, non-commercial use.

This license and its authors make no representation whatsoever of your rights to use a particular codec in conjunction with `IPTVInterface`. It is your responsibility to ensure that you have the right to use the codecs in your specific environment.

If you wish to use `IPTVInterface` outside of the terms of the GPL2, for example with a proprietary program, you are welcome to contact Acreo AB (www.acreo.se) to obtain a commercial license.

IPTVInterface

`IPTVInterface` is written as a dynamically-linked library for Matlab. It is composed of a few functions handling the interface with Matlab, as well as a small number of C++ classes. These classes are used directly or indirectly by the main class, `IPTVInterface`, which implements the real-time video recording functionality.

An instance of the `IPTVInterface` class is created as a global object in the main entry function called `mexFunction`. Simple calls to the object allow the codecs to be initialized and the recording to take place.

The `IPTVInterface` library totals 18 source files:

- **`mx_main.cpp,h`**: Main file, implements the interface with Matlab;
- **`IPTVInterface.cpp,h`**: implements the `IPTVInterface` class, which handles the real-time video recording;
- **`aviwriter.cpp,h`**: implements a class handling the writing of uncompressed AVI files;
- **`globalGraph4.cpp,h`**: implements a class which creates a chain of DirectShow codecs to receive an IPTV stream, uncompress it and copy it to memory.
- **`memManager.cpp,h`**: implements a class which handles the memory allocation to store the uncompressed video stream
- **`samplegrabbercallback.cpp,h`**: implements a class taking care of the video recording and called by one of the codecs during playback.
- **`directshow_tools.cpp,h`**: implements functions which handle DirectShow objects.
- **`sjdebug.cpp,h`**: implements functions which write debug information to a log file.
- **`filter_clsid.h`**: contains a list of references to DirectShow codecs
- **`iptverrors.h`**: contains a list of error codes.

The Matlab interface

The interface with Matlab is handled inside the `mx_main.cpp` file. Function `mexFunction`, which is the main entry point, is similar to a C-code main function. It receives the number of, and list of command-line arguments. The call arguments are first checked, then acted upon by either:

- Creating and initializing an `IPTVInterface` instance, if none exists.
- Requesting the `IPTVInterface` instance to record a number of video frames and save them as an AVI file. After the recording, the instance is deleted and a new one is created and initialized.
- Deleting the `IPTVInterface` instance, thereby releasing all the previously-allocated library resources.

The file contains a few helper functions used either to convert Matlab variables to standard C-style ones, or to display information on the Matlab console. The Matlab-related code is contained in `mx_main.cpp` and `mx_main.h`, and all other files are independent of Matlab.

The IPTVInterface class

`IPTVInterface` is the class holding the program general functionality. It contains only 3 publicly-accessible methods: one is `Init(...)`, which initializes the class. A second one is `RecordSequence(...)`, which records a given number of frames and saves them in an AVI file. And a third one is `GetChannel(...)` and provides the IP-address and port number currently used to receive the IPTV channel.

The Init method

A newly-created instance first needs to be initialized with a proper IP-address and port number and a debug level, using the `Init(...)` method. The IP-address and port number are those of the IPTV channel to be recorded. The debug level is an index used by most functions of the classes to properly indent messages in a log file. `Init(...)` stores the IPTV data inside the class instance. It then initializes the `DirectShow` subsystem, something that should be done once in a program before any `DirectShow`-specific object is used. An instance of the `SampleGrabberCallback` class is created. Finally, an instance of the `globalGraph4` class is created and initialized.

The `globalGraph4` class contains the real-time part of the program, including receiving, decoding and making a copy of the video stream. It is controlled in `IPTVInterface` through public methods and events.

The RecordSequence method

`RecordSequence(...)` is the method called after the `IPTVInterface` instance has been initialized, when it is time to record a number of frames. It is provided with a number of frames to record, the name of the AVI file to store the frames in, and a `debuglevel` index as previously discussed.

A few requests are sent to the `globalGraph4` instance to know the width and height of the video frames, so the proper memory buffer can be allocated to store them. The frame rate is also requested, as it is needed in the AVI header.

Then the recording starts in another execution thread by letting the graph run until all the frames have been captured. The graph is an object contained in the `globalGraph4` class and is discussed together with it.

Once the frames have been captured and the graph has been stopped, the recorded sequence is stored in an uncompressed AVI file and the corresponding memory released.

The globalGraph4 class

The `globalGraph4` class sets up a chain of filters to receive an IPTV stream, separate the audio from the video stream, send the video stream to a filter making a copy of the video, and finally render the video. It contains a public method called `BuildGraph(...)`, which is used to initialize the class, as well as a few other functions used to retrieve information about the video (width, height and duration of the frames) and give direct access to the codec chain to the parent class.

The BuildGraph method

In the `DirectShow` framework, filters are instantiated as objects in an environment: the graph. A graph contains a chain of filters starting with a source and ending with a renderer. The filter sequence used in `globalGraph4` is shown in Figure 1. The filters are first added to the graph in sequence, then connected together.

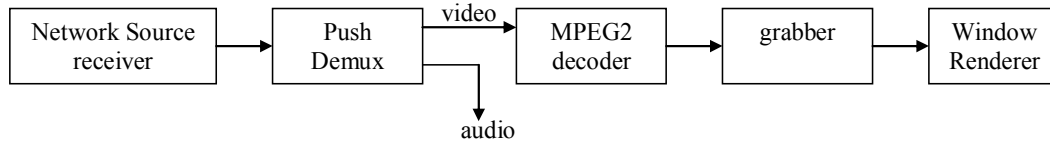


Figure 1: Example of codec sequence used to receive and save a video sequence.

The network source receiver, the push demultiplexer (“PushDemux”) and the MPEG2 decoder are not installed by default on a Windows system. The specific filters used in this class are codecs added to the system when installing the Elecard MPEG media player version 4.5 (www.elecard.com). It should be noted that each codec contains an internal ID code called its CLSID. A CLSID is filter-specific, so two codecs implementing the same functionality will have different CLSIDs if they are not binary identical. As filters are instantiated using their CLSID, they are not exchangeable without modifying and recompiling IPTVInterface.

The Network Source filter listens to the specific IP-address and port, retrieves the network stream and outputs the transport stream to the next filter. The PushDemux removes the transport stream encapsulation and outputs each elementary it contains as a separate output. There are usually one audio and one video stream, as shown in Figure 1. The video stream is sent to the MPEG2 decoder, which decodes it and outputs an uncompressed video stream. Since we are only interested in video, the audio stream is not connected to any input and is lost.

The two remaining filters are standard in the DirectShow environment. The video stream is sent to the grabber filter, a filter with a hook to call a user-implemented function: it is an easy way to implement a filter for simple filters. The grabber filter is hooked to an instance of the `SampleGrabberCallback` class, which makes an in-memory copy of all the frames traversing the filter. The video stream is then sent to the renderer, which displays in a window the video sequence being recorded.

Using another set of filters

Replacing the filters with another set is desirable in particular to replace a codec with one providing another implementation of the same functionality (hence with a different CLSID), or to add support for a new compression format such as MPEG4.

The first thing to do is identify which filters are to be used. An easy way is to start graphEdit (a program provided with Microsoft’s DirectX Software Development Kit), insert the filters (menu Graph => Insert Filters and browse for DirectShow filters in the dialog window) and build a chain starting from a source filter and ending with a renderer.

When the filters are identified, each filter CLSID needs to be retrieved: reopen the dialog window used to insert filters, and select a filter you are interested in. In the lower part of the dialog window appears its “filter moniker”, as shown in Figure 2. The second string between curly braces is the CLSID. In our example (the video renderer), it looks like this in the dialog window: {70E102B0-5556-11CE-97C0-00AA0055595A}. It should be rearranged like this: {0x70e102b0, 0x5556, 0x11ce, 0x97, 0xc0, 0x00, 0xaa, 0x00, 0x55, 0x59, 0x5a} and added to the list of CLSIDs in the file `filter_clsid.h` using a descriptive name. Repeat for each filter that you want to use inside IPTVInterface.

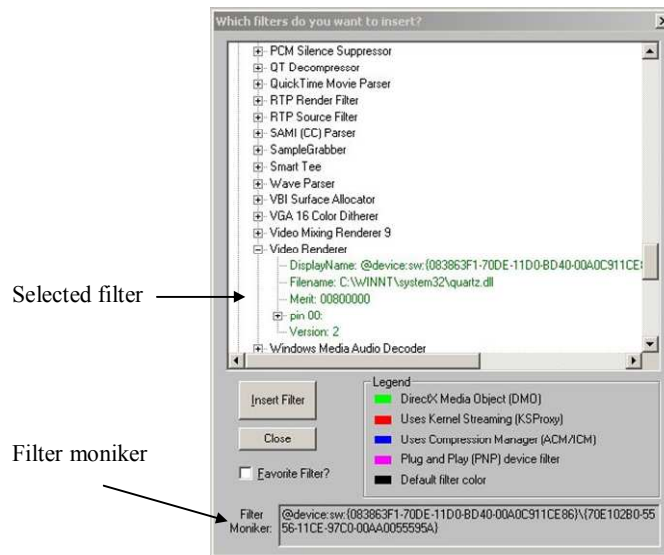


Figure 2: Picture of the dialog window used to insert filters in graphEdit, showing where to find the CLSID of a filter.

Finally, the chain of filters inside the graph has to be modified to use the new filters. This is implemented in method `BuildGraph(...)` of class `globalGraph4`.

Adding support for other filters

Modifying the class `globalGraph4` to use another set of filters is not very flexible: the source code has to be modified and recompiled each time. A better approach would be to create a `genericGlobalGraph` class which implements everything except the `BuildGraph(...)` methods, and have each specific filter chain implemented in its own class derived from `genericGlobalGraph`. This would allow choosing the filter chain to use dynamically, for example from a command line parameter within Matlab.

The `SampleGrabberCallback` class

The `sampleGrabberCallback` class is designed to work with the `SampleGrabber` filter class. An instance of the `sampleGrabberCallback` class, when registered with an instance of the filter class, is called each time the filter receives a new video frame. The `sampleGrabberCallback` class copies that frame into a memory segment. Once the number of copied frames has reached the requested value, the instance sets a flag to let the controlling thread of execution know that the recording is finished (the graph containing the filter chain is executed in another thread than the main program). The instance then writes the recorded sequence to an uncompressed AVI file and releases the memory buffer.

Overview of the execution flow of an instance

The `sampleGrabberCallback` class is derived from class `ISampleGrabberCB`, instances of which can be registered to instances of the `SampleGrabber` filter class.

In a typical use of the class, the execution flow of an instance is composed of three main steps:

- A class instance is created and registered to an instance of the `SampleGrabber` filter. The `sampleGrabberCallback` instance is then initialized: it preallocates a memory buffer to be able to later store the video frames, using the `memManager` class; and it prepares the recording of the video sequence into an AVI file using the `aviWriter` class.

- The instance is used during video playback: it copies in real-time the video frames to its memory buffer, one frame at each call of its `BufferCB(...)` method by the `SampleGrabber` instance.
- After recording, the instance writes the video sequence to an AVI file using the `aviWriter` instance.

Use in the `IPTVInterface` class

An instance of the class is created as a member of the `IPTVInterface` class, inside `IPTVInterface::Init(...)`. A reference to this instance is provided to the `SampleGrabber` filter in `globalGraph4::BuildGraph(...)`. Outside of the `SampleGrabber` filter, calls to the instance are mainly performed in `IPTVInterface::RecordSequence_globalGraph(...)` where the instance is initialized.

The class has two groups of methods: one group meant to be called by the `SampleGrabber` class and following the interface of the `ISampleGrabberCB` class (from which `sampleGrabberCallback` derives); and one used by the classes of the `IPTVInterface` program and used to control the class.

The methods following the interface of `ISampleGrabberCB` are `BufferCB(...)`, `SampleCB(...)`, `QueryInterface(...)`, `AddRef(...)` and `Release(...)`. Of these, `BufferCB(...)` is called by the `SampleGrabber` instance for each new frame and the real-time recording of frames is implemented in the method. `Release(...)` contains some clean-up code. The other methods use default implementations found in sample code.

The methods used to configure the class are `SetNbOfFrames(...)`, `Reset(...)`, `GetCount(...)`, `IsDone(...)`, `CloseAvi(...)`, `PreAllocateMemory(...)`, `SetFileRecording(...)` and `Dump_to_disk(...)`.

The `BufferCB` method

This method is called by the `SampleGrabber` instance every time a new frame is available. The method copies the frame to the buffer managed by the `memManager` instance.

The `SetFileRecording` method

This method prepares the recording of the video sequence to an AVI file using the `aviWriter` class. Giving the NULL pointer as the string for the file name indicates that no recording to file is requested.

The `PreAllocateMemory` method

This method is called with the frame size and number of frames to record as parameters. It allocates a memory buffer large enough to store all the frames as one chunk using the `memManager` class. If the memory allocation fails, it tries using three smaller chunks instead.

The `SetNbOfFrames` method

This method sets the number of frames to record, and allocates an array of pointers to the video frames. This functionality is not in the `PreAllocateMemory(...)` method for historical reasons, but it should be moved there.

The class keeps track of the recorded frame by storing one pointer to each of them. This is because the class originally allocated the memory itself using an `malloc(...)` for each frame on the fly. This proved to be inefficient: the operative system does not guarantee how long it may take to allocate memory and it sometimes took longer than the time slice available between two consecutive frames, causing the next frame to be dropped.

Now that the memory buffer is managed by the `memManager` class, an index should be enough to handle the recorded frames.

The Dump_to_disk method

This method goes through the list of recorded frames. It writes each of them in order to the AVI file and releases the corresponding memory.

The CloseAvi method

This method closes the AVI file and deletes the `aviWriter` instance.

The IsDone method

This method returns whether or not the requested number of frames has been recorded. It is called by the main execution thread, as the recording happens in another thread.

The GetCount method

This method returns the number of recorded frames so far.

The Reset method

This method cleans up the memory allocations, so the class is ready to be initialized to record a new video sequence.

The memManager class

The `memManager` class was written to decouple the memory allocation for the frames from the frame recording, allowing allocation, reallocation and error handling to be handled in a separate module.

The class contains only four public methods, which work by pair:

- Two are used to initialize and clean up the class: they handle the allocation and deallocation of the global chunk of memory used by the class.
- Two handle frame use and do only book-keeping and no memory allocation, to avoid having to wait for the operating system (memory allocation can be quite slow, depending on the system load).

The PreAllocate method

This method receives a number of frames and a frame size as parameters, and allocates a memory buffer of the corresponding size (member `m_memory`). It also allocates an allocation table: it is an array of indices (member `m_allocationTable`), which is used to keep track of which frames have been used or not.

`PreAllocate(...)` can also be called with a parameter telling it that a secondary `memManager` instance should be used. Handling of the secondary instance is then delegated to method `AllocateNextBuffer(...)`. This is useful when allocating the buffer in one chunk has failed due to memory fragmentation: a chain of `memManager` instances may still be able to get hold of the needed amount of memory.

The decision to divide the memory allocation to several chunks is currently taken by the caller of `PreAllocate(...)` (usually the owner of the `memManager` instance), mostly for historical reasons. The code should probably be moved to `PreAllocate(...)`.

The Deallocate method

This method releases the memory allocated by `PreAllocate(...)` and `AllocateNextBuffer(...)`.

The AllocateFrame method

When a new memory frame is requested, the method looks in the allocation table for a free frame. It looks at the indices starting from the index stored in member `m_i`. `m_i` is maintained so that all frames with a smaller index are guaranteed to be allocated. Once a free frame is found, the corresponding value in the allocation table is changed from 0 to 1, `m_i` is updated and the start address of the frame in the memory buffer is calculated and returned.

The FreeFrame method

This method is provided with a pointer to the frame to free. Since the class does not keep a list of all frame pointers, the index of the frame is calculated back from the pointer address. The allocation table is then updated

Proposed design improvements

With the current design, the class calling the `memManager` instance, `sampleGrabberCallback`, has to keep a pointer to each of the frames so that the content can be retrieved and the frames deallocated. This is due to historical reasons, as `sampleGrabberCallback` used to handle itself the memory allocation. There are several drawbacks to this design:

- The caller is provided a direct access to the memory buffer of the `memManager` instance: no checks for out-of-bound reads or writes are done.
- The caller has to allocate and deallocate an array of frame pointers.

An alternative would be:

- To let `AllocateFrame(...)` take as a parameter a pointer to the frame to copy, and perform the copy inside the method.
- To let `FreeFrame(...)` take a frame index as a parameter.
- To add a `RetrievePointer(...)` method so the frames can be retrieved and further processed.

This would allow the indices used in `sampleGrabberCallback` to manage the array of frame pointers to be used directly with `memManager` and remove the only C-style memory allocation from `sampleGrabberCallback`.

The aviWriter class

The `aviWriter` class was written to create an uncompressed AVI file in a simple way. It uses the legacy `VideoForWindows` framework of the MS Windows platform.

A new instance is initialized by successively calling its `SetAviName(...)`, `SetImageSize(...)`, `SetFrameLength(...)` and `InitStream(...)` methods. After initialisation, a new frame is added by a call to `AddFrame(...)`, and the AVI file is closed by a call to `CloseAvi(...)`.

The Initialisation methods

The `InitClass(...)` method is called by the class creators. It initialises the member variables to some sane values. The `SetAviName(...)` method allocates a string and copies the file name of the AVI file in it. The `SetImageSize(...)` and `SetFrameLength(...)` method set their respective member variables accordingly.

The `InitStream(...)` method initialises the AVI header using the provided data about the sequence to record, specifying an uncompressed, 24-bit RGB video stream. It then opens an AVI file and creates a new video stream inside it.

The AddFrame and CloseAvi Methods

The `AddFrame(...)` method sequentially adds frames to the video stream of the AVI file, one frame per function call, until the number of frames declared in the header has been reached. The `CloseAvi(...)` method unconditionally closes the AVI file.

The iptvdebug functions

These are a handful of C-style functions used to write debug messages to a log file. The functions are active only if `_iptvdebug_`, which is declared in `sjdebug.cpp`, is set to `true`.

`debug_openfile(...)` and `debug_closefile(...)` are used to create and close a log file, respectively.

`debug_writestatus(...)` writes a string to the log file, preceded by the number of tabulations specified in the second argument (that is the variable called `debuglevel` throughout the program).

`debug_test_hr(...)` is like `debug_writestatus(...)`, but it takes an additional input parameter of type `HRESULT`. This type is used by a number of Microsoft functions to return error codes. In addition to writing the provided string to the log file, `debug_test_hr(...)` writes there the meaning of the error code.

The directshow_tools functions

These are a few C-style functions used to retrieve the input and output pins of filter classes. They were adapted from code samples provided with [2], who himself acknowledges the code as coming from the DirectX 9.0 SDK.

`GetPin(...)` retrieves the pins unconditionally and `GetPinByName(...)` retrieves them by name. Indeed, filters providing several output pins do not always provide them in the same order, and it is important to distinguish between for example an audio and a video output. `stringCompare(...)` is used to compare the names of pins. These functions are only used within the `directshow_tools.cpp` file.

`GetPinSafe(...)` and `GetPinSafeByName(...)` are accessible outside of `directshow_tools.cpp`. `GetPinSafe(...)` uses `GetPin(...)` to provide filter pins, but controls the validity of the ping. If the pin returned by the filter is invalid, the function sleeps for a while and tries again. This gives time to newly-connected filters to be initialized and start providing output pins. `GetPinSafeByName(...)` does the same thing for named pins, using `GetPinByName(...)`.

Calling IPTVInterface from Matlab

The following Matlab script illustrates how to call `IPTVInterface`:

```
% Define commands
IPTV_INIT=0 ;
IPTV_RECORD=1;
IPTV_RELEASE=3;

% Initialise IPTVInterface
% IP address and port number of the IPTV channel
iptv_address='235.1.1.1:1234';
ret = IPTVInterface(IPTV_INIT, iptv_address);
if (ret == 0)
    disp('Initialising IPTVInterface succeeded.')
else
```

```
% The ret value is the error code that can be found in iptverrors.h
disp('Initialising IPTVInterface failed.') ;
disp('Look in the log file for the reason') ;
end

% Record a sequence
% Name of the AVI file to save the sequence to
Name1='name_1.avi'
% Number of frames to record
Nf1=400 ;
ret = IPTVInterface(IPTV_RECORD, Nf1, Name1);
if (ret == 0)
    disp('recording sequence 1 succeeded.')
else
    % The ret value is the error code that can be found in iptverrors.h
    disp('Recording sequence 1 failed.') ;
    disp('Look in the log file for the reason') ;
end

% Record a second sequence
% Name of the AVI file to save the sequence to
Name2='name_2.avi'
% Number of frames to record
Nf1=500 ;
ret = IPTVInterface(IPTV_RECORD, Nf2, Name2);
if (ret == 0)
    disp('recording sequence 2 succeeded.')
else
    % The ret value is the error code that can be found in iptverrors.h
    disp('Recording sequence 2 failed.') ;
    disp('Look in the log file for the reason') ;
end

% Release the IPTVInterface resources
% The return code is always 0
IPTVInterface(IPTV_RELEASE);
```

IPTVcpp

The console-based program

IPTVcpp is a MS Windows program with the same functionality as IPTVInterface, but compiled as a standalone executable instead of a Matlab library. The Matlab interface file, mx_main.cpp, is replaced by a file called IPTVcpp.cpp and containing a standard C/C++ main(...) function. The main(...) function parses the command line arguments, and creates and initialises an IPTVInterface instance

and records the number of requested sequences. It then deletes the `IPTVInterface` instance, thereby releasing all the previously-allocated library resources.

Calling IPTVcpp

With `IPTVcpp`, all the arguments have to be provided with one call. The executable expects the following arguments:

```
C:\user\IPTVInterface\IPTVcpp.exe 225.1.1.1:1234 400 10
```

The first argument is composed of the IP address of the channel to record (255.1.1.1) followed by a colon and the channel port number (1234). The second argument is the number of frames to record for each sequence (400). The third argument is the number of video sequences to record (10). The sequences are saved in files with names `testavi_1.avi`, `testavi_2.avi`, ...

Compiling and using the code

Compiling the library and the standalone executable

The code source contains VC++ project files (version 7.1, also known as .Net, version 2003) for both the library and the console program. To build the Matlab library, Matlab has to be installed. At the time of this writing, we use Matlab version 7.1.0.246 (R14) Service Pack 3, but others should work too (the software was mostly developed using Matlab version 6.5).

Maturity and stability of the code

The Matlab library built with the Debug configuration has been extensively tested. It has been used in our lab with `iVQM` for periods of 1 to 2 weeks at a time. It should be completely stable, but differences in recording between two systems running `iVQM` and fed with the same channel were sometimes observed (in approximately 1% of the recorded sequences). The cause is believed to be frame drops. A faster system equipped with more memory should solve the problem.

The standalone based executable built with the Debug configuration has been moderately tested. However, the code that is not shared with the Matlab library is contained in a single file. It is meant as a way to demonstrate the `IPTVInterface` class without having to use Matlab.

The Release configurations of the Matlab library and of the standalone program are known to generate code, but neither the library nor the executable built in this way have been tested.

Conclusion

Acknowledgements

The authors wish to thank the US National Telecommunications and Information administration (NTIA) and in particular Margaret Pinson for the support and encouragements, and for integrating and testing `IPTVInterface` with `iVQM`.

This work was financed by the MUSE EU project (IST-026442) and VINNOVA (The Swedish Governmental Agency for Innovation Systems)

Future work

Memory management

The `memManager` class should be modified to do the frame copying itself, instead of exporting pointers to its memory buffer, as described in paragraph [Proposed design improvements](#) of the section describing

the class (page 10). The calls to the `memManager` instance within the `SampleGrabberCallback` class should be updated accordingly.

The memory handling within `SampleGrabberCallback` class should be modified to use only an index to handle frames stored inside the `memManager` instance: the array of pointers to image frames should be removed.

Handling of codecs and filter chains

The code does not currently allow choosing which set of filters to use to record a video stream. Support for choosing the filters at run-time should be added, as described in paragraph [Adding support for other filters](#) of the section describing the `globalGraph4` class (page 7), using a virtual base class, and a derived class for each filter chain. Filter chains using the most commonly-found codecs should also be added.

Appendix: the GNU General Public License v2.0

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
- b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
- c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and

so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

References

- 1 <http://www.its.bldrdoc.gov/n3/video/index.php>
- 2 Mark D. Pesce, "Programming Microsoft DirectShow for digital video and television", Microsoft Press, ISBN 0-7356-1821-6 (2003).