Intel® Video Metadata Framework SDK

Developer's Guide – version 3.0

Content

Introduction

Intel® Video Metadata Framework SDK (Intel® VMF SDK) provides functionality for creating, editing, and embedding metadata into video files.

This guide explains how to install the library package, describes the samples and demo application, and shows how to use the SDK in user applications.

For an overview of the library's functionality and how to develop applications, please review the specification document (VMF-Specification.pdf).

For detailed information of the library functions and their parameters, please use the Doxygen documentation shipped with the library (API Specification).

Any software source code reprinted in this document is furnished under a software license (VMF-EULA.docx) and may only be used or copied in accordance with the terms of that license.

Installation

The best way of getting VMF binaries is to build it from the sources downloaded from its GitHub page (https://github.com/01org/vmf).

Anyway, to install the Intel VMF SDK from the pre-built installation package, perform these steps:

1. Run the installer package (Intel-VMF-SDK-3.0.msi) and follow the prompts given by the installation wizard
    or just unpack the provided ZIP archive (Intel-VMF-SDK-3.0.zip).

2. Add the `VMF_DIR` environment variable with the path to the Intel VMF installation to allow 'find_package(VMF)' CMake command to automatically find the Intel VMF SDK
    or just define the 'VMF_DIR' CMake variable to that path each time you need CMake to find the Intel VMF SDK include and lib-s files.

3. Add to the `PATH` environment variable the value
    `%VMF_DIR%\<your_arch>\<your_vc_version>\bin`
    before running your application if you use dynamic VMF lib-s (VMF_STATIC=OFF).

Project structure

Intel VMF installer creates the following folders:

- android_libs – precompiled Android* libraries for 3 architectures:

    - armeabi

    - armeabi-v7a

    - x86

- cmake – CMake script necessary to compile user-dependent projects

- data – files used by demos and test applications

- docs – all project documentation

    - APPI-Spec – Doxygen-generated API documentation

- include – C++ project header files

- samples – sample applications

- x64 – files for Win-x64 targets

    - vc12 – files for MSVS 2013 projects for Win-x64 targets

        - bin – dynamic lib-s (DLL-s)

        - lib – (import lib-s for DLL-s)

        - static-lib (static lib-s)

- x86

    - vc12

Building the samples

The Intel VMF binary package contains a set of sample applications for illustrating the basic concepts of Intel VMF library usage. To build the samples, follow the instructions below:

- Run CMake GUI and select the `<vmf_install_path>\samples` folder as the "source code" path, also select the build folder (where you want to generate the MSVS project).

- If you plan to build QT samples, in "Ungrouped entries" check the BUILD_QT_SAMPLES box.

- Press the Configure button.

- In the pop-up dialog select the MSVS version and bitness level (32/64CPU architecture (i.e. Win64) and click Finish.
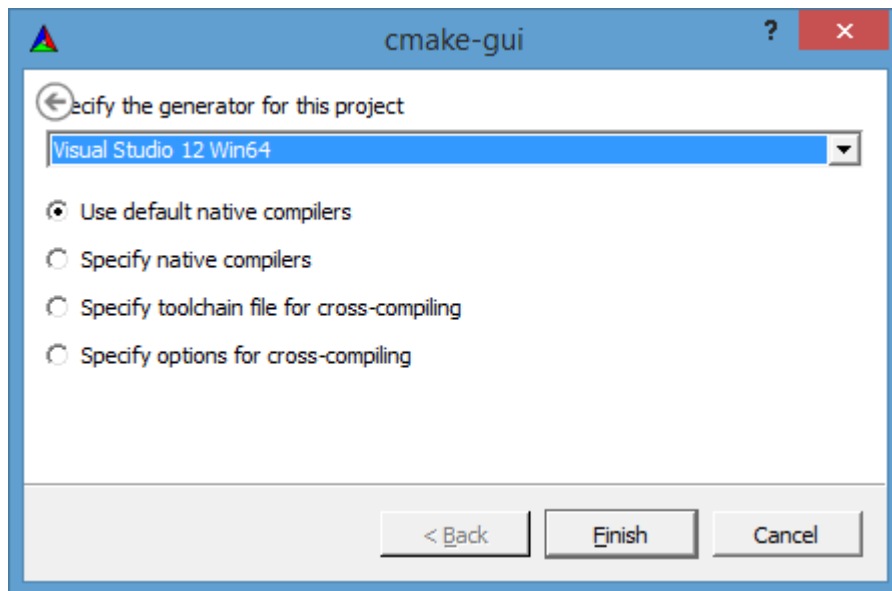
- Set the correct value of the OpenCV_DIR variable (a path to OpenCV* build directory) and press Configure.

- If Qt5 is not found, add the QT5_DIR variable of PATH type and put a path to your Qt5 libs directory, then press Configure.
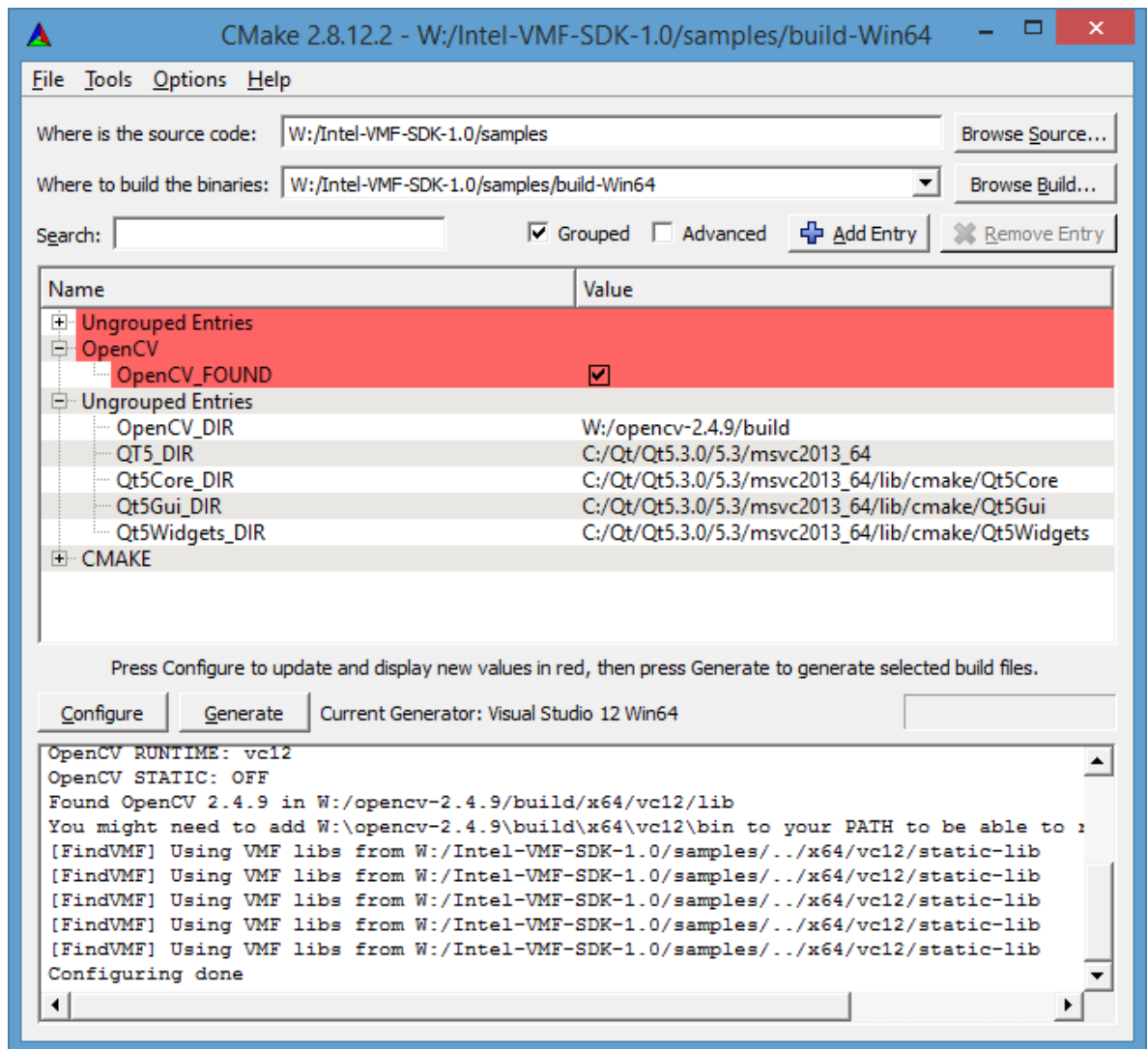
- The prebuilt Intel VMF libs should be found automatically, but if not, add the VMF_DIR variable of PATH type and set its value to the top-level Intel VMF SDK path and press Configure.

- Press the Generate button.

- In MSVS open the samples.sln file in the build folder. Select BUILD -> Build Solution menu item to build the solution.
  NOTE: When building the Debug configuration, you may get many warnings regarding missing PDB files for the Intel VMF lib. These are not critical and can be ignored.
  NOTE: You may get an error message that a post-build step of one or more project(s) has failed. It may happen in parallel builds when more than one concurrent build tries to copy the same video file (BlueSquare.avi) to the same location (binary directory). Just ignore this error and build again–the error should disappear.

Creating applications using Intel® VMF SDK

This section contains three subsections, each dealing with different development platforms: Windows*, Android, and iOS*.

Creating Windows applications using Intel® VMF SDK

On the Windows platform, you have a choice of either Visual Studio* or CMake.

Configuring applications with Visual Studio GUI*

To use Intel VMF SDK in your Visual Studio project, you should manually set up include and library directories and link the appropriate Intel VMF library to your project. This procedure consists of the following steps:

- Open Properties -> Configuration Properties -> Linker-> General -> Additional Library Directories list of your project and add <VMF Dir>\<your arch>\<your vc ver>\lib to use the dynamic Intel VMF library or <VMF Dir>\<your arch>\<your vc ver>\static-lib to use the static one.

- Add vmfd.lib (debug version) or vmf.lib (release version) to the Properties -> Configuration Properties -> Linker -> Input -> Additional Dependencies list of your project.

- Add <VMF Dir>\include to the list of directories with headers (Properties -> Configuration Properties -> C/C++ -> General -> Additional Include Directories).

When using dynamic Intel VMF lib, please copy vmfd.dll (debug version) or vmf.dll (release version) to the folder with your executable or setup %PATH% accordingly.

Configuring applications with CMake

To generate MSVS solutions with CMake, the steps are:

- Install or build the Intel VMF Library (see the sections above for more details)

- Add <VMF Dir>/cmake to CMAKE_MODULE_PATH in your project
  or copy <VMF Dir>/cmake/FindVMF.cmake under any directory into your
  CMAKE_MODULE_PATH (e.g., where the CMakeLists.txt is located)
  or just provide the VMF_DIR variable containing the path to Intel VMF to CMake
  or define the VMF_DIR environment variable containing the path to Intel VMF SDK.

Then you will be able to use the following variables:

- find_package(VMF) for searching

- VMF_LIB_DIR variable adding the Intel VMF library directory by link_directories()

- VMF_LIBS for linking via target_link_libraries() and VMF_INCLUDE_DIR for adding to includes path with include_directories()

  Look in the <VMF Dir>/cmake/FindVMF.cmake file for details.

Creating Android applications using Intel® VMF SDK

This section explains the basic principles of creating Android applications that use the Intel® VMF SDK. The steps of how to rework the sample distributed with Android NDK are provided. You should have basic Android development skills and have the Android NDK, Android SDK, and the Intel VMF library installed.

C++ level

Let's rework the hello-jni sample from Android NDK (http://developer.android.com/tools/sdk/ndk/index.html). Open this project in your development environment (Eclipse* ADT bundle is preferable).

Create an Application.mk file in the jni directory of your sample and include the following content:

```
APP_STL := gnustl_shared
APP_GNUSTL_FORCE_CPP_FEATURES := exceptions rtti
APP_ABI := x86 armeabi-v7a armeabi
APP_MODULES := vmf hello-jni
APP_PLATFORM := android-9
```

Change non-commented content of Android.mk to:

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := vmf
LOCAL_SRC_FILES :=
<vmf_installation_dir>/android_libs/$(TARGET_ARCH_ABI)/libvmf.so
LOCAL_EXPORT_C_INCLUDES += <vmf_installation_dir>/include
include $(PREBUILT_SHARED_LIBRARY)
include $(CLEAR_VARS)
LOCAL_MODULE := hello-jni
LOCAL_SRC_FILES := hello-jni.cpp
LOCAL_CPPFLAGS += -std=gnu++11
LOCAL_SHARED_LIBRARIES := vmf
include $(BUILD_SHARED_LIBRARY)
```

Rename hello-jni.c to hello-jni.cpp

Change non-commented content of hello-jni.cpp to:

```
#include <string.h>
#include <jni.h>
#include <vmf.hpp>

#ifdef __cplusplus
extern "C" {
#endif

jstring
Java_com_example_hellojni_HelloJni_stringFromJNI( JNIEnv* env, jobject thiz );
#ifdef __cplusplus
}
#endif

jstring
```

```cpp
Java_com_example_hellojni_HelloJni_stringFromJNI( JNIEnv* env, jobject thiz )
{
  std::string message("Test Android message");
  // Part 1
  // Save message to manually created file
  vmf::MetadataStream outStream;
  if(!outStream.open("<your_test_file>", vmf::MetadataStream::ReadWrite))
    return env->NewStringUTF("Couldn't open test file (RW)");
  const std::string SCHEMA_NAME("Test Android Schema");
  std::shared_ptr<vmf::MetadataSchema> schema =
    std::make_shared<vmf::MetadataSchema>(SCHEMA_NAME);
  const std::string METADATA_NAME("Test Android Metadata");
  std::shared_ptr<vmf::MetadataDesc> desc =
    std::make_shared<vmf::MetadataDesc>(METADATA_NAME,
  vmf::Variant::type_string);
  schema->add(desc);
  try {
    outStream.addSchema(schema);
  } catch (...) {
    // do nothing
  }
  std::shared_ptr<vmf::Metadata> metadataForSaving =
    std::make_shared<vmf::Metadata>(desc);
  metadataForSaving->addValue(message);
  outStream.add(metadataForSaving);
  if(!outStream.save())
    return env->NewStringUTF("Couldn't save message to file..");
  outStream.close();
  // Part 2
  // Read message from file
  vmf::MetadataStream inStream;
  if(!inStream.open("<your_test_file>", vmf::MetadataStream::ReadOnly))
    return env->NewStringUTF("Couldn't open test file (RO)");
  inStream.load();
  vmf::MetadataSet testMetadata = inStream.queryByName(METADATA_NAME);
  std::string readMessage = (std::string) testMetadata.at(0)->at(0);
  return env->NewStringUTF(readMessage.c_str());
}
```

The code above adds the text "Test Android message" to a video file, then reopens this video file and displays the text embedded on the previous step.

Add code to load two more libraries (in addition to hello-jni) to the HelloJni.java file:

```java
  /* this is used to load the 'hello-jni' library on application
   * startup. The library has already been unpacked into
   * /data/data/com.example.hellojni/lib/libhello-jni.so at
   * installation time by the package manager.
   */
  static {
      System.loadLibrary("gnustl_shared");
      System.loadLibrary("vmf");
      System.loadLibrary("hello-jni");
  }
```
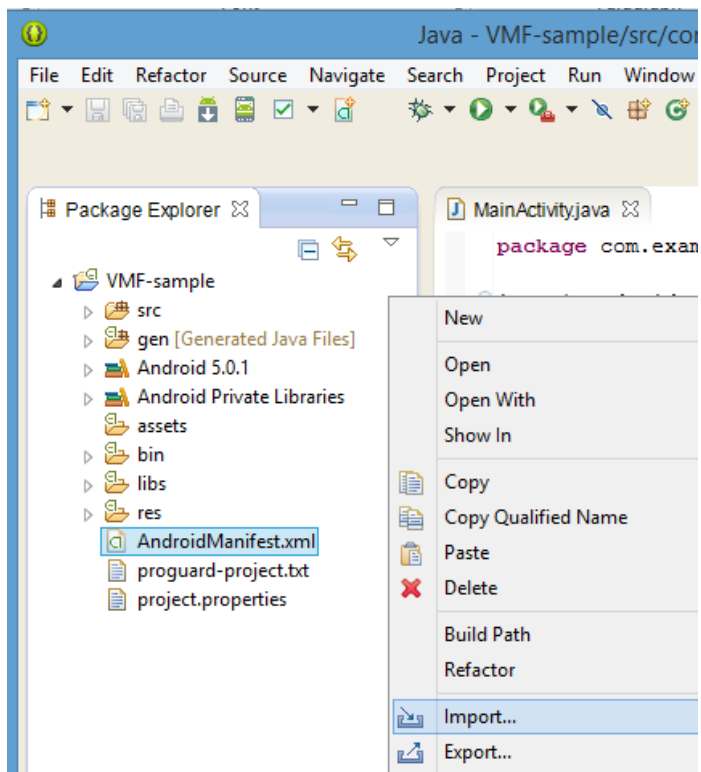
Build your application. Copy the test application file to an Android device and run it. It will create a window with a string "Test Android message" like the screenshot below:
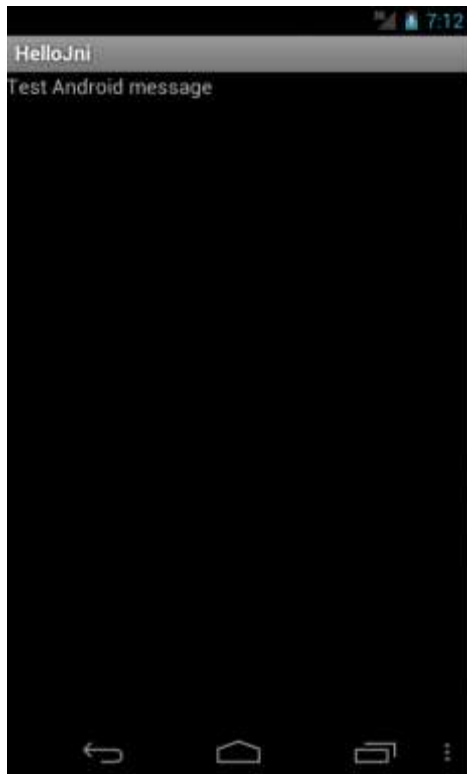
Java level

It's also possible to utilize the VMF Android Library Project for Eclipse+ADT IDE to use the VMF via Java API. To enable VMF Java API for an existing Android application project in Eclipse+ADT just add the VMF Android Library Project to your Eclipse+ADT workspace and set dependency from it on your application project like shown on the screenshots below.

Import

Select

Create new projects from an archive file or directory.

Select an import source:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
- Android
- C/C++
- Git
- Install
- Run/Debug
- Team
- XML

< Back   Next >   Finish   Cancel

Browse For Folder
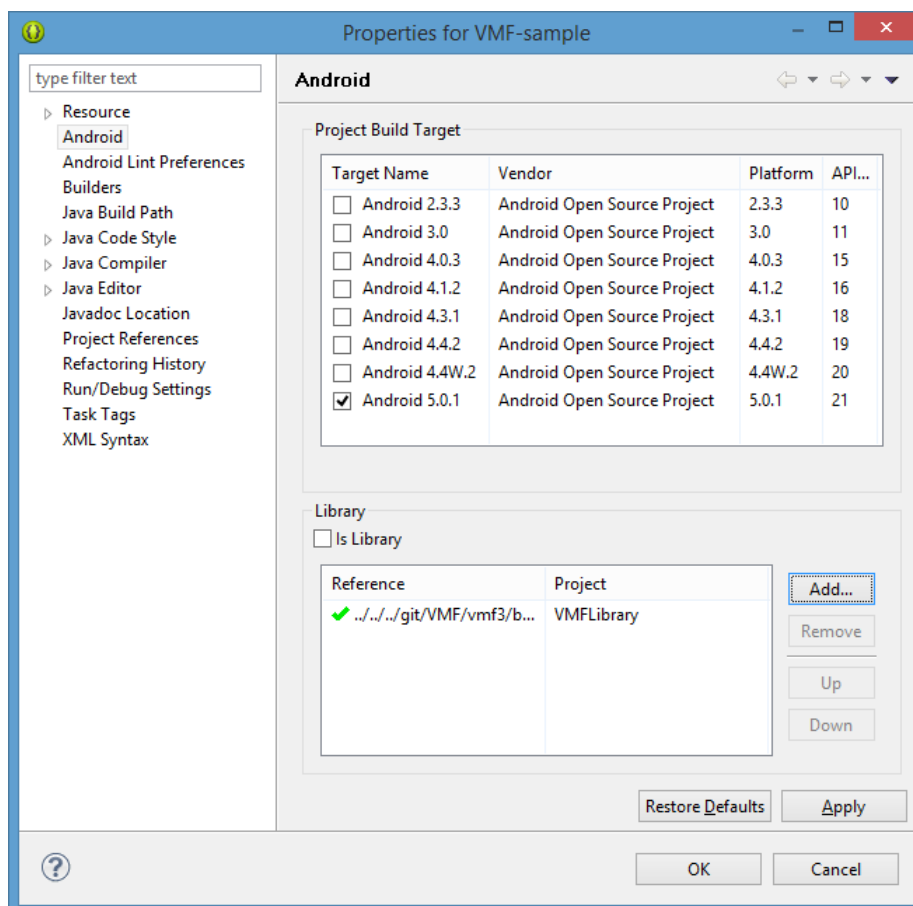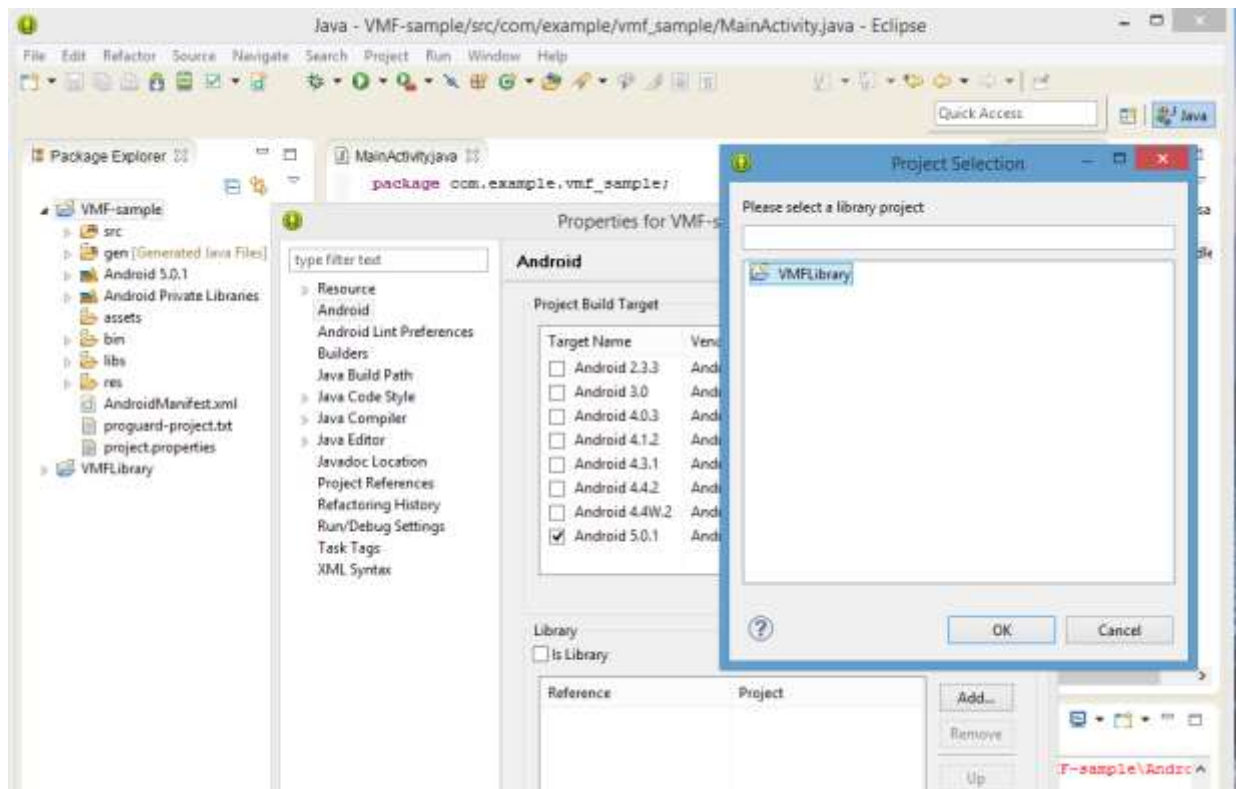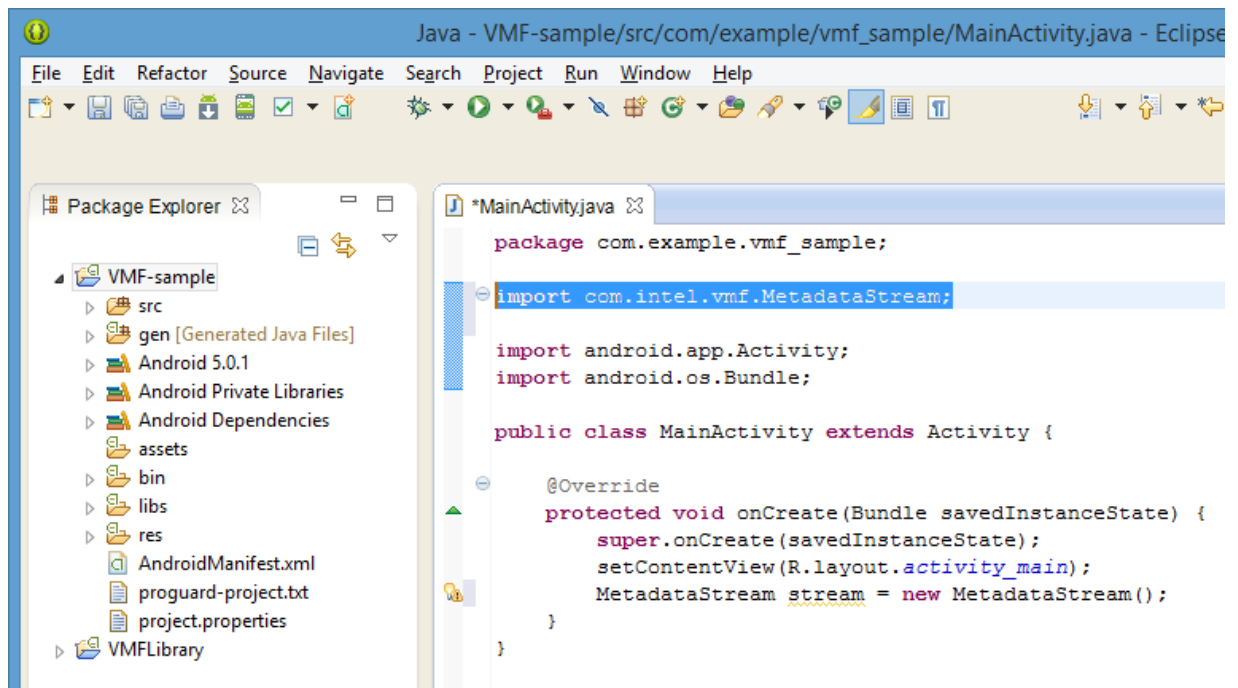
Select root directory of the projects to import

- vmf3
  - 3rdparty
  - build-x86
    - 3rdparty
    - bin
    - CMakeFiles
    - install
      - android-lib
        - .settings
        - jni
        - libs
        - res
        - src
      - lib
      - modules
    - cmake
    - data
    - demo
    - docs
    - modules

Folder: install

Make New Folder   OK   Cancel

Import

Import Projects

Select a directory to search for existing Eclipse projects.

- Select root directory: W:\Dev\git\VMF\vmf3\build-x86\ir   Browse...
- Select archive file: Browse...

Projects:

- ☑ VMFLibrary (W:\Dev\git\VMF\vmf3\build-x86\install\

Select All
Deselect All
Refresh

☐ Copy projects into workspace

Working sets
☐ Add project to working sets

Working sets: Select...

< Back   Next >   Finish   Cancel

File Edit Refactor Source Navigate Search Project Run Window Help

Package Explorer ⊠

▲ 🗁 VMF-sample
  ▷ 🔀 src
  ▷ 🔀 gen [Generated Java Files]
  ▷ 📐 Android 5.0.1
  ▷ 📐 Android Private Libraries
  ▷ 📐 Android Dependencies
    🗁 assets
  ▷ 🗁 bin
  ▷ 🗁 libs
  ▷ 🗁 res
    📄 AndroidManifest.xml
    📄 proguard-project.txt
    📄 project.properties
▷ 🗁 VMFLibrary

*MainActivity.java ⊠

```java
package com.example.vmf_sample;

import com.intel.vmf.MetadataStream;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        MetadataStream stream = new MetadataStream();
    }

}
```

Creating iOS applications using Intel® VMF SDK

Using 'vmf.framework' in an Xcode project*

1. Create/open an Xcode project.
2. The 'vmf.framework' can be added through project properties by navigating to Project -> Build Phases -> Link Binary With Libraries. To open project properties, click the project name in the Project Navigator area.
3. Include 'vmf.framework' in the Xcode project. You need to add a line "`#import <vmf/vmf.hpp>`" to all files that will use 'vmf.framework' functions.
4. Set the value of the "Compile Sources As" property to Objective-C++. The property is available in the project settings and can be accessed by navigating to Project -> Build Settings -> Apple LLVM 5.1 - Language.

Demos Samples

The Intel VMF solution contains a few basic demo applications: demo-author, demo-gps-time, and demo-Unicode. The first demo application shows how to insert and read global metadata embedded in a media file. The application creates a global 'author' metadata with the author's name and inserts it to a video file. It then reopens the file, reads the inserted metadata, and prints the extracted info to the console.

The second demo shows how to insert and retrieve metadata with associated values. This demo application inserts four metadata values in a video file. Each metadata consists of GPS coordinates and associated time values. The application then reads metadata from the file and prints results to the console.

The last demo saves UTF-8 coded text to a file to illustrate dealing with non-English characters.

For instructions on building the demos and the samples, refer to the "Building the samples" section.

Metadata-manipulation

This sample is a simple interactive console application performing three basic metadata manipulation actions: shows metadata structure, dumps metadata items, and removes metadata from the specified video file.

NOTE: You cannot delete schemas or sets–only metadata items (due to Intel VMF API restrictions).

Metadata-read-write

This demo shows how to use metadata descriptions to store data structures such as GPS coordinates with associative time.

Compression

This sample shows how to save metadata using built-in and user-provided compression algorithms and load data back. It's also shown how to handle a situation when the data in the file are compressed using unknown algorithm.

Encryption

This sample shows how to save metadata using built-in and user-provided encryption algorithms and load data back. It's also shown how to handle a situation when the data in the file are encrypted using unknown algorithm.

Metadata-statistics

This sample shows how to use built-in and user-provided statistics operations.

Checksum-utility

This sample shows how to use the checksum API for checking whether a video file was modified or not.

Std-schema

This sample shows how to use predefined STD schema and built-in predefined metadata fields.

Unicode

This sample shows how the Intel VMF SDK can work with Unicode strings.

Photo book

Overview

The photo book application works with media files containing faces. The application can be used for labeling faces (i.e., marking a rectangle around a face). Users can assign names to people labeled on the media file and assign attributes to each such as sex, age, weight, and height. Names, rectangles, and the attributes can be saved to the media file as metadata. If a user opens a media file with pre-created metadata, the application pre-loads the metadata and visualizes it (i.e., shows labeled faces and rectangles).

UI Description

This section describes the user interface of the photo book sample application.

Main window

The figure shows the main window of the photo book application.
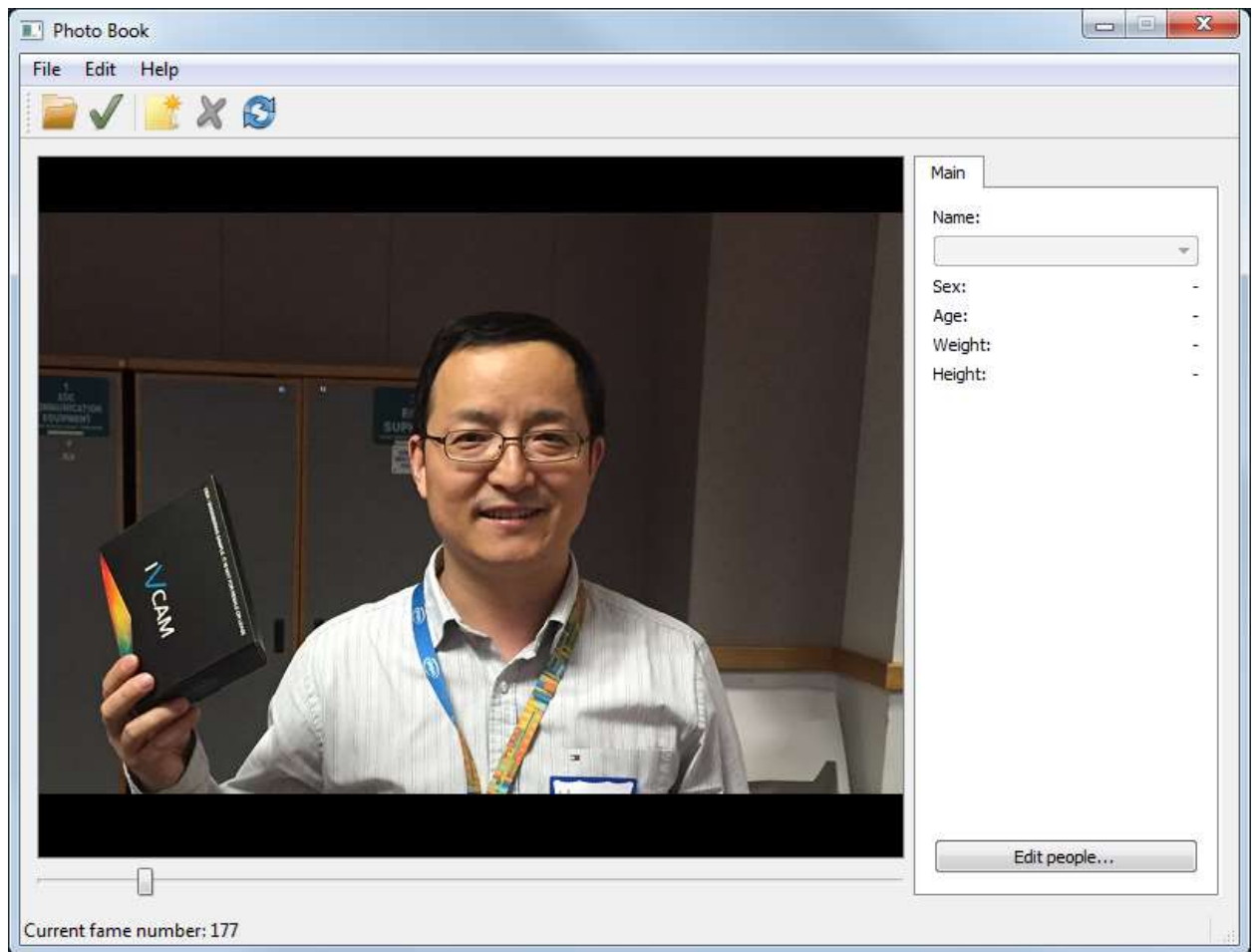
On this window you can open video files, select face regions (manually or automatically), associate regions to selected persons, and save changes as metadata. The central part of the window shows the current frame of the opened video file. The right part contains a tab with information about the associated person.

Opening a file

To open a media file, select the File -> Open... menu item or press the button on the toolbar. A dialog window will display where you can select a media file to open (currently only video files are supported).
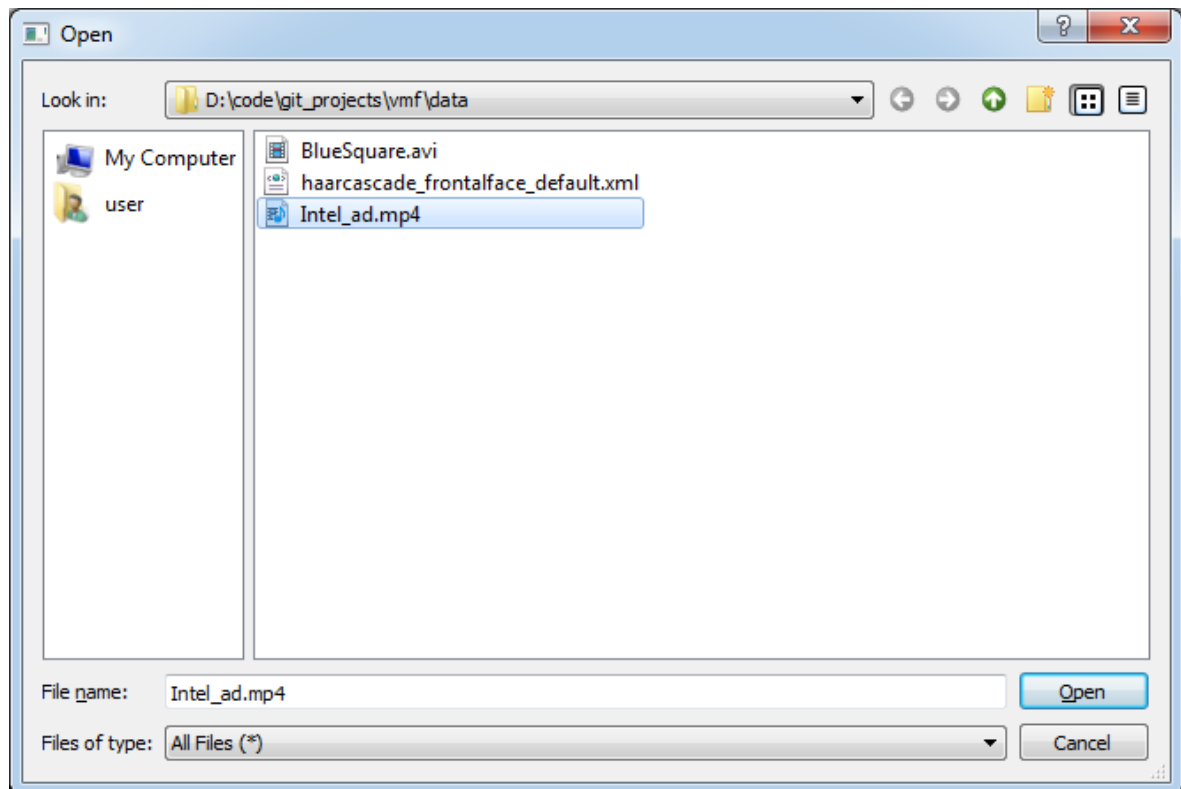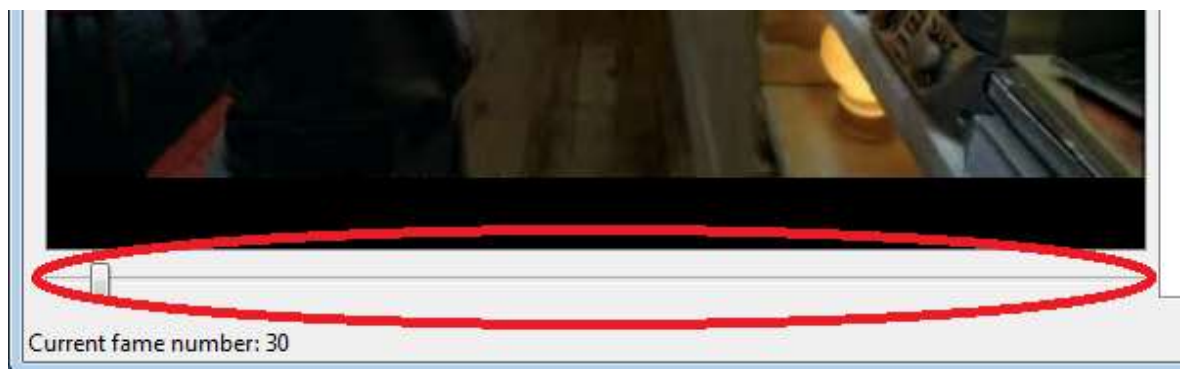
After you press Open, the selected media file will be loaded. If the file already contains compatible metadata, then the metadata will be loaded too.

File navigation

For fast navigation you can move the slider at the bottom part of the main window:

For frame-by-frame navigation you can use the ← and → buttons on your keyboard. In both cases the current frame number is displayed on the status bar.

The application can automatically detect faces on the displayed frame and mark them with green rectangles. Note that auto-detection of human faces works only with previously unshown frames. If you want to force face detection, select the Edit -> Renew markup menu item or press the button on the toolbar.

The next figure shows an example face detection result.
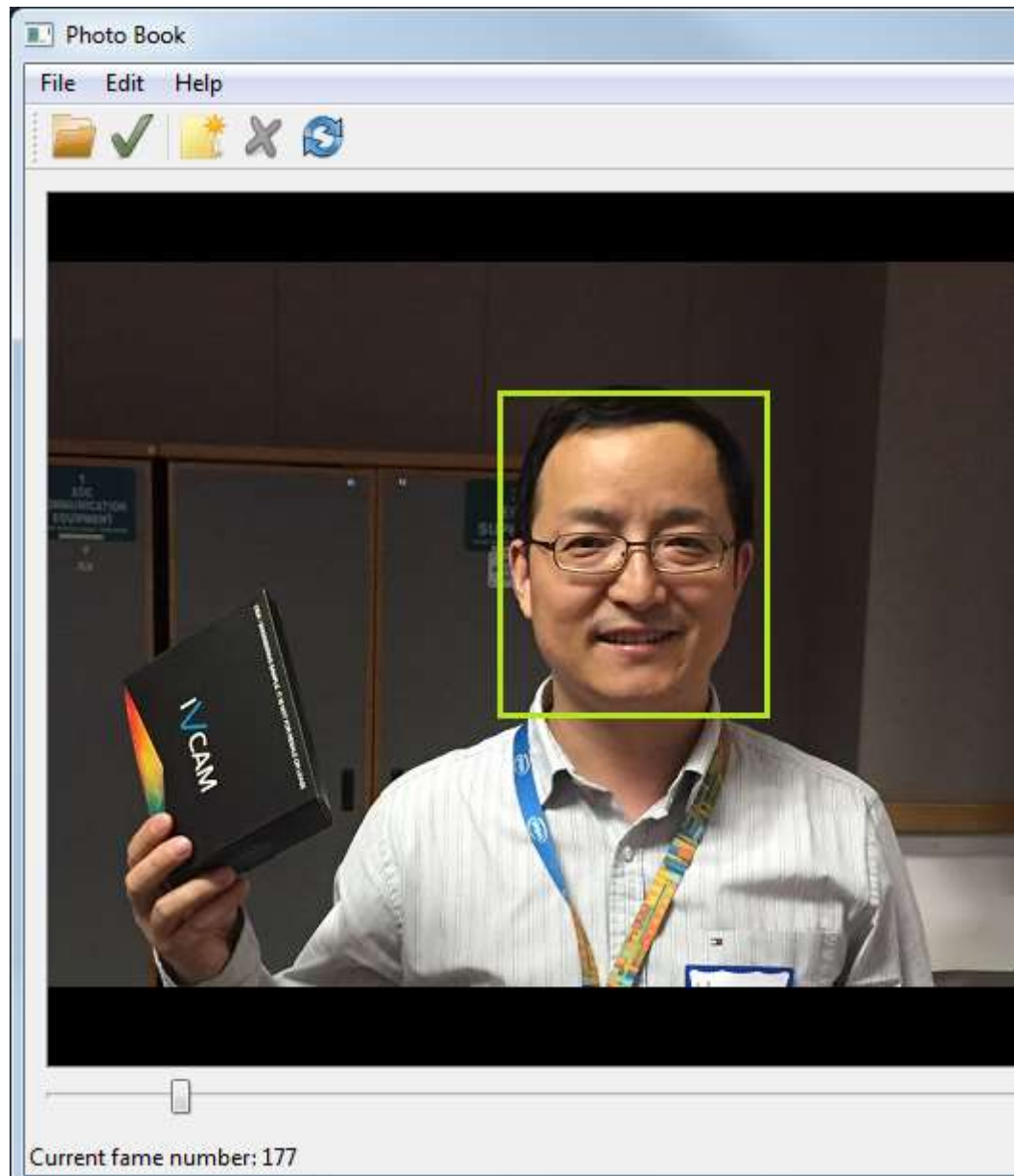
Adding, resizing, and deleting regions

You can add a region on the current frame manually. Just right click on the frame area and select the New region item from the context menu. In this case the new frame region appears near the mouse cursor. Also you can select the Edit -> New region menu item or press the [button] button on the toolbar to add a new region to the left top corner of the frame.

You can resize the regions by dragging the corners using the mouse.

To move a region, put the mouse cursor into the region area, press the left mouse button, and move the region to the new place.

To delete a region, select the region by right clicking on the region area and then selecting the Delete context menu item. Alternatively, you can select the region with a left mouse click and then clicking the Edit -> Delete region menu item or pressing the ✖ button on the toolbar.

Region assignment

The photo book sample application supports two types of regions: associated and unassociated. Unassociated regions do not describe any person on the displayed frame. Regions of this type have the color green.

You can associate unassociated regions to a person. To do this select the region using the mouse. A list with all known people names will be displayed on the right part of the main window. Select the name of the person with whom you want to associate the region:
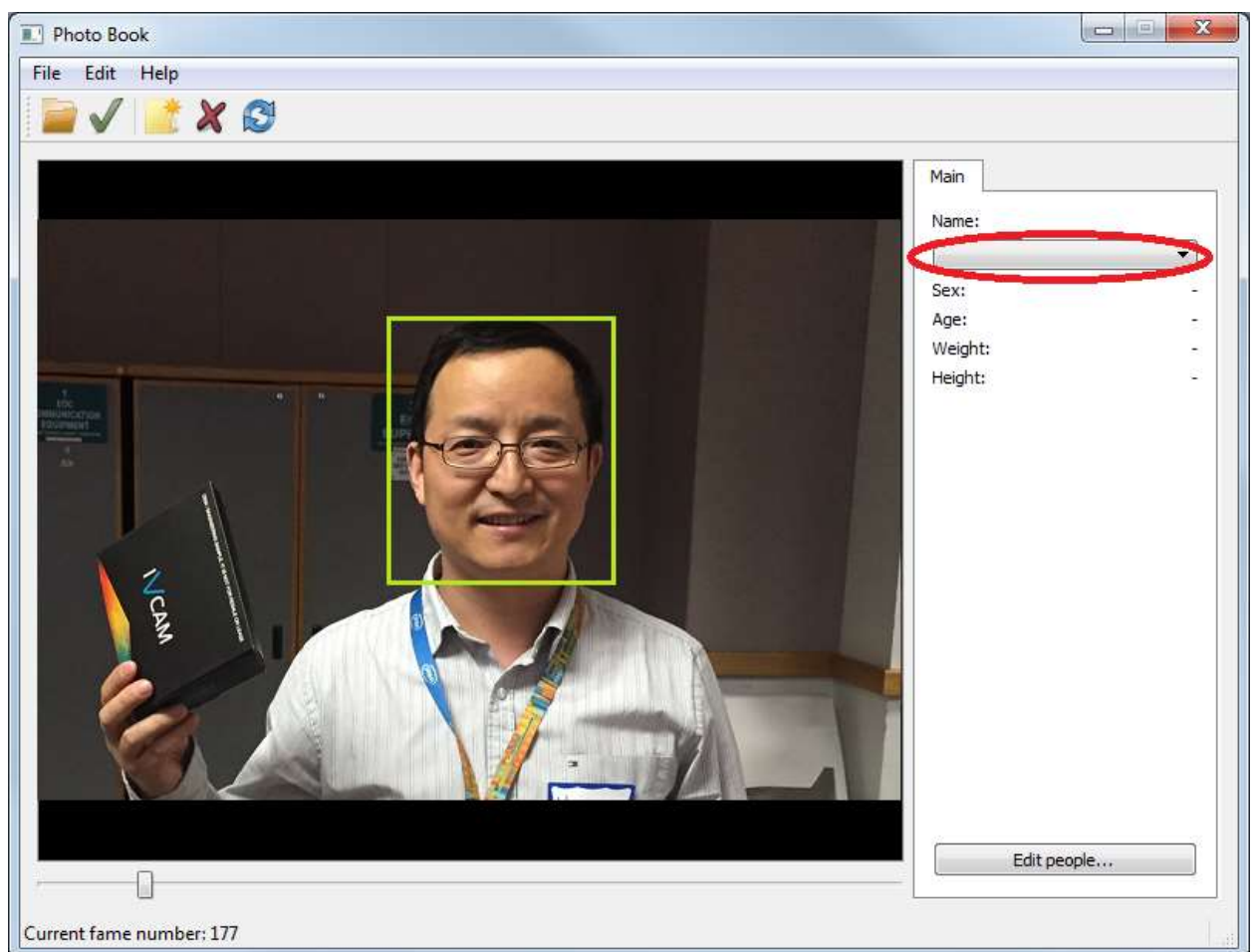
The region will change color to blue as shown in the next figure:
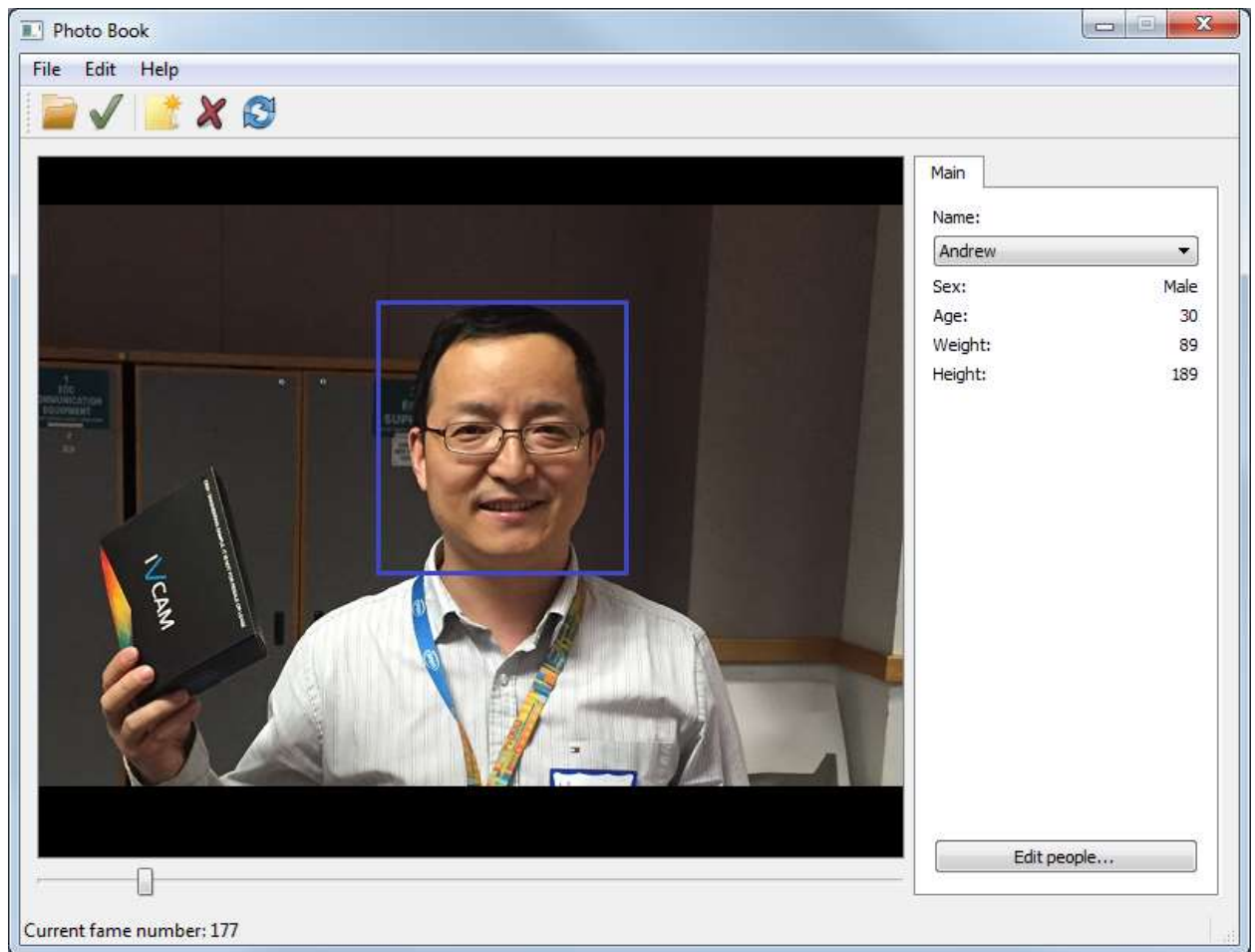
You can change the size and position of associated regions the same way as you can unassociated regions. Also you can clear region association. Right click on the region area and select the Remove association context menu item.

To change region assignment simply select the specified region and change the person's name in the combo box.

Saving metadata

To save all metadata changes to a media file, select the File -> Commit markup menu item or press the

button on the toolbar. After that, all added regions and information about each person will be saved.
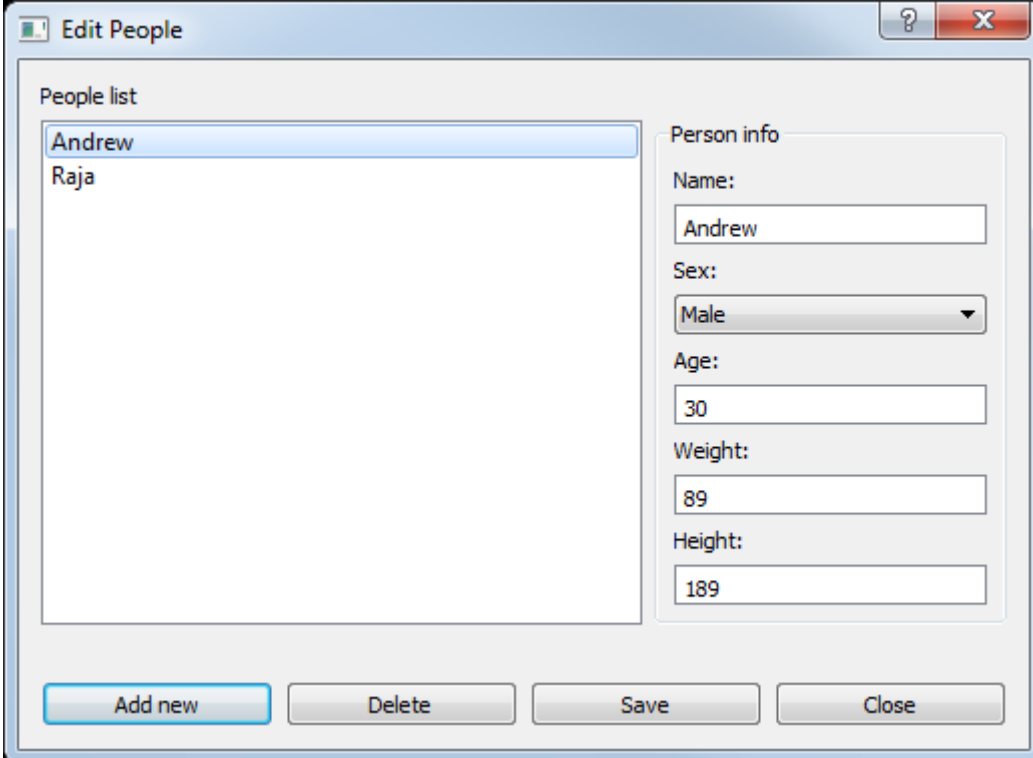
People information tab

The people information tab is located on the right side of the main window. The tab displays information about a person: name, sex, age, weight, and height. These values are automatically updated when you select a region.

To edit people information, press the Edit people button at the bottom of the tab. More information on how to edit people information is provided in the next section.

Editing people information

The next figure shows the UI for editing people information. You can add, change, and remove people and their attributes.

To add a new person, click the Add new button. Then enter information about the new person such as name, sex, age, etc.

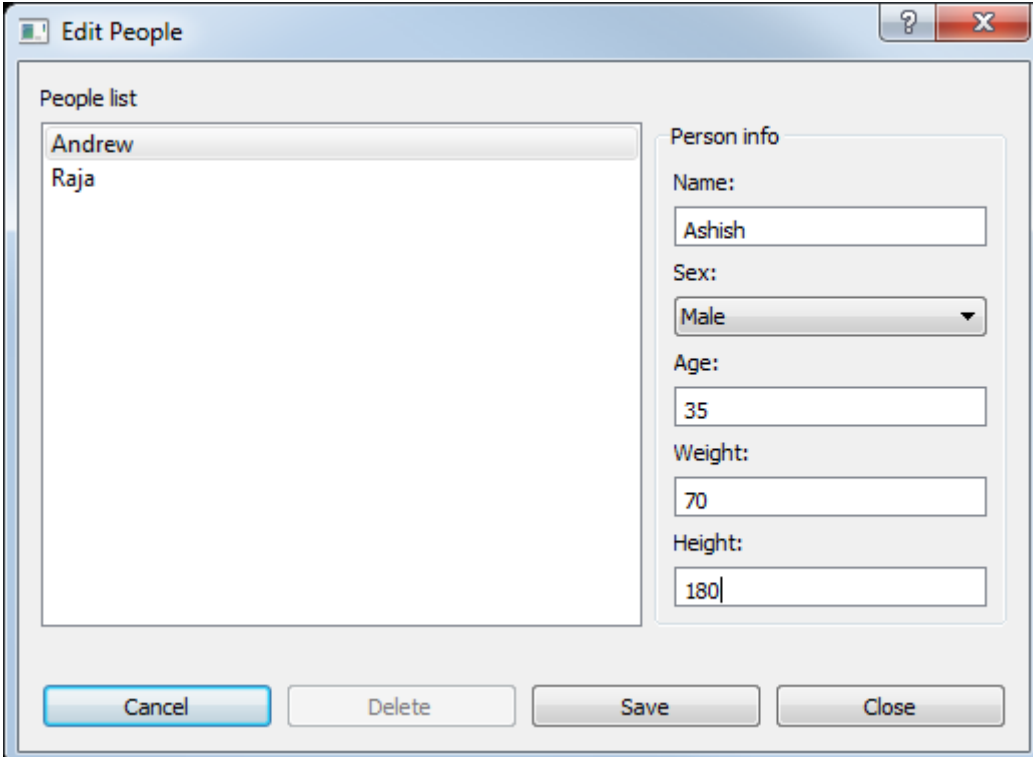Press the Save button to save information about the newly added person or press the Cancel button to decline any inputs and return to the person list view.

To delete a person from the list, select the person and press the Delete button. Note that when you delete a person all associated regions on all frames will be deleted too.

To change information about a person, select the name in the list, change values in the Person info area, and press Save.

Implementation details

Metadata structure

The next figure shows the metadata structure used in the photo book sample.

```
MetadataSchema
        │
        └──►  photo_book:MetadataSchema
                        │
                        ├──►  people_info:MetadataDesc
                        │                │
                        │                ├──►  item_1:Metadata
                        │                │
                        │                ├──►  item_2:Metadata
                        │                │
                        │                └──►  …..............
                        │
                        └──►  markup:MetadataDesc
                                        │
                                        ├──►  item_1:Metadata
                                        │
                                        ├──►  item_2:Metadata
                                        │
                                        └──►  …..............
```
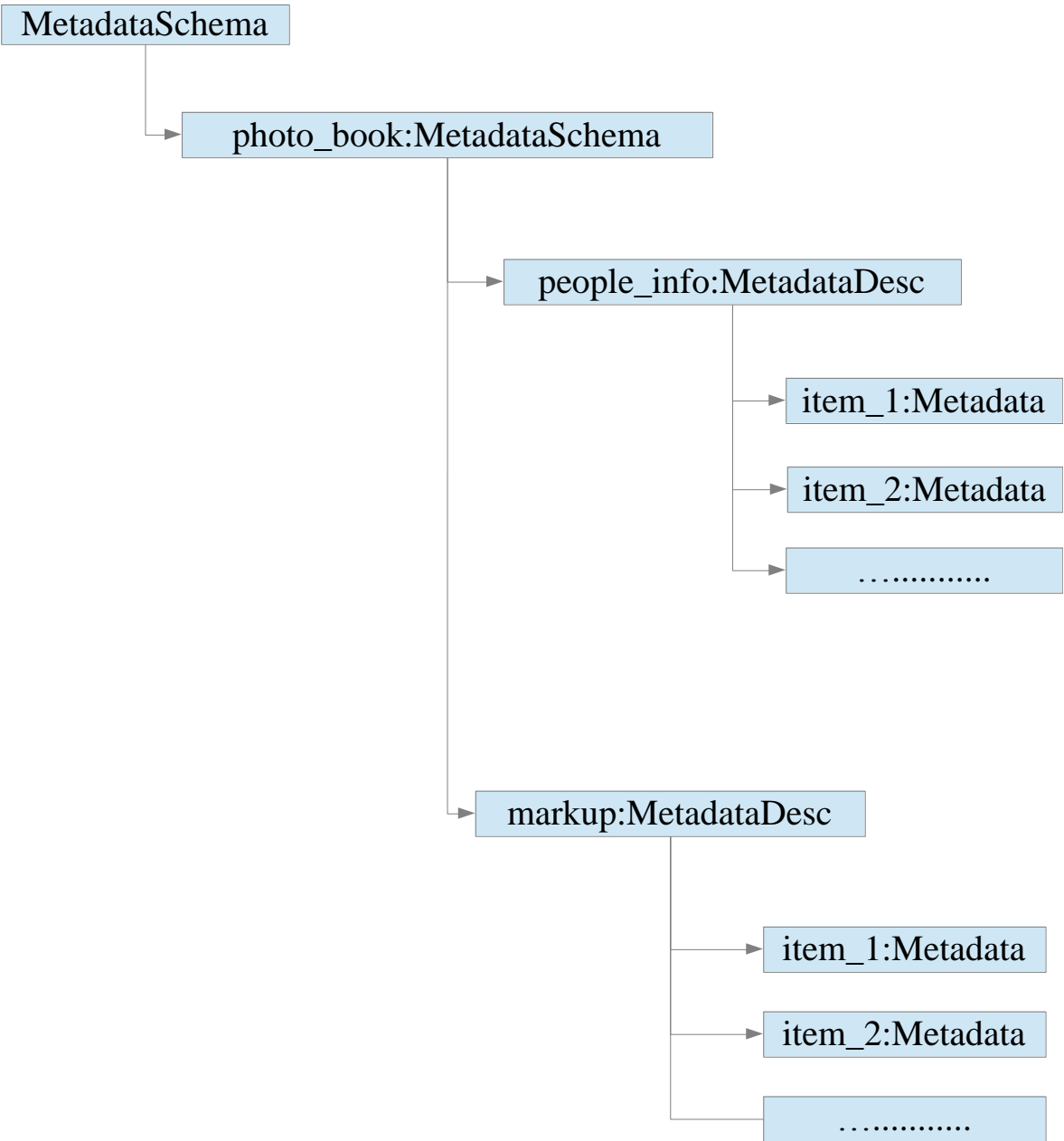
The <Name>:<EntityType> notation describes the type of metadata entity (MetaStream, Metadata, etc.) and name of its instance.

People_info items are Metadata that represent people_info MetadataDesc and contain information about all the people in the media file. The next table shows fields and their types of people_info property items:

| Field No.	Field name	Field type
1	name	type_string
2	sex	type_integer
3	age	type_integer
4	weight	type_integer
5	height	type_integer

Markup items are Metadata that represent markup MetadataDesc. This is service metadata that is used to store unassociated frame regions.

The next figure shows the association structure for an item of the people_info metadata. You can see that each item can be associated with zero or more regions on multiple frames.
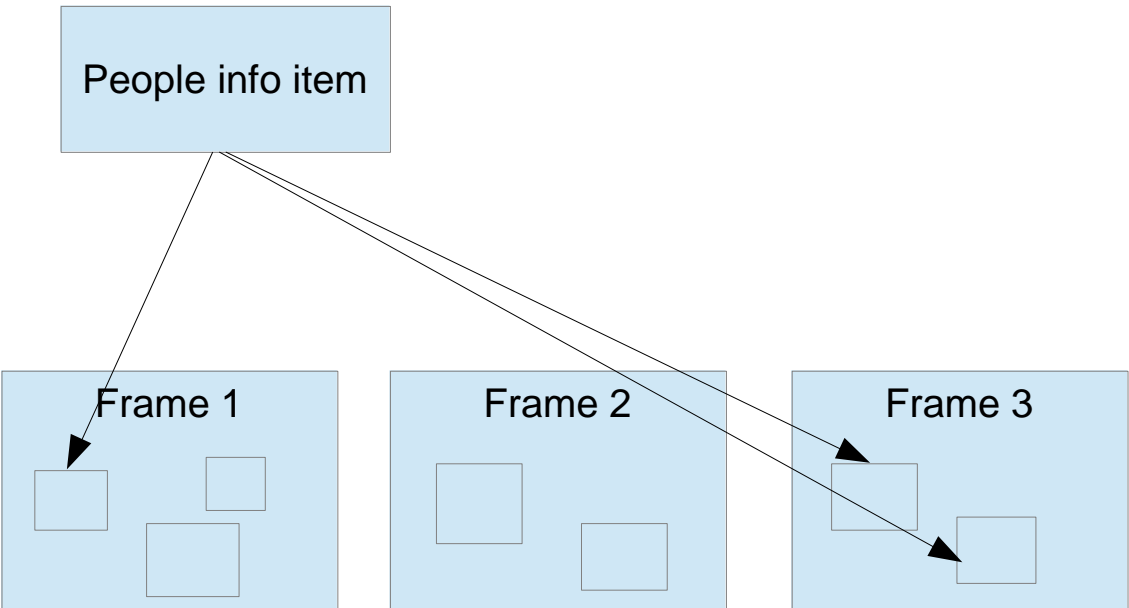
The next figure shows the association structure of an item of the markup Metadata type. You can see that one item can be associated with one region on a corresponding frame.

No... ...an be ass... ...rkup item ...
bu...

So...

Th... ...book ap...

File No.	File name	Comments
1	aboutwindow.hpp	About dialog header file
2	corner_grabber.hpp	CornerGrabber class header file
3	editwindow.hpp	People information edit form header file
4		...event
5		...ss head
6		...iewers
7		...m class
8	markup_model.hpp	MarkupModel class header file
9	metadata_helper.hpp	MetadataHelper class header file
10	photo_book.hpp	Photo book sample app common declarations
11	photographics.hpp	PhotoGraphics widget header file
12	aboutwindow.cpp	About dialog source file
13	corner_grabber.cpp	CornerGrabber class source file
14	editwindow.cpp	People information edit form source file
15	events.cpp	Internal event classes implementations
16	face_recognizer.cpp	FaceRecognizer class source file
17	face_rectangles.cpp	Source file contains associated and unassociated region viewers
18	main.cpp	Application startup file. Contains startup logic and Intel VMF library initialization/deinitialization code
19	mainwindow.cpp	Application main window source file
20	markup_model.cpp	MarkupModel class source file. Provide functionality to control frame regions

File No.	File name	Comments
21	metadata_helper.cpp	MetadataHelper class source file. Contains logic for loading and saving metadata
22	photo_book.cpp	Source file for some utility functions and constants
23	photographics.cpp	PhotoGraphics widget source file
24	images/1374079400_Add.png	PNG file. Add region icon
25	images/1374079438_Refresh.png	PNG file. Renew markup icon
26	images/1374079519_Delete.png	PNG file. Delete region icon
27	images/1375900454_Open.png	PNG file. Open file icon
28	images/1375901020_Save.png	PNG file. Commit markup icon
29	images/1375901254_Log Out.png	PNG file. Exit application icon
30	images/1375908250_Information.png	PNG file. About dialog icon
31	images/1375908611_iPhoto.png	PNG file. Photo book sample application icon
32	about.ui	About dialog UI file
33	editwindow.ui	People information edit form UI file
34	mainwindow.ui	App main window UI file
35	photographics.ui	PhotoGraphics widget UI file
36	resources.qrc	Application resource file

Ski resort

Overview

The ski resort application performs several simple queries on video that contains embedded GPS coordinates metadata.

The application consists of two parts:

- Creator is an application running on an Android device with a video camera and a GPS receiver. (For development purposes, you can run a "fake-creator" application on your desktop to embed GPS coordinates in a video file.)

- Visualizer runs on a desktop to display the embedded metadata.

The creator application records video and GPS coordinates with associated time relative to the start of the video file.

The visualizer application shows video and GPS positions associated with the current frame being displayed. In addition, it calculates the approximate speed of video camera.

Embedding GPS coordinates

To test the sample, embed GPS coordinates in your file by running the following command:

```
ski_resort_fake_creator.exe <GPS coordinates> <media file>,
```

where `<GPS coordinates>` is a text file with a list of GPS coordinates in the following format (it is assumed the step between each coordinates line is 1 second, the first coordinate is associated with the first second of the media file):

```
<x1 coordinate> <y1 coordinate>
<x2 coordinate> <y2 coordinate>
<x3 coordinate> <y3 coordinate>
```
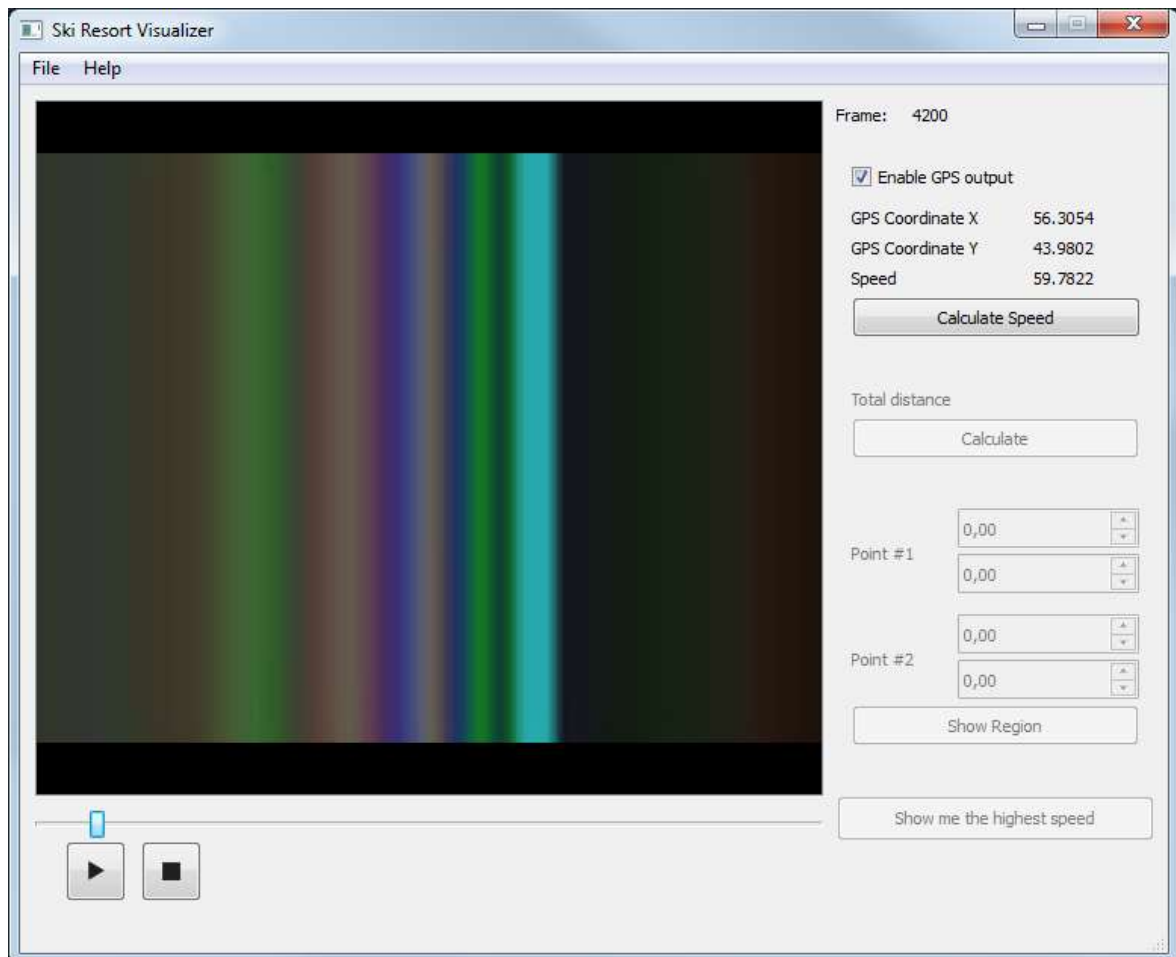
An example of the GPS file is located at `SOURCES/data/example_coordinates_for_ski_resort.txt`.

`<media file>` is the media file that contains the embedded GPS coordinates.

UI Description

Main Window

The next figure shows the main window of the ski resort visualizer.

The main window contains the controls. Some controls have not yet been implemented, but will be in the second stage of the development.

Open file

The next figure shows the open file dialog. To open a file, select the File → Open file menu item. To choose a file to work with, double-click the file you want to open. Another way is to select it and click Open.

In case another file is already opened in the visualizer, a dialog window with the question about saving a file will be shown. Click Yes to save changes and No to discard all the changes. The same window will be shown when you close the application.

Playback Control

You can move the slider located at the bottom of the window with your mouse for fast scrolling through the video file. You can also move the slider with the left and right arrow buttons on your keyboard. You can control payback of video using the Play/Resume and Stop buttons on the bottom.

In addition, the index of the current frame is shown in the top right corner of the main window.

Working with metadata

Initial development only includes visualization of GPS coordinates and features for displaying and calculating speed . All control elements for this functionality are located on the right side of the main window.

By unchecking the "Enable GPS output" box you can disable showing GPS coordinates and speed information. Checking the box re-enables the output.

You can press the Calculate Speed button to calculate speed if the currently opened file doesn't contain speed values. Calculated speed will be saved to the media file when you open another file or close the application. Calculated values will be shown in the Speed field if GPS output is enabled.

Implementation Details

This section explains implementation details of the ski resort sample application.

Intel® VMF SDK Entities used in the sample

The application uses one MetadataSchema and two MetadataDescs for storing all data in a video file. On the top hierarchy we see a MetadataSchema called "ski-resort". It should contain at least one Metadata called "gps-data". This property consists of a structure of GPS coordinates and associated local time. The structure includes two x and y fields of `type_real` for keeping latitude and longitude, respectively, and a time field of `type_integer`. This content is mandatory. The application will not be able to open a media file without those attributes.

The second Metadata, "speed", used in the ski resort sample is optional. It's intended to store speed values associated with the local time similar to the "gps-data" Metadata. It also contains values of `type_real` type and time as instance of `type_integer`.

The creator application should fill the "gps-data" metadata with values associated with time points relative to the beginning of the video. This feature provides fast coordinate access corresponding to the current timestamp.

Code Examples

All Intel VMF SDK related logic, except initialization and termination, is encapsulated inside the `VideoMetadataStream` class. Files corresponding to this class are `vmf/samples/ski-resort/visualizer/videometadatastream.cpp` and `vmf/samples/ski-resort/visualizer/videometadatastream.hpp`.

Initialization and termination is located in the main function of the visualizer application:

```cpp
int main(int argc, char *argv[])
{
    vmf::initialize();
    // ...
    // routines for starting qt application

    vmf::terminate();
    return retval;
```

```
}
```

Let's take a look at the main Intel VMF library entities declared as class members:

```
vmf::vmf_integer currentTime;
vmf::vmf_integer frameDelay;
vmf::MetadataStream metaStream;
vmf::MetadataSet gpsCoordinatesSet;
vmf::MetadataSet speedValuesSet;
```

The first task is opening a file and getting all mandatory data:

```
if (!metaStream.open(fileName, vmf::MetadataStream::ReadWrite))
{
    VMF_EXCEPTION(vmf::NotFoundException, "Not found");
}
if (!metaStream.load(SKI_RESORT_SCHEMA))
{
    VMF_EXCEPTION(vmf::DataStorageException, "Can't load metadata schema");
}
gpsCoordinatesSet = metaStream.queryByName(SKI_RESORT_GPS_DESC);
speedValuesSet = metaStream.queryByName(SKI_RESORT_SPEED_DESC);
```

Then check the existence of the speed property and get its values:

```
if (!speedValuesSet.empty())
{
    hasSpeedValues = true;
}
```

Now we have all needed metadata stored in the video file. Also, take a look at how Intel VMF SDK is used to calculate speed:

```cpp
for(size_t i = 0; i < gpsCoordinatesNumber - 1; ++i)
{
    auto currentCoordinatesItem = gpsCoordinatesSet[i];
    auto nextCoordinatesItem = gpsCoordinatesSet[i+1];

    float x1, x2, y1, y2;
    x1 = (float) (vmf::vmf_real) currentCoordinatesItem-
>getFieldValue(SKI_RESORT_GPS_FIELD_X);
    y1 = (float) (vmf::vmf_real) currentCoordinatesItem-
>getFieldValue(SKI_RESORT_GPS_FIELD_Y);
    x2 = (float) (vmf::vmf_real) nextCoordinatesItem-
>getFieldValue(SKI_RESORT_GPS_FIELD_X);
    y2 = (float) (vmf::vmf_real) nextCoordinatesItem-
>getFieldValue(SKI_RESORT_GPS_FIELD_Y);

    float meters = distance(x1, y1, x2, y2);
    vmf::vmf_integer t1 = currentCoordinatesItem-
>getFieldValue(SKI_RESORT_GPS_FIELD_TIME);
    vmf::vmf_integer t2 = nextCoordinatesItem-
>getFieldValue(SKI_RESORT_GPS_FIELD_TIME);
    vmf::vmf_integer timeDiff = t2 - t1;

    float seconds = (float) (timeDiff / 1000);
    float speed = (meters * KMS_IN_METER) / (seconds * HOURS_IN_SEC);

    std::shared_ptr<vmf::Metadata> speedMetadata(new vmf::Metadata(speedDesc));
    speedMetadata->setFieldValue(SKI_RESORT_SPEED_FIELD_SPEED, (vmf::vmf_real)
speed);
    speedMetadata->setFieldValue(SKI_RESORT_SPEED_FIELD_TIME, (vmf::vmf_integer)
t1);

    metaStream.add(speedMetadata);
}
```

Next, the `currentGpsCoordinateIndex` variable contains an index of items from `gpsCoordinatesSet` that corresponds to the current frame in the video. Now you can obtain the coordinates:

```cpp
auto currentGpsCoordinateItem = gpsCoordinatesSet[currentGpsCoordinateIndex];
currentCoordinates.first = (float) (vmf::vmf_real)
    currentGpsCoordinateItem->getFieldValue(SKI_RESORT_GPS_FIELD_X);
currentCoordinates.second = (float) (vmf::vmf_real)
    currentGpsCoordinateItem->getFieldValue(SKI_RESORT_GPS_FIELD_Y);
```

Where `currentCoordinates` is a variable of type `std::pair<float, float>`.

Since speed values are associated with the same time on calculation, we can also obtain speed:

```cpp
auto currentSpeedItem = speedValuesSet[currentGpsCoordinateIndex];
    currentSpeed = (float) (vmf::vmf_real) currentSpeedItem-
>getFieldValue(SKI_RESORT_SPEED_FIELD_SPEED);
```

Here the `currentSpeed` is a variable of type `float`.

Resources

This section includes links to useful resources:

- [Intel® VMF SDK on Intel Developer Zone](#).

- [Intel® VMF SDK 3.0 GitHub project](#)