How to build an application using RobartsVTK and PLUS using Windows

In this tutorial, you'll learn on how to setup your application to build (and work with) RobartsVTK and PLUS in Windows. The final code is presented in the end of the tutorial.

Prerequisites

For working with PLUS, you should have the software packages listed in the "Prerequisites" section of the Windows Build Instructions, which can be found in https://www.assembla.com/spaces/plus/wiki/Windows_Build_Instructions. Keep in mind that PLUS can only use a device that has an SDK if the SDK is available on the machine. If you are a researcher you might get it from the instructions in the link above (under "Device SDKs"). If not, it is on you to have the SDK files available for your build.

If not specified, PLUS will download and build VTK and ITK automatically. If you want to use a certain version of VTK and/or ITK, you should have them built from source, then specify the correspondent build directory (more details about that later).

Note that RobartsVTK requires VTK and might additionally require ITK and Qt (version 4) depending on the application, so if you decide to work only with RobartsVTK, you might still need to have them built/installed. In order to install Qt, follow the recommended approaches presented in the link above. In order to build VTK and ITK from source, follow the instructions below.

Building VTK

If you don't want to build VTK from source, skip this part.

In order to build VTK, you should firstly download the source code (available at http://www.vtk.org/download/). Unless specified, it is recommended that you download the latest release. Now extract the files you downloaded into a "src" directory and open CMake (you can download version 3.2 from http://www.cmake.org/download/). In CMake, it is recommended that you specify the following:

- Where is the source code: point to the "src" directory you extracted
- Where to build the binaries: point to a new "bin" directory

Now click on the "Configure" button and select your desired C++ compiler (if you want to use Visual Studio in 64-bits, for example, choose "Visual Studio 11 2012 Win64"). A lot of options will appear.

- Set BUILD_SHARED_LIBS on

If you want to work with Qt:

- Set VTK_Group_Qt on, if VTK version ≥ 6.0
- Set VTK_USE_QT on, if VTK version ≤ 5.10

If you are working with a VTK version older than 6.0 and you want to work with parallel programming:

- Set VTK_USE_PARALLEL on

Click on “Configure” again. If an error appears regarding the “qmake” file not found, you will have to specify it manually. Click on the field that is highlighted and on the “…” button on its side. Now you have to find and select the “qmake.exe” file. It should be in a similar path as:

C:\Qt\qt-4.8.6-x64-msvc2012\qt-4.8.6-x64-msvc2012\bin\qmake.exe

Make sure you select the x64 directory only if you are building VTK in 64-bits.

Click on “Configure” again. If you encounter more errors, try searching their keywords on an online search engine (such as Google), since other people might have encountered and solved the same errors you are getting. Keep on solving errors and clicking “Configure” until no more errors show up. When that happens, click on “Generate”.

Note that after clicking on “Generate”, the build directory you created now has a lot of files. If you are using Visual Studio, there should be a file called “VTK.sln”. It represents the solution to your build. Open that file and wait for Visual Studio to open and configure itself. Then press F7 or click on “Build -> Build Solution”. Note that this might take hours. For better compatibility, build in both Debug and Release modes.

Building ITK

If you don’t want to build ITK from source, skip this part.

In order to build ITK, download the source code from http://itk.org/ITK/resources/software.html. Note that for Windows you should download the .zip file. Extract the files you downloaded into a “src” directory and open CMake. Specify the following:

- Where is the source code: point to the “src” directory you extracted
- Where to build the binaries: point to a new “bin” directory

Now click on the “Configure” button and select your desired C++ compiler (if you want to use Visual Studio in 64-bits, for example, choose “Visual Studio 11 2012 Win64”). A lot of options will appear.

- Set BUILD_SHARED_LIBS on

Click on “Configure” again and then on “Generate”. Now open the “bin” directory you created, which should have new files created by CMake. If you are using Visual Studio, open the “ITK.sln” file. Then press F7 or click on “Build -> Build Solution”. Note that this might take some time. For better compatibility, build in both Debug and Release modes.

Basics

Now that you have all the prerequisites ready, be sure that your application uses CMake. Let’s suppose you have a very small application that only has one .cpp and one .h file. Along with these files you should have a CMakeLists.txt file. For building a VTK application, you can base your CMakeLists file on a VTK Wiki Example (such as http://www.vtk.org/Wiki/VTK/Examples/Cxx/Rendering/Cylinder). The

CMakeLists.txt file can be written using any fairly “programming” text editor. The simplest of them would be using Notepad or Notepad++. Notepad++ is recommended because it shows the color scheme of the different components in the language you are programming, beyond other things. You can get Notepad++ from https://notepad-plus-plus.org/download/v6.7.9.2.html. For a better understanding on how to write CMake files, you can check other tutorials online, such as http://www.cmake.org/cmake-tutorial/, http://mathnathan.com/2010/07/getting-started-with-cmake/ (suited for Linux, but listed here as a good starter guide), and http://ilcsoft.desy.de/portal/e279/e346/infoboxContent560/CMake_Tutorial.pdf.

For building such application, you should configure it through CMake and then build it in your preferred compiler. Assuming you are using Visual Studio, the steps you should do are similar to the ones on building VTK and ITK: open CMake and point the source code to the directory where your code and your CMakeLists.txt file are and point the binaries to a new directory - if the application is small, you can create a directory called “bin” inside the source code directory, if not, is it recommended to create a directory separate from the source code directory, for organization purposes. In this example, you should then simply click on “Configure” and “Generate”. If your project is more complex, you might need to set some CMake variables accordingly in the CMake GUI before clicking on the last “Configure”/”Generate” step.

Let’s get to the point

Now that you have your project CMake ready, let’s go into the details of building RobartsVTK and PLUS from your CMake setup. The most important function you will use in your CMake files is:

```
ExternalProject_Add(<name> [<option>…])
```

This function creates a custom target to build external projects. It works on the download, update, configure, build, install and test steps of an external project. You can check the function’s documentation in http://www.cmake.org/cmake/help/v3.2/module/ExternalProject.html for a better description.

The goal is to setup all the arguments you need to pass to this function. This includes setting all the options and variables that you would usually set on the CMake GUI if you were to build the external project manually, and giving them to CMake configure the project automatically.

See below a step-by-step on one way to set CMake to build an external project.

In this tutorial, I’m going to use a minimal code and CMake configuration. These will be my files:

tutorial.cpp

```
#include <iostream>

int main(void)
{
    std::cout << "Hello World!" << std::endl;

    return 0;
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.2)
```

```
PROJECT(Tutorial)

add_executable(Tutorial tutorial.cpp)
```

The files presented should work properly with CMake. Note that when you configure a project with CMake and open the ".sln" file to build it, it builds the "ALL_BUILD" project that CMake created. If you want to execute an executable code (which means, not just compile a library like we did with VTK and ITK before) in Visual Studio, you should right click on your project's name (in this example, the "Tutorial" project) and click on "Set as StartUp Project". Then you can click on "Run" or press F5.

Integrating PLUS

The first thing you want to do is set an option (variable) in your CMake GUI for enabling your project to build PLUS. This gives your application's user the freedom to choose whether or not to install PLUS without modifying your CMake code (assuming it would be available). In order to do that, you should open your CMakeLists.txt file in your preferred text editor (Notepad++ is recommended). And add the following lines:

```
PROJECT(Tutorial)

OPTION(TUT_USE_PLUS "Download and build the PLUS library" ON)

IF(TUT_USE_PLUS)

ENDIF(TUT_USE_PLUS)

add_executable(Tutorial tutorial.cpp)
```

The name "TUT_USE_PLUS" was chosen arbitrarily and can be replaced by any name you prefer. The advantage of writing a chosen prefix in the variable is that you can group all your custom variables by their prefix in the CMake GUI, so all your project's related variables would be grouped together as "TUT" in this example. In the `OPTION` function, you can provide a small description to the variable as well as define it an initial value.

So far this code only creates the option and verifies whether it was selected or not. The following code will be written inside the `IF` statement, so that the PLUS related modifications are done only if the user actually wants PLUS to be built.

Now you need to setup all the options and cache variables required to build PLUS. Some of them don't need to be defined by the user, but it is recommended to give the user freedom to decide the major components to build. The following code is extensive and more complicated than what you have done so far. Some explanation will be given for each part, but make sure to copy and paste it (in the same order the lines appear) between the `IF` and `ENDIF` lines that you have just written.

The following code was based in PLUS' own CMake files, which can be found in https://www.assembla.com/code/plus/subversion/nodes/HEAD/trunk/PlusBuild/CMakeLists.txt.

```
IF(TUT_USE_PLUS)

  SET(CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR}/Plus
${CMAKE_MODULE_PATH})
```

```
  FIND_PACKAGE(VTK)
  FIND_PACKAGE(ITK)

  # QT - check for Qt framework
  FIND_PACKAGE(QT REQUIRED)
  IF (NOT QT_FOUND)
    MESSAGE(FATAL_ERROR "PlusLib requires Qt. One of these components is
missing. Please verify configuration")
  ENDIF (NOT QT_FOUND)
```

This part firstly shows CMake to look for other modules inside a specific directory (that will be created later). Then, it looks for VTK and ITK (although nothing happens if it doesn't find it, since PLUS can build it too). Lastly, it looks for the Qt framework and shows an error if it doesn't find it in the system.

```
 # GIT - needed for the automatic update of the repository
  OPTION(PLUS_USE_GIT_PROTOCOL "If behind a firewall turn this off to use
http instead." ON)
  SET(GIT_PROTOCOL "git")
  IF(NOT PLUS_USE_GIT_PROTOCOL)
    SET(GIT_PROTOCOL "http")
  ENDIF(NOT PLUS_USE_GIT_PROTOCOL)

  MARK_AS_ADVANCED(CLEAR GIT_EXECUTABLE)
  FIND_FILE(GIT_EXECUTABLE git${CMAKE_EXECUTABLE_SUFFIX}
    PATHS "c:/Program Files/Git/bin/" "c:/Program Files (x86)/Git/bin/")
  FIND_PACKAGE(Git)
  IF(NOT GIT_FOUND)
    MESSAGE(FATAL_ERROR "error: Install Git and try to re-configure")
  ENDIF(NOT GIT_FOUND)
```

This part handles the Git protocol options and looks for Git in the system. Like the rest of the code presented, if it doesn't find it, it shows an error in CMake. The `SET` function defines or modifies a variable with the following data (in this case, a string).

```
  # SVN - needed for the automatic update of the repository
  MARK_AS_ADVANCED(CLEAR Subversion_SVN_EXECUTABLE)
  FIND_FILE(Subversion_SVN_EXECUTABLE svn${CMAKE_EXECUTABLE_SUFFIX}
    PATHS
      "c:/Program Files/TortoiseSVN/bin/"
      "c:/Program Files/SlikSvn/bin/"
    )
  FIND_PACKAGE(Subversion REQUIRED)

  # ASSEMBLA - Set username and password for assembla
  # Anonymous user name for Plus assembla server:
https://subversion.assembla.com/svn/plus/
  SET(PLUSBUILD_ASSEMBLA_USERNAME "perklab_anonymous" )
  SET(PLUSBUILD_ASSEMBLA_PASSWORD "anonymous" )

  SET(PLUS_SVN_REVISION 0 CACHE STRING "Set Plus desired SVN revision number
(0 means latest)")
```

This part tells CMake to look for subversion (SVN). Then it sets the PLUS's anonymous username and password for the Assembla server and defines a variable containing the desired SVN revision to download from it.

```
  # -----
  # Options to control build process
```

```
  # -----

  # Determine the operating system to set default values accordingly
  SET (ENABLED_BY_DEFAULT_ON_WINDOWS_ONLY OFF)
  SET (ENABLED_BY_DEFAULT_ON_WINDOWS32_ONLY OFF)
  if (CMAKE_HOST_WIN32)
    SET (ENABLED_BY_DEFAULT_ON_WINDOWS_ONLY ON)
    if (NOT CMAKE_CL_64)
      SET (ENABLED_BY_DEFAULT_ON_WINDOWS32_ONLY ON)
    endif ()
  endif (CMAKE_HOST_WIN32)
```

This code determines the operating system being used for customizing the build process (setting the default values accordingly).

```
  OPTION(PLUS_USE_OpenIGTLink "Use OpenIGTLink" ON)
  OPTION(PLUS_BUILD_SHARED_LIBS "Build shared libraries for PLUS." ON)

  # Documentation
  OPTION(PLUS_DOCUMENTATION "Build Plus documentation (Doxygen)." OFF)
  IF(PLUS_DOCUMENTATION)
    # Try to detect GraphViz path (CMake's Doxygen package finder only tries
some obsolete paths on Windows)
    # Parentheses is not permitted due to CMP0053
    SET(PROGRAMFILESX86 "ProgramFiles(x86)")
    find_program(DOXYGEN_DOT_EXECUTABLE
      NAMES dot
      PATHS
        "$ENV{ProgramFiles}/Graphviz2.38/bin"
        "$ENV{${PROGRAMFILESX86}}/Graphviz2.38/bin"
        "$ENV{ProgramFiles}/Graphviz2.34/bin"
        "$ENV{${PROGRAMFILESX86}}/Graphviz2.34/bin"
      DOC "Graphviz Dot tool for using Doxygen"
      NO_SYSTEM_ENVIRONMENT_PATH
    )
    find_package(Doxygen REQUIRED)

    IF(NOT DOXYGEN_FOUND)
      message(FATAL_ERROR "Documentation: Doxygen not found. Either specify
location of doxygen or disable PLUS_DOCUMENTATION.")
    ENDIF(NOT DOXYGEN_FOUND)
    IF(NOT DOXYGEN_DOT_FOUND)
      message(FATAL_ERROR  "Documentation:  Graphviz  dot  tool  not  found
(http://www.graphviz.org/Download.php,  required  by  Doxygen  for  diagram
generation).    Either    specify    location    of    dot    or    disable
PLUSBUILD_DOCUMENTATION.")
    ENDIF(NOT DOXYGEN_DOT_FOUND)
    OPTION(PLUS_DOCUMENTATION_SEARCH_SERVER_INDEXED   "Search   index   for
documentation is generated by the web server. Provides full-text search but
only works on web servers." OFF)
  ENDIF(PLUS_DOCUMENTATION)

  # Imaging hardware

  # determine current OS
  if("$ENV{PROCESSOR_ARCHITEW6432}" STREQUAL "")
    if("$ENV{PROCESSOR_ARCHITECTURE}" STREQUAL "x86")
```

```
      SET( TEMP_OS_ARCH "x86" )
    else()
      SET( TEMP_OS_ARCH "x64" )
    endif()
  else()
    SET( TEMP_OS_ARCH "x64" )
  endif()

  OPTION(PLUS_USE_ULTRASONIX_VIDEO "Provide support for the Ultrasonix
ultrasound systems" OFF)
  SET (PLUS_ULTRASONIX_SDK_MAJOR_VERSION 5 CACHE STRING "Set Ultrasonix SDK
major version (version: [major].[minor].[patch])")
  SET (PLUS_ULTRASONIX_SDK_MINOR_VERSION 7 CACHE STRING "Set Ultrasonix SDK
minor version (version: [major].[minor].[patch])")
  SET (PLUS_ULTRASONIX_SDK_PATCH_VERSION 4 CACHE STRING "Set Ultrasonix SDK
patch version (version: [major].[minor].[patch])")
  OPTION(PLUS_TEST_ULTRASONIX "Enable testing of acquisition from Ultrasonix
ultrasound systems. Enable this only if an Ultrasonix device accessible from
this computer. " OFF)
  IF (PLUS_TEST_ULTRASONIX)
    SET (PLUS_TEST_ULTRASONIX_IP_ADDRESS "130.15.7.24" CACHE STRING "IP
address of the Ultrasonix scanner that is used during testing")
  ENDIF(PLUS_TEST_ULTRASONIX)

  OPTION(PLUS_USE_BKPROFOCUS_VIDEO "Provide support for BK ProFocus
ultrasound systems through the OEM (TCP/IP) interface" OFF)
  IF (PLUS_USE_BKPROFOCUS_VIDEO)
    OPTION(PLUS_USE_BKPROFOCUS_CAMERALINK "Enable acquisition from BK
ProFocus ultrasound systems through CameraLink interface" OFF)
    OPTION(PLUS_TEST_BKPROFOCUS "Enable testing of acquisition from BK
ProFocus ultrasound systems. Enable this only if a BK ProFocus device is
connected to this computer. " OFF)
  ENDIF (PLUS_USE_BKPROFOCUS_VIDEO)
  IF ( (NOT ${CMAKE_GENERATOR} MATCHES "Win64") AND TEMP_OS_ARCH MATCHES "x64"
AND PLUS_USE_BKPROFOCUS_CAMERALINK)
    # warning regarding cross compilation of bkprofocus
    MESSAGE( "BK ProFocus support on a 64-bit OS requires 64-bit Plus build.
A 64-bit OS and a 32-bit Plus build configuration is detected. Compilation
will be successful, but the resulting executables will fail to start." )
  ENDIF ()
  IF ( PLUS_USE_BKPROFOCUS_CAMERALINK AND (NOT PLUS_USE_BKPROFOCUS_VIDEO) )
    MESSAGE(FATAL_ERROR "error: PLUS_USE_BKPROFOCUS_VIDEO must be enabled if
the PLUS_USE_BKPROFOCUS_CAMERALINK option is enabled" )
  ENDIF ()

  OPTION(PLUS_USE_ICCAPTURING_VIDEO "Provide support for the IC framegrabber
device" OFF)
  OPTION(PLUS_USE_VFW_VIDEO "Provide support for the Video-for-Windows video
digitizer (legacy, use Microsoft Media Foundation instead)" OFF)
  OPTION(PLUS_USE_MMF_VIDEO "Provide support for the Microsoft Media
Foundation video digitizers (requires installation of Windows Platform SDK
7.1 or later)" OFF)
  IF(PLUS_USE_MMF_VIDEO)
    OPTION(PLUS_TEST_MMF_VIDEO "Enable testing of acquisition from MMF video
device (webcam). Enable this only if an MMF device is connected to this
computer." OFF)
  ENDIF(PLUS_USE_MMF_VIDEO)
```

```
  OPTION(PLUS_USE_EPIPHAN "Provide support for the Epiphan framegrabber
device" OFF)
  OPTION(PLUS_USE_INTERSON_VIDEO "Provide support for the Interson USB
ultrasound probes" OFF)
  OPTION(PLUS_USE_INTERSONSDKCXX_VIDEO "Provide support for the Interson SDK
1.X with C++ Wrapper USB ultrasound probes" OFF)
  OPTION(PLUS_USE_TELEMED_VIDEO "Provide support for the Telemed ultrasound
probes" OFF)
  OPTION(PLUS_USE_THORLABS_VIDEO "Provide support for the ThorLabs Compact
Spectrometers" OFF)

  # Tracking hardware

  OPTION(PLUS_USE_POLARIS "Provide support for the NDI POLARIS and AURORA"
OFF)
  OPTION(PLUS_USE_CERTUS "Provide support for the NDI Certus" OFF)
  OPTION(PLUS_USE_MICRONTRACKER   "Provide   support   for   the   Claron
MicronTracker" OFF)
  OPTION(PLUS_USE_BRACHY_TRACKER "Provide support for the Brachy Steppers"
${ENABLED_BY_DEFAULT_ON_WINDOWS32_ONLY})
  OPTION(PLUS_USE_Ascension3DG "Provide support for the Ascension 3DG
Tracker" ${ENABLED_BY_DEFAULT_ON_WINDOWS32_ONLY})
  OPTION(PLUS_USE_Ascension3DGm "Provide support for the Ascension MedSafe
Tracker" OFF)
  IF (PLUS_USE_Ascension3DG AND PLUS_USE_Ascension3DGm)
    MESSAGE(FATAL_ERROR "PLUS_USE_Ascension3DGm and PLUS_USE_Ascension3DGm
options cannot be enabled at the same time. See more details at
https://www.assembla.com/spaces/plus/tickets/851")
  ENDIF()

  OPTION(PLUS_USE_PHIDGET_SPATIAL_TRACKER "Provide support for the Phidget
Spatial accelerometer" OFF)
  OPTION(PLUS_USE_3dConnexion_TRACKER "Provide support for the 3dConnexion 3d
mouse" OFF)
  OPTION(PLUS_USE_STEALTHLINK "Provide support for the Medtronick StealthLink
Server" OFF)
  OPTION(PLUS_USE_IntuitiveDaVinci "Provide support for the da Vinci Surgical
System" OFF)
  OPTION(PLUS_USE_PHILIPS_3D_ULTRASOUND "Provide support for the Philips ie33
3D ultrasound probe" OFF)
  IF(PLUS_USE_PHILIPS_3D_ULTRASOUND)
    OPTION(PLUS_TEST_PHILIPS_3D_ULTRASOUND "Enable testing of acquisition
from Philips 3D ultrasound systems. Enable this only if a Philips device is
accessible from this computer. " OFF)
  ENDIF(PLUS_USE_PHILIPS_3D_ULTRASOUND)
  IF (PLUS_TEST_PHILIPS_3D_ULTRASOUND)
    SET   (PLUS_TEST_PHILIPS_3D_ULTRASOUND_IP_ADDRESS   "129.100.44.8"   CACHE
STRING "IP address of the Philips scanner that is used during testing")
  ENDIF(PLUS_TEST_PHILIPS_3D_ULTRASOUND)

  # -----
  # Plus executable output path
  # -----
  SET (PLUS_EXECUTABLE_OUTPUT_PATH "${CMAKE_BINARY_DIR}/bin")
```

This part sets numerous options and variables needed for configuring PLUS' CMake build.

```
  # -----
```

```
  # Specify common external project properties
  # -----
  INCLUDE(${CMAKE_ROOT}/Modules/ExternalProject.cmake)
  INCLUDE(CTest)

  SET(ep_base "${CMAKE_BINARY_DIR}")

  SET(ep_common_args
    #-DCMAKE_INSTALL_PREFIX:PATH=${ep_install_dir}
    )
  IF(NOT ${CMAKE_GENERATOR} MATCHES "Visual Studio")
    SET(ep common args
      ${ep_common_args}
      -DCMAKE_BUILD_TYPE:STRING=${CMAKE_BUILD_TYPE}
      )
  ENDIF()

  SET(ep_common_c_flags "${CMAKE_C_FLAGS_INIT} ${ADDITIONAL_C_FLAGS}")
  SET(ep_common_cxx_flags "${CMAKE_CXX_FLAGS_INIT} ${ADDITIONAL_CXX_FLAGS}")

  # Compute -G arg for configuring external projects with the same CMake
generator:
  IF(CMAKE_EXTRA_GENERATOR)
    SET(gen "${CMAKE_EXTRA_GENERATOR} - ${CMAKE_GENERATOR}")
  ELSE()
    SET(gen "${CMAKE_GENERATOR}")
  ENDIF()
```

This part of the code enables the use of CTest and sets the arguments that will be passed to the compiler.

The following code sets the external projects for PLUS in a similar way that PLUS was set as an external project. Some files referenced will be created later in the tutorial. While some projects will be set as external projects, others will just require CMake to find the required packages in the user's system.

```
  # -----
  # Specify external projects
  # -----
  INCLUDE(Plus/External_VTK.cmake)
  INCLUDE(Plus/External_ITK.cmake)

  IF(PLUS_USE_OpenIGTLink)
    INCLUDE(Plus/External_OpenIGTLink.cmake)
  ENDIF(PLUS_USE_OpenIGTLink)

  IF ( PLUS_USE_BKPROFOCUS_VIDEO )
    INCLUDE(Plus/External_GrabbieLib.cmake)
  ENDIF()

  IF(PLUS_USE_CERTUS)
    FIND_PACKAGE (NDIOAPI)
    IF (NOT NDIOAPI_FOUND)
      MESSAGE( FATAL_ERROR "This project requires NDI Oapi for CERTUS
tracking. One of the components is missing. Please verify configuration or
turn off PLUS_USE_CERTUS.")
    ENDIF()
  ENDIF(PLUS_USE_CERTUS)
```

```
  IF (PLUS_USE_ULTRASONIX_VIDEO)
    FIND_PACKAGE (ULTRASONIX_SDK)
    IF (NOT ULTRASONIX_SDK_FOUND)
      MESSAGE( FATAL_ERROR "This project requires Ultrasonix SDK
${ULTRASONIX_SDK_VERSION} for Ultrasonix video. One of the components is
missing. Please verify configuration or turn off PLUS_USE_ULTRASONIX_VIDEO.")
    ENDIF()
  ENDIF(PLUS_USE_ULTRASONIX_VIDEO)

  IF ( PLUS_USE_MICRONTRACKER )
    FIND PACKAGE (MicronTracker)
    IF ( NOT MICRONTRACKER_FOUND)
      MESSAGE( FATAL_ERROR "This project requires Claron MicronTracker SDK
for supporting the MicronTracker tracking device. One of the components is
missing. Please verify configuration or turn off PLUS_USE_MICRONTRACKER.")
    ENDIF()
  ENDIF(PLUS_USE_MICRONTRACKER)

  IF ( PLUS_USE_ICCAPTURING_VIDEO )
   FIND_PACKAGE (ICCAPTURING)
    IF ( NOT ICCAPTURING_FOUND)
      MESSAGE( FATAL_ERROR "This project requires IC Capturing SDK for
supporting the Imaging Source USB frame grabber. One of the components is
missing. Please verify configuration or turn off
PLUS_USE_ICCAPTURING_VIDEO.")
    ENDIF()
  ENDIF(PLUS_USE_ICCAPTURING_VIDEO)

  IF ( PLUS_USE_STEALTHLINK )
    FIND_PACKAGE (STEALTHLINK)
    IF ( NOT STEALTHLINK_FOUND)
      MESSAGE( FATAL_ERROR "This project requires Stealthlink2 SDK for
supporting communication with Medtronic StealthStation. Please verify
configuration or turn off PLUS_USE_STEALTHLINK.")
    ENDIF()
  ENDIF()

  IF ( PLUS_USE_INTERSON_VIDEO )
   FIND_PACKAGE (INTERSON)
    IF ( NOT INTERSON_FOUND)
      MESSAGE( FATAL_ERROR "This project requires Interson iSDK for supporting
the Interson USB ultrasound probes. One of the components is missing. Please
verify configuration or turn off PLUS_USE_INTERSON_VIDEO.")
    ENDIF()
  ENDIF()

  IF( PLUS_USE_IntuitiveDaVinci)
    FIND_PACKAGE (IntuitiveDaVinci)
    IF( NOT IntuitiveDaVinci_FOUND )
      MESSAGE( FATAL_ERROR "This project requires headers and library provided
by Intuitive. One of the components is missing. Please verify configuration
or turn off PLUS_USE_IntuitiveDaVinci.")
    ENDIF()
  ENDIF()

  IF ( PLUS_USE_TELEMED_VIDEO )
```

```
    FIND_PACKAGE (Telemed)
    IF ( NOT TELEMED_FOUND )
      MESSAGE( FATAL_ERROR "This project requires Telemed SDK for supporting
the Telemed ultrasound probes. One of the components is missing. Please verify
configuration or turn off PLUS_USE_TELEMED_VIDEO.")
    ENDIF()
  ENDIF()

  IF ( PLUS_USE_THORLABS_VIDEO )
    FIND_PACKAGE (ThorLabs)
    IF ( NOT THORLABS_FOUND )
      MESSAGE( FATAL ERROR "This project requires ThorLabs CCS VISA  SDK for
supporting the ThorLabs devices. One of the components is missing. Please
verify configuration or turn off PLUS_USE_THORLABS_VIDEO.")
    ENDIF()
  ENDIF()

  IF (PLUS_USE_PHILIPS_3D_ULTRASOUND)
    IF(NOT CMAKE_HOST_WIN32)
    # Philips is windows only
    MESSAGE( FATAL_ERROR "Philips SDK is only available for Windows.")
    ENDIF(NOT CMAKE_HOST_WIN32)
    IF(PLUS_USE_ULTRASONIX_VIDEO)
    # Ultrasonix contains its own libmmd.dll which does not support all the
functionality needed for the Philips probe
    # They both cannot be enabled at the same time.
    MESSAGE( FATAL_ERROR "Plus cannot enable both Ultrasonix and Philips
devices due to .dll conflicts in their respective SDK packages.")
    ELSE(PLUS_USE_ULTRASONIX_VIDEO)
      SET (PLUS_Philips_MAJOR_VERSION 1 CACHE STRING "Set Philips library
major version (version: [major].[minor].[patch])")
      SET (PLUS_Philips_MINOR_VERSION 0 CACHE STRING "Set Philips library
minor version (version: [major].[minor].[patch])")
      SET (PLUS_Philips_PATCH_VERSION 0 CACHE STRING "Set Philips library
patch version (version: [major].[minor].[patch])")
      FIND_PACKAGE(Philips)
      IF( NOT PHILIPS_FOUND )
        MESSAGE( FATAL_ERROR "In order to use the Philips ie33 ultrasound
system, the requisite DLLs must be made available. Please verify configuration
or turn off PLUS_USE_PHILIPS_3D_ULTRASOUND.")
      ENDIF(NOT PHILIPS_FOUND)
    message( "To use the Philips devices you must:" )
    message(                          "1.                          Register
'${CMAKE_CURRENT_BINARY_DIR}/bin/Debug|Release/Stream3d.dll'.  If  you  are
using a 32-bit OS, this can be done by running 'regsvr32 Stream3d.dll' in
command line. If you are using a 64-bit OS, this can be done by the following
procedure. Open the command line in administrator mode (Right click on cmd,
choose 'Run as administrator'). Go to directory 'C:/Windows/SysWow64'. Run
the regsvr32 command in SysWow64." )
    ENDIF(PLUS_USE_ULTRASONIX_VIDEO)
  ENDIF(PLUS_USE_PHILIPS_3D_ULTRASOUND)
```

The first lines of this part of the code tell CMake to include (and execute) the files referenced. These files (which you will write later) include more definitions for specifying and adding the external projects (including the `ExternalProject_Add` function mentioned earlier).

```
  # -----
```

```
  # Specify target dependencies
  # -----
  SET(PlusLib_DEPENDENCIES)

  IF ( NOT VTK_DIR )
    # VTK_DIR is not supplied, so it is built inside Plus, therefore we need
to specify dependencies to make sure it is built early enough
    SET(VTK_DEPENDENCIES)
    LIST(APPEND PlusLib_DEPENDENCIES vtk)
  ENDIF()

  IF ( NOT ITK_DIR )
    # ITK_DIR is not supplied, so it is built inside Plus, therefore we need
to specify dependencies to make sure it is built early enough
    SET(ITK_DEPENDENCIES)
    LIST(APPEND PlusLib_DEPENDENCIES itk)
  ENDIF()

  IF(PLUS_USE_OpenIGTLink AND NOT OpenIGTLink_DIR)
    # OpenIGTLink_DIR is not supplied, so it is built inside Plus, therefore
we need to specify dependencies to make sure it is built early enough
    SET(OpenIGTLink_DEPENDENCIES)
    LIST(APPEND PlusLib_DEPENDENCIES OpenIGTLink)
  ENDIF()

  IF ( PLUS_USE_BKPROFOCUS_VIDEO )
    SET(GrabbieLib_DEPENDENCIES)
    LIST(APPEND PlusLib_DEPENDENCIES GrabbieLib)
  ENDIF()

  IF( PLUS_USE_INTERSONSDKCXX_VIDEO AND NOT IntersonSDKCxx_DIR )
    SET(IntersonSDKCxx_DEPENDENCIES)
    LIST(APPEND PlusLib_DEPENDENCIES IntersonSDKCxx)
    INCLUDE(Plus/External_IntersonSDKCxx.cmake)
  ENDIF()

  INCLUDE(Plus/External_PlusLib.cmake)

  SET(PLUSLIB_DIR ${CMAKE_BINARY_DIR}/PlusLib-bin CACHE PATH "The directory
containing PlusLib binaries" FORCE)

ENDIF(TUT_USE_PLUS)
```

Finally all the dependencies are setup for this project (so that some package that depends on another doesn't build before it should). One more external project file is also included, then you can set the PLUS library directory into a CMake variable to indicate that PLUS was indeed configured.

Now you need to get the files that were referenced before. For organization purposes, create a directory called "Plus" in your source directory – this is where all your PLUS related files will be. You can copy the most recent files directly from the PLUS's SVN page (https://www.assembla.com/code/plus/subversion/nodes/HEAD/trunk/PlusBuild), but some variables' names need to be slightly modified. Make sure to copy the following files and make the following changes:

External_VTK.cmake

External_ITK.cmake

External_OpenIGTLink.cmake

External_GrabbieLib.cmake

External_IntersionSDKCxx.cmake

After copying them, open these files in your preferred text editor and use the Find/Replace tool to rename the variables PLUSBUILD_USE_VTK6 and PLUSBUILD_BUILD_SHARED_LIBS to PLUS_USE_VTK6 and PLUS_BUILD_SHARED_LIBS whenever they appear.

These files firstly check if the correspondent package is already installed (and was found), by checking the "PACKAGE"_DIR variable. If it was defined, it means that it was found, so they just configure PLUS to use it. If it wasn't found, they set some more arguments and execute the `ExternalProject_Add` function with all the configuration variables needed, so that PLUS can build it as well.

Now you need to copy some other files. Their function is to provide CMake the paths to look for each described component. You can copy them from PLUS' SVN page as well in https://www.assembla.com/code/plus/subversion/nodes/HEAD/trunk/PlusBuild/Modules.

FindCCapturing.cmake

FindIntersion.cmake

FindIntuitiveDaVinci.cmake

FindMicronTracker.cmake

FindNDIOAPI.cmake

FindPhilips.cmake

FindSTEALTHLINK.cmake

FindTelemed.cmake

FindThorLabs.cmake

FindULTRASONIX_SDK.cmake

FindWindowsSDK.cmake

PlusTestWindowsSdkCompatible.cxx

TestWindowsSDK.cmake

There is one more (and very important) file that you need to add. You can copy its latest version from PLUS' SVN page (https://www.assembla.com/code/plus/subversion/nodes/HEAD/trunk/PlusBuild), but some modifications presented here need to be done.

External_PlusLib.cmake

- Rename the following variables:
  PLUSBUILD_ADDITIONAL_SDK_ARGS to PLUS_ADDITIONAL_SDK_ARGS

PLUSBUILD_USE_OpenIGTLink to PLUS_USE_OpenIGTLink
PLUSBUILD_DOCUMENTATION to PLUS_DOCUMENTATION

- Delete the following Slicer related code:

```
IF (PLUSBUILD_USE_3DSlicer)
  SET(PLUSBUILD_ADDITIONAL_SDK_ARGS ${PLUSBUILD_ADDITIONAL_SDK_ARGS}
    -DSLICER_BIN_DIRECTORY:PATH=${PLUSBUILD_SLICER_BIN_DIRECTORY}
    )
ENDIF()
…
-DPLUS_USE_SLICER:BOOL=${PLUSBUILD_USE_3DSlicer}
```

- Delete some code we have added in the CMakeLists.txt file:

```
SET(PLUSLIB_DIR ${CMAKE_BINARY_DIR}/PlusLib-bin CACHE PATH "The directory
containing PlusLib binaries" FORCE)
```

External_PlusLib.cmake is the file that actually adds the PLUS library itself. Note that all the options and variables that we have set before are used here.

With these modified files you are now able to run CMake and have it build PLUS correctly. Just configure it through CMake, choosing the components you need, and go through the same "Configure"/"Generate"/Build process described before.

If you already have VTK or ITK installed, CMake will probably find them. In case it doesn't find them, you can manually set the VTK-DIR or ITK-DIR variable to the path of your binaries directory, i.e. "VTK-bin" or however you called it. Just click on the variable, on the "…" button besides it, and point to the right path. If those variables are not defined when CMake generates your build, it will set your project to build VTK and/or ITK as well.

Integrating RobartsVTK

This part will be simpler than integrating PLUS. Firstly, you should set an option for using RobartsVTK, as it gives the user freedom to choose whether they want to install RobartsVTK or not. In order to do that, add the following lines to your CMakeLists.txt file:

```
ENDIF(TUT_USE_PLUS)

OPTION(TUT_USE_RobartsVTK "Download and build the RobartsVTK library" ON)

IF(TUT_USE_RobartsVTK)

ENDIF(TUT USE RobartsVTK)

add_executable(Tutorial tutorial.cpp)
```

Inside the `IF` statement, you should now provide all the options for building RobartsVTK, so the user can configure the appropriate build options.

```
IF(TUT_USE_RobartsVTK)

  # Git
  OPTION(RobartsVTK_USE_GIT_PROTOCOL "If behind a firewall turn this off to
use http instead." ON)
```

```
  SET(RobartsVTK_GIT_PROTOCOL "git")
  IF(NOT RobartsVTK_USE_GIT_PROTOCOL)
    SET(RobartsVTK_GIT_PROTOCOL "http")
  ENDIF(NOT RobartsVTK_USE_GIT_PROTOCOL)

  MARK_AS_ADVANCED(CLEAR GIT_EXECUTABLE)
  FIND_FILE(GIT_EXECUTABLE git${CMAKE_EXECUTABLE_SUFFIX}
    PATHS "c:/Program Files/Git/bin/" "c:/Program Files (x86)/Git/bin/")
  FIND_PACKAGE(Git)
  IF(NOT GIT_FOUND)
    MESSAGE(FATAL_ERROR "error: Install Git and try to re-configure")
  ENDIF(NOT GIT_FOUND)
```

This part handles the Git protocol and finds the Git executable in the system. As described before, if it doesn't find it, an error message shows in the CMake log.

```
  FIND_PACKAGE(VTK REQUIRED)

  OPTION(RobartsVTK_BUILD_SHARED_LIBS "Build shared libraries for RobartsVTK"
OFF)
  OPTION(RobartsVTK_BUILD_DOCUMENTATION "Build Documentation for RobartsVTK"
OFF)
  OPTION(RobartsVTK_BUILD_EXAMPLES "Build RobartsVTK examples." OFF)

  OPTION(RobartsVTK_USE_ITK "Use ITK in RobartsVTK" OFF)
  IF(RobartsVTK_USE_ITK)
    FIND_PACKAGE(ITK QUIET)
  ENDIF(RobartsVTK_USE_ITK)
```

This part firstly looks for VTK, as it is required for building RobartsVTK. Then it shows the first options that are needed for building RobartsVTK

```
IF(${VTK_MAJOR_VERSION} GREATER 4)
    # LIBXML2_FOUND: We need this variable because CMake's FindLibXml2.cmake
module does
    #                not set LibXml2_DIR like most FIND_PACKAGE modules.
    OPTION(RobartsVTK_USE_LIBXML2 "Use libxml2 in RobartsVTK (only VTK 5 and
higher)" ON)
    IF(RobartsVTK_USE_LIBXML2)
      FIND_PACKAGE(LibXml2 QUIET)
    ENDIF(RobartsVTK_USE_LIBXML2)
  ENDIF(${VTK_MAJOR_VERSION} GREATER 4)
```

Here it is provided the option to build RobartsVTK with LibXml2. We use the `IF` statement to make sure that this is only enabled when available (when the VTK version is 5 or higher).

```
  OPTION (RobartsVTK_USE_QT "Build with QT enabled" OFF)
  IF (RobartsVTK_USE_QT)
    FIND_PACKAGE(QT REQUIRED)
  ENDIF (RobartsVTK_USE_QT)

  OPTION (RobartsVTK_USE_MWIERZ "Build the vtkMwierz module" OFF)
  OPTION       (RobartsVTK_USE_ROBARTSUTILITIES       "Build       the
vtkRobartsUtilities module" OFF)

  OPTION (RobartsVTK_USE_CUDA "Build the CUDA modules" OFF)
  IF (RobartsVTK_USE_CUDA)
```

```
    FIND_PACKAGE(CUDA REQUIRED)
    INCLUDE(FindCUDA)
    OPTION   (RobartsVTK_USE_CUDA_VISUALIZATION   "Build   the   CUDA
visualization modules" ON)
    IF (RobartsVTK_USE_CUDA_VISUALIZATION)
      OPTION  (RobartsVTK_USE_CUDA_ANALYTICS  "Build  the  CUDA  image
analytics modules" ON)
    ENDIF (RobartsVTK_USE_CUDA_VISUALIZATION)
  ENDIF (RobartsVTK_USE_CUDA)

  INCLUDE(${CMAKE_ROOT}/Modules/ExternalProject.cmake)
  INCLUDE(External_RobartsVTK.cmake)

  SET(RobartsVTK_DIR  ${CMAKE_BINARY_DIR}/RobartsVTK-bin  CACHE  PATH
"The directory containing RobartsVTK binaries" FORCE)

ENDIF(TUT_USE_RobartsVTK)
```

This part provides the rest of the options, includes a file which you still need to write (the file that has the `ExternalProject_Add` function for actually including RobartsVTK in this build), and defines a variable called RobartVTK_DIR for controlling whether it was built or not.

Now you should write the file needed. Using your preferred editor, create a new file called External_RobartsVTK.cmake. You can save it in the "src" directory, since it is the only new file regarding RobartsVTK. In this file, you should write:

```
SET(RobartsVTK_GIT_REPOSITORY
"git.imaging.robarts.ca/RobartsVTK.git")
SET(RobartsVTK_GIT_TAG "88b61f90e09f26f6cc285ec6bbd9a515fd92f9ba")

SET  (RobartsVTK_DIR  ${CMAKE_BINARY_DIR}/RobartsVTK  CACHE  INTERNAL
"Path to store RobartsVTK contents.")
ExternalProject_Add(RobartsVTK
  SOURCE_DIR "${RobartsVTK_DIR}"
  BINARY_DIR "RobartsVTK-bin"
  #--Download step--------------
  GIT_REPOSITORY
"${RobartsVTK_GIT_PROTOCOL}://${RobartsVTK_GIT_REPOSITORY}"
  GIT_TAG ${RobartsVTK_GIT_TAG}
  #--Configure step-------------
  CMAKE_ARGS
    ${ep_common_args}
    -DVTK_DIR:PATH=${VTK_DIR}
    -DITK_DIR:PATH=${ITK_DIR}
    -DBUILD_DOCUMENTATION:BOOL=${RobartsVTK_DOCUMENTATION}
    -DBUILD_SHARED_LIBS:BOOL=${RobartsVTK_BUILD_SHARED_LIBS}
    -DBUILD_TESTING:BOOL=${RobartsVTK_BUILD_TESTING}
    -DRobartsVTK_BUILD_EXAMPLES=${RobartsVTK_BUILD_EXAMPLES}
    -DRobartsVTK_USE_CUDA=${RobartsVTK_USE_CUDA}
    -DCUDA_HOST_COMPILER=${CUDA_HOST_COMPILER}
    -DCUDA_SDK_ROOT_DIR=${CUDA_SDK_ROOT_DIR}
    -DCUDA_TOOLKIT_ROOT_DIR=${CUDA_TOOLKIT_ROOT_DIR}
```

```
    -
DRobartsVTK_USE_CUDA_VISUALIZATION=${RobartsVTK_USE_CUDA_VISUALIZATIO
N}
    -DRobartsVTK_USE_CUDA_ANALYTICS=${RobartsVTK_USE_CUDA_ANALYTICS}
    -DRobartsVTK_USE_ITK=${RobartsVTK_USE_ITK}
    -DRobartsVTK_USE_LIBXML2=${RobartsVTK_USE_LIBXML2}
    -DRobartsVTK_USE_MWIERZ=${RobartsVTK_USE_MWIERZ}
    -DRobartsVTK_USE_QT=${RobartsVTK_USE_QT}
    -DQT_QMAKE_EXECUTABLE:FILEPATH=${QT_QMAKE_EXECUTABLE}
    -DDESIRED_QT_VERSION=${DESIRED_QT_VERSION}
    -
DRobartsVTK_USE_ROBARSTUTILITIES=${RobartsVTK_USE_ROBARSTUTILITIES}
    -DCMAKE_CXX_FLAGS:STRING=${ep_common_cxx_flags}
    -DCMAKE_C_FLAGS:STRING=${ep_common_c_flags}
    -
DROBARTSVTK_EXECUTABLE_OUTPUT_PATH:STRING=${RobartsVTK_EXECUTABLE_OUT
PUT_PATH}

  #--Build step-----------------
  #--Install step---------------
  INSTALL_COMMAND ""
  DEPENDS ${RobartsVTK_DEPENDENCIES}
  )
```

This file does two tasks mainly: defines the Git repository's URL and the Git tag, which in this case is a commit number; and adds the external project with all the options you have set before. You can add a new option in the CMakeLists file to enable the user to choose which Git tag to use, but in this case you are only allowing the developer to choose which tag is appropriate for building. Make sure to update the Git tag if you want to use an up to date version of RobartsVTK, or simply remove the green lines if you always want to clone the master branch.

These files are enough for building RobartsVTK correctly. Just configure it through CMake, choosing the components you need, and go through the same "Configure"/"Generate"/Build process described before.

You are now able to build PLUS and RobartsVTK within your application. You can find this tutorial's code in: https://github.com/nicolezk1/ep-tutorial/tree/master/src.