Universidad Veracruzana

Department of Artificial Intelligence

NAO Developers Tutorial: with Naoqi C++ SDK

Author:
Antonio Marin-Hernandez

January 30, 2014

Contents

Chapter 1

Installation

1.1 Requirements

- Linux

  GCC version 4.4 or higher, CMake, Python version 2.7 (strictly) and a recent version of QtCreator.

- Mac

  Xcode, GCC version 4.4 or higher, CMake, Python version 2.7 (strictly) and a recent version of QtCreator.

1.2 Installation

Retrieve the C++ SDK archive corresponding to your Operating System (Mac or Linux) from Aldebaran Community Website:

```
naoqi-sdk-1.14.x-[OS].tar.gz
```

Extract it on your machine, preferentially as root, for example in /opt/-NAO/ if root is used or at your home path if not.

Retrieve and extract the qiBuild archive:

```
qibuild-x.zip
```

Install it by running:

```
./instal-qibuild.sh
```

and then configure it by:

```
qibuild config -wizard
```

To do it, verify that `/.local/bin` is in your `PATH`

1.3 Environment Setup

1.3.1 Creating a Worktree

First you need to choose a `qibuild` worktree directory, it's required an empty directory, so you can create a new one, called for example:`/home/nao/worktree/`

now, go to your worktree directory

```
$ cd <path/to/your/worktree>
```

This path will be the root from where `qibuild` will search to find your projects. Now initialize your worktree directory by running:

```
$ qibuild init --interactive
```

In order to access easily to your work tree put following lines in your `.bashrc` or `.login` file, depending on your system:

```
export NAO_WORKSPACE=<path/to/your/worktree>
alias naocd="cd  $ NAO_WORKSPACE"
```

Create a toolchain by:

```
$ qitoolchain create linux <path/to/cpp/sdk>/toolchain.xml --
    default
```

To ask about information in the current toolchain you can give option `info` to `qitoolchain` and you should you get something like:

```
$ qitoolchain info
Toolchain cross-geode
Using feed from /opt/NAO/linux64-nao-geode-cross-toolchain-1.14.5/
    toolchain.xml
  Packages:
    naoqi-geode-ctc
      in /opt/NAO/linux64-nao-geode-cross-toolchain-1.14.5
      using /opt/NAO/linux64-nao-geode-cross-toolchain-1.14.5/
          cross-config.cmake
toolchain file
      sysroot: /opt/NAO/linux64-nao-geode-cross-toolchain-1.14.5/
          sysroot
      cross-gdb: /opt/NAO/linux64-nao-geode-cross-toolchain
          -1.14.5/cross/bin/i686-aldebaran-linux-gnu-gdb

Toolchain naoqi-sdk
Using feed from /opt/NAO/naoqi-sdk-1.14.5-linux64/toolchain.xml
  Packages:
```

```
    naoqi-sdk-linux64
      in /opt/NAO/naoqi-sdk-1.14.5-linux64
```

If you have more than one toolchain you can change toolchain to use modifying the `qibuild.xml` file in <your/path/to/worktree>/.qi directory

Chapter 2

Writing with C++

In order to create a project, then configure it and build it, you need to write the following lines:

```
$ qisrc create my_module
New project initialized in </path/to/worktree>/my_module
```

you will create a directory named my_module with the following files: my_module CMakeLists.txt qiproject.xml main.cpp test.cpp
To verify this, write:

```
$ cd my_module
$ ls
CMakeLists.txt  main.cpp qiproject.xml test.cpp
```

Now you can configure and make your module by:

```
$ qibuild configure my_module
Current build worktree: <path/to/worktree>
Using toolchain: cross-geode
Build type: Debug
* (1/1) Configuring my_module
-- Using qibuild v3.2
-- Binary: my_module
-- Binary: test_my_module
-- Configuring done
-- Generating done
-- Build files have been written to: </path/to/worktree>/my_module
/build-cross-geode
```

```
$ qibuild make my_module
Current build worktree: </path/to/worktree>
Using toolchain: cross-geode
Build type: Debug
* (1/1) Building my_module
Scanning dependencies of target my_module
[ 50\%] Building CXX object CMakeFiles/my_module.dir/main.cpp.o
Linking CXX executable sdk/bin/my_module
[ 50\%] Built target my_module
```

```
Scanning dependencies of target test_my_module
[100\%] Building CXX object CMakeFiles/test_my_module.dir/test.cpp
    .o
Linking CXX executable sdk/bin/test_my_module
[100\%] Built target test_my_module
```

Your new package will create a `build-<tootchain>` directory having between them the executable files created.

To run executable you can use `qibuild`. First of all we are going to test the module by :

```
$ qibuild test my_module
Current build worktree: </path/to/worktree>
Using toolchain: naoqi-sdk
Build type: Debug
Testing my_module ...
* (1/1) test_my_module   [OK]
Ran 1 tests in 0s
All pass. Congrats!
```

Now we can run my_module example by:

```
$ qibuild run my_module
Current build worktree: </path/to/worktree>
Using toolchain: naoqi-sdk
Build type: Debug
Hello, world
```

2.1 Connecting our module to the Robot

Lets modify the `main.cpp` file by adding the following lines:

```
#include <alproxies/altexttospeechproxy.h>
```

and in main function

```
AL::ALTextToSpeechProxy tts("<your NAO IP or address>", 9559);
tts.say("Hello, my name is NAO, nice to meet you!");
```

you should also need to modify your `CMakeLists.txt` in order to add ALCOMMON package.

Adding the following line `qi_use_lib(my_module ALCOMMON)` just after `qi_create_bin(my_module "main.cpp")`

Lets configure our project and make it with `qibuild`. Now you can execute your module by:

```
$ qibuild run my_module
Current build worktree: </path/to/worktree>
Using toolchain: naoqi-sdk
Build type: Debug
[INFO ] Starting ALNetwork
```

```
[INFO ] NAOqi is listening on 127.0.0.1:54010
Hello, world
[INFO ] Stopping ALNetwork
[INFO ] Exit
```

The robot should say the text.

2.2 Proxies

There are two ways to communicate with proxies

Specialized proxies as the example it has been done and Generic proxies

Make a new file in my_module directory named `genericproxies.cpp` And put the following lines

```
#include <iostream>
#include <alcommon/alproxy.h>

int main()
{

  const std::string phraseToSay = "Hello world";
  AL::ALProxy proxy("ALTextToSpeech", "148.226.110.96", 9559);
  proxy.callVoid("say", phraseToSay);

 bool ping = proxy.call<bool>("ping");

  std::cout << "Hello, world" << std::endl;
  return 0;
}
```

Now, lets modify `CMakelists.txt` to compile the new code by adding the following lines:

```
# Create a executable named genericproxies
# with the source file: genericproxies.cpp
qi_create_bin(genericproxies "genericproxies.cpp")
qi_use_lib(genericproxies ALCOMMON)
```

Configure and make again the module with `qibuild`. Then run new app by:

```
$ qibuild run genericproxies
```

you should get something very similar to the former app.

Always prefer this kind of proxy to generic proxies if it exists for a desired module. These proxies are optimized, efficient and easy to use: once created, they give direct access to already existing methods.

However, generic proxies give access to any module including the ones, which also have specialized proxies. But, the generic proxy has no information about the methods that are bound to these modules, contrary to specialized proxies.

Chapter 3

Doing Vision with OpenCV

C++ NAOqi SDK provides some OpenCV libraries however It's recommended to use system OpenCV in order to debug. Primarily, because image windows are not displayed with Toolchain OpenCV Version.

3.1 Removing OpenCV from the NAOqi SDK

Once you have OpenCV installed for your system, you have to remove it from the SDK. Do not remove the CMake configuration files.

Remove all OpenCV libraries from the lib/ directory of the SDK. They will have the following format: libopencv_modulename.so, libcv.so, libhighui.so and libml.so

Remove the opencv folder from the include folder

Now clean your project, then run again

```
qibuild configure [-c mytoolchain]
qibuild make [-c mytoolchain]
```

Chapter 4

Remote Modules

Once compiled, you can run your module by:

```
$ ./mymodule --pip <robot_ip> --pport <robot_port>
```

Example:

```
$ ./mymodule --pip 148.226.110.95 --pport 9559
```

Chapter 5

References