AR8x application note

1. SCOPE

This guide providels details on how to remotely debug your C/C++ -projects on the AR8x debian based OS with Eclipse IDE.

1. Remote debug setup
   1. Prerequisities

Nordic ID Ar8x reader. The current setup of Nordic ID fixed readers does not support emulator debugging, so you’ll need acquire one device to be able to debug your projects on it.

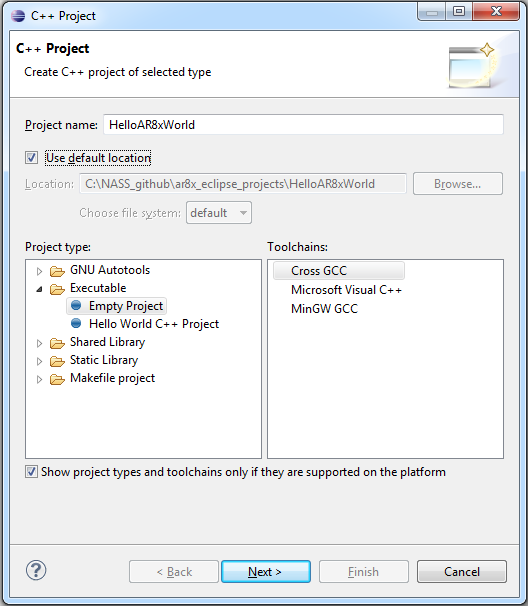
Download Eclipse IDE for C/C++ developers. Other packages might work too, but for this documentation we have used the Kepler version(4.3). You can acquire the IDE in question from this link for your OS: <http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/keplerr>

Download the Cross Compiling toolchain from github from address: <https://github.com/raspberrypi/tools> and unpack the files to your hard drive.

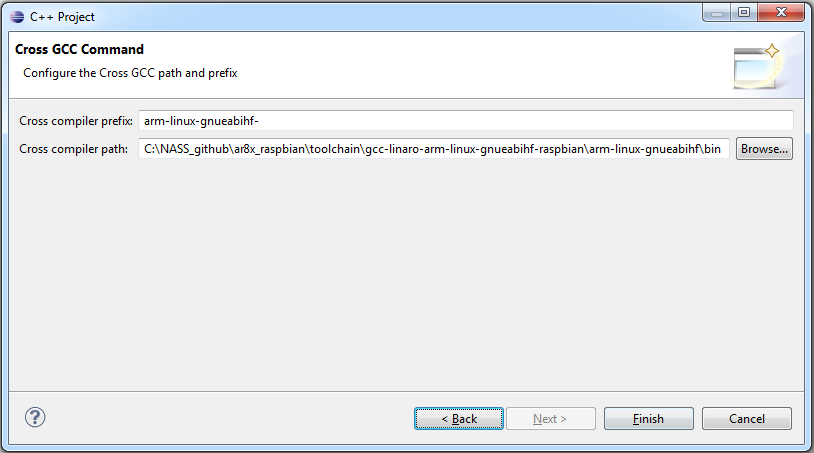
After you have successfully downloaded and installed the IDE & the toolchain, make sure that you have already read **AN001\_AR8x\_GettingStarted.docx** and gained access to the Web Config UI of your AR8x device. If you have not yet configured your device and accessed this view, please refer to documentation mentioned before proceeding.

* 1. Configuration

Start Eclipse IDE, and then create new C++ project from File->New->C++ Project. Then define the name for your project and select “Empty Project” and “Cross GCC” as the toolchain. Then click next.



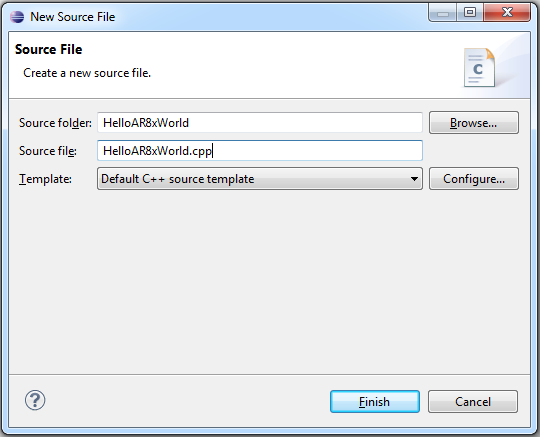
On the “Select Configurations” view, you can use the default settings and continue by clicking “Next”. After that you need to define the Cross Compile prefix and paths. For the path, use the location where you previously stored the Cross Compiling toolchain. Example below:



Note that Cross Compiler prefix needs to contain exactly the **arm-linux-gnueabihf-** in it. The CC path needs to be the absolute path the bin-folder located in the toolchain package i.e. **<installation folder>\toolchain\gcc-linaro-arm-linux-gnueabihf\arm-linux-gnueabihf\bin**. The toolchain contains also a x64 version of the compiler, so in case you are using a 64-bit OS, the path should be **<installation folder>\toolchain\gcc-linaro-arm-linux-gnueabihf-x64\arm-linux-gnueabihf\bin**. After the correct path is set up, you can click “Finish”.

* + 1. Hello AR8x World

On the “Project explorer, right click on the project name and click “New->Source file” and type in to “Source file” the name you wish to add for your C++ file. In this example we use “HelloAR8xWorld.cpp”. You can select the “Default C++ source template” as the default.



The source file will contain only the creation date and author information at this point, so let’s put something informational to it so we can test run it on the AR8x. Edit the file to contain something as shown in the below example:

/\*

\* HelloAR8xWorld.cpp

\*/

**#include** <iostream>

**using** **namespace** std;

**int** **main** (**void**)

{

cout << "Hello AR8x Development World !"<< **endl**;

**return** 0;

}

Then try out building the project with ctrl+b or from “Project->Build all”. If all is set up correctly, you should see something like this appearing on the Console:

15:50:37 \*\*\*\* Incremental Build of configuration Debug for project HelloAR8xWorld \*\*\*\*

make all

'Building target: HelloAR8xWorld'

'Invoking: Cross G++ Linker'

arm-linux-gnueabihf-g++ -o "*HelloAR8xWorld" ./HelloAR8xWorld.o*

*'Finished building target: HelloAR8xWorld'*

' '

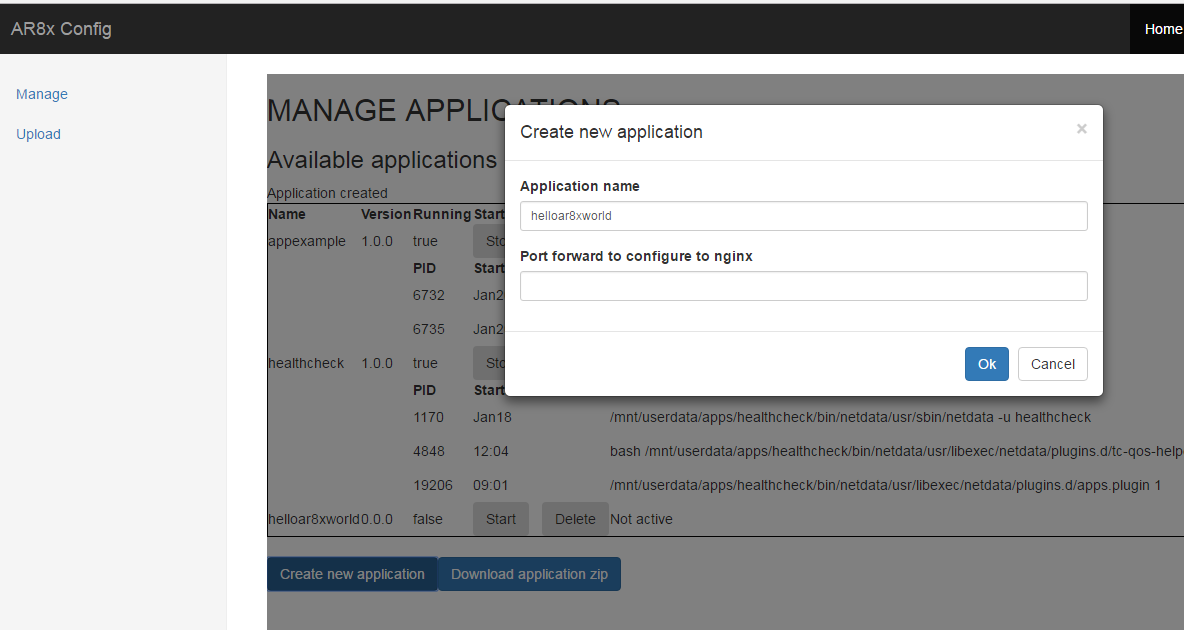
15:50:37 Build Finished (took 321ms)

If errors do however appear, they are usually due to invalid prefix and/or cross compiler paths. You can modify them by opening the project properties by clicking its name in the “Project explorer” and by selecting “Properties” from the drop-down menu. On the Properties you can find the mentioned setting from “C/C++ Build->Settings”, where on the “Tool settings” tab you should see the Prefix and Path. Correct them and try rebuilding.

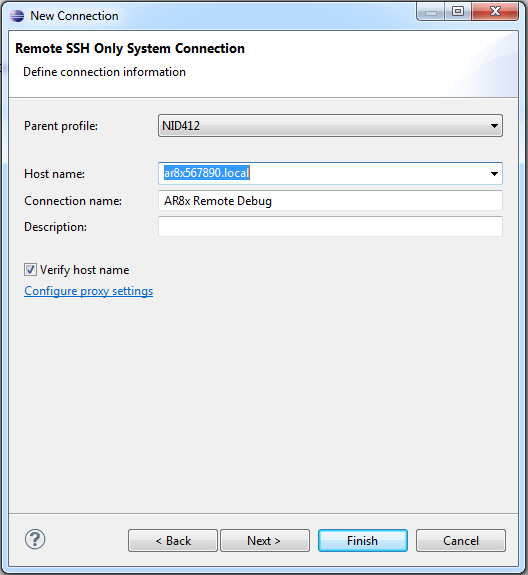
* + 1. dEbug connection to AR8x

**<EDIT THIS>**Once compiling the project works, we can move further on to remote debugging settings. If you have successfully powered your device up and connected to it as shown on AN001\_AR8x\_GettingStarted.docx, open up your web browser and open the Web Config UI of your AR8x device. On the WebConfig, open “System” and click “Access”. Find the “ Developer tools”-title and from below it, enable SSH and GDB. After that click “Save”. After this the IDE should be able to open a SSH connection to the device and start debugging the project in question. **</EDIT THIS>**

Before running the project, you’ll need to create “blank” project from Applications->Manage->Add blank project. Once you have given the project a name, it will automatically create an username and a password for the application. The password & username combination is applicationname:applicationname. *\*Note that the platform does not allow SSH for any users by default, so these login details are meant only for debugging purposes. The SSH will be disabled once the device is restarted again.*

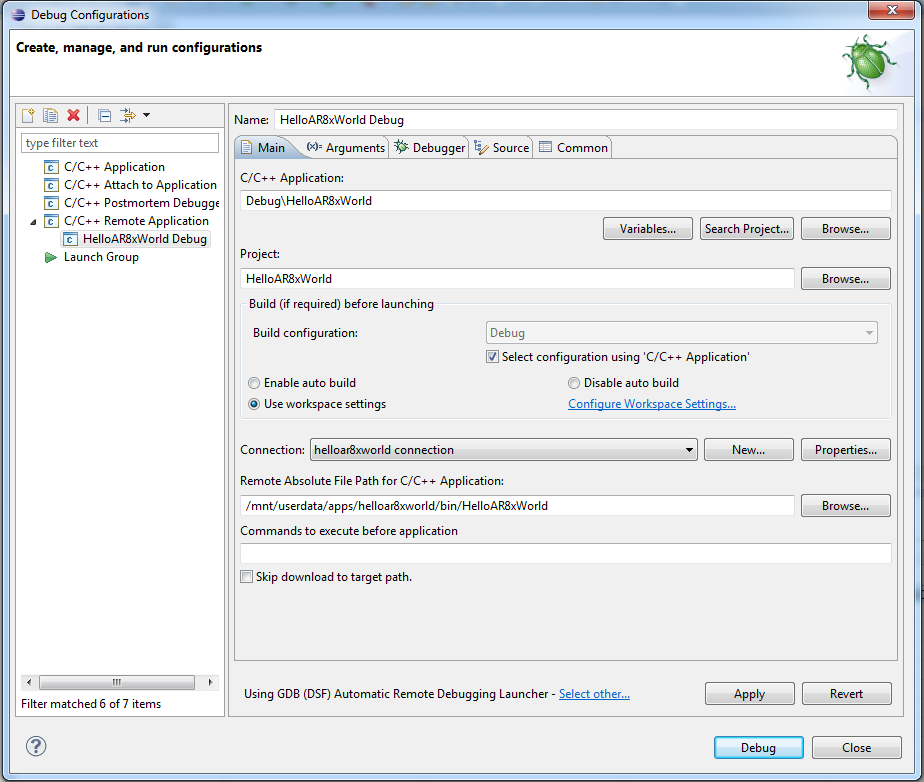


Then move back to Eclipse IDE which has your project open. Click “Window->Show View->Other”. From this view, select “Remote Systems->Remote Systems Details” and click Ok. Right-click on the “Remote system details” and select “New->Connection..”. From the pop-up, select SSH Only and click next. At this step you should already know the hostname of your device i.e. “ar8x567890.local” or similar. Input that as the “Host name” to the connection details. If you wish you, can also add your own name for the connection.

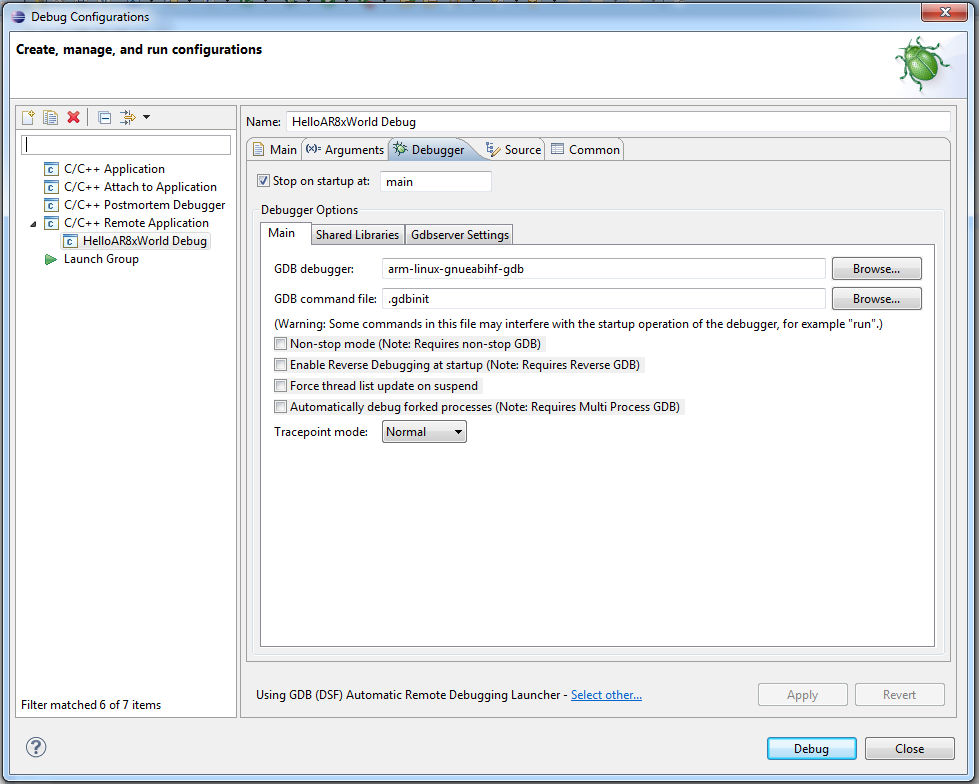


After the above you can click “Finish”, or check the Sftp files, SSH shell and SSH terminal configurations on the next pages. This however is not needed, since the connection should work with the default configuration.

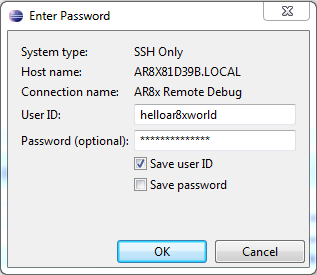
One more thing to setup is the define the remote paths to be used for starting your application. Open “Run->Debug Configurations..”. You should see under “C/C++ Remote Application” the name of the project i.e. “HelloAr8xWorld Remote Debug”. If that does not exist, right-click the “C/C++ Remote Application” and select New to create it. From that view, look for “Remote Absolute File Path for C/C++ Application:”. On that setting you need to add the absolute path to your previously created blank project. The contents for it will be something like **/usr/data/apps/<name of blank project>/bin** or **/mnt/userdata/apps/<name of blank project>/bin**. This needs to be added correctly, since the user(created for the blank project) you are using to access the platform has read/write/execute rights only to this application folder. You can also select the folder by clicking “Browse...” and selecting “AR8x Remote Debug” as the connection. which will open a connection to your device. The username and password will be requested at this point if you have not yet connected to the platform. Once the path is set up check also that on the “Main”-tab the “Connection is set to “AR8x Remote Debug”(or the name you have chosen for the connection).



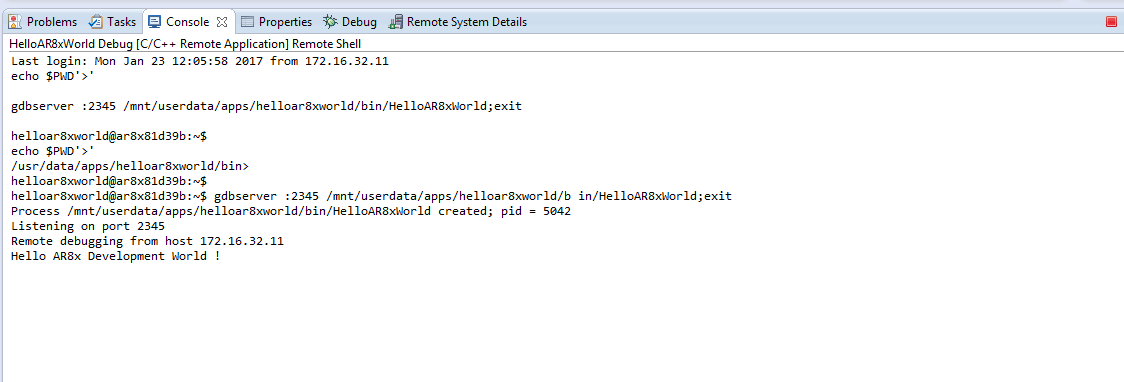
Select also correct debugger on the “Debugger”-tabpage. By default Eclipse will use gdb as the debugger. You however need to change it to **arm-linux-gnueabihf-gdb**.



Once everything is set up, you should be able to start debugging your project on the AR8x. Click first “Apply on the “Debug configurations” to save the settings. Then you can click “Debug” and you should see on the Console the output from your AR8x console and the execution should stop at the breakpoint in your main-function. Note that when your IDE connects to the platform, it will prompt for the username and password if a connection has not yet been established. At this point you can enter the appname:appname details for the connection.



After this you should seen an output similar to this on your IDE:



* 1. Additional libraries

Since the installation of the external applications to the platform has been restricted to signed packages, you should take a look at **AN002\_AR8x\_appinterface.docx** in case you have not yet gotten familiar with it.

Basically you can place “additional” libraries directly to the home-folder of the app i.e. **/bin** so that the application can load and find them, but if you wish to keep things a bit more organized, you can also place them in to the **/lib** –folder. This folder is included to the LD\_LIBRARY\_PATH when the application is started through the WebUI or through the autostart-feature of the CoreService.

During development the above does not most likely matter, but it’s rather a thing to keep in mind while creating your project and using external libraries. For example let’s say that your application uses the NurApi to command the internal UHF reader of your AR8x device and you wish to use the dynamically linked library instead of the static version. In this scenario you need to add the library in question to the **/lib**-folder on the final production package. During development it should be enough that your IDE deploys the libraries in question to the folder where the app is being debugged(or that you have copied them manually).

* 1. OTHER RELEVANT

This document instructs you how to debug your applications remotely on the AR8x. For constructing your production package of the application, please refer to the **AN002\_AR8x\_appinterface.docx**.