Developing using C on iMX Developer's Kit

Copyright 2021 © Embedded Artists AB

Developing using C on iMX Developer's Kits



Embedded Artists AB

Rundelsgatan 14 SE-211 36 Malmö Sweden

http://www.EmbeddedArtists.com

Copyright 2021 © Embedded Artists AB. All rights reserved.

No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Embedded Artists AB.

Disclaimer

Embedded Artists AB makes no representation or warranties with respect to the contents hereof and specifically disclaim any implied warranties or merchantability or fitness for any particular purpose. Information in this publication is subject to change without notice and does not represent a commitment on the part of Embedded Artists AB.

Feedback

We appreciate any feedback you may have for improvements on this document. Send your comments by using the contact form: <u>www.embeddedartists.com/contact-us</u>.

Trademarks

All brand and product names mentioned herein are trademarks, services marks, registered trademarks, or registered service marks of their respective owners and should be treated as such.

Table of Contents

1	Document Revision History 4
2	Introduction 5
2.1	Conventions
3	Getting started 6
3.1	Install toolchain6
3.2	Hello world7
3.3	Run the application on target7
4	Eclipse9
4 4.1	Eclipse
4 4.1 4.2	Eclipse 9 Updates to the Yocto image 9 Install Eclipse 9
4 4.1 4.2 4.3	Eclipse 9 Updates to the Yocto image 9 Install Eclipse 9 Create and configure a project 10
4 4.1 4.2 4.3 4.4	Eclipse 9 Updates to the Yocto image 9 Install Eclipse 9 Create and configure a project 10 Run the application on target 19
4 4.1 4.2 4.3 4.4 4.5	Eclipse9Updates to the Yocto image9Install Eclipse9Create and configure a project10Run the application on target19Debug the application29
4 4.1 4.2 4.3 4.4 4.5 5	Eclipse9Updates to the Yocto image9Install Eclipse9Create and configure a project10Run the application on target19Debug the application29Troubleshooting34

1 Document Revision History

Revision	Date	Description
А	2017-01-10	First release
В	2017-02-15	Added section 4.1 Updates to the Yocto image
С	2021-05-21	Added link to prebuilt toolchain for 64-bit architectures in section 3.1

2 Introduction

When developing applications for Linux you have a large selection of programming languages, editors, development environments, libraries and toolchains to choose from. This document will provide you with instructions for how to get started with application development using the C programming language.

This document is not a course in C programming or Embedded Linux application development. Instead, it will guide you in setting up the tools that exist for building your first "Hello world" application.

If you have never worked with Embedded Linux a recommended course is *bootlin's* "Embedded Linux training". The slides are available for download on their site:

https://bootlin.com/training/embedded-linux/

Additional documentation you might need:

- The Getting Started document for the board you are using.
- The Working with Yocto document

2.1 Conventions

A number of conventions have been used throughout to help the reader better understand the content of the document.

Constant width text - is used for file system paths and command, utility and tool names.

```
$ This field illustrates user input in a terminal running on the
development workstation, i.e., on the workstation where you edit,
configure and build Linux
```

This field illustrates user input on the target hardware, i.e., input given to the terminal attached to the COM Board

This field is used to illustrate example code or excerpt from a document.

This field is used to highlight important information

3 Getting started

The instructions in this section have been tested on a virtual machine running **lubuntu 16.04**. The document "Working with Yocto to build Linux" has a chapter that explains how to create a VMware based virtual machine running lubuntu.

If you are an experienced Linux user it shouldn't be a problem using another Linux distribution with the instructions below as a guideline.

3.1 Install toolchain

To be able to build an application that will run on Embedded Artists iMX based COM boards a toolchain is needed. The toolchain contains cross compiler, linker, and needed libraries.

The toolchain can be built in Yocto using the **meta-toolchain** image. See the document "Working with Yocto to build Linux" for more information about Yocto.

\$ bitbake meta-toolchain

The build will result in a file located at <build directory>/tmp/deploy/sdk. The exact name of the file depends on several parameters, but in our example, it is called:

```
fsl-imx-fb-glibc-x86_64-meta-toolchain-cortexa9hf-vfp-neon-
toolchain-4.1.15-1.2.0.sh
```

Part of file name	Description
fsl-imx-fb	The distribution (DISTRO parameter) used when initializing the build.
x86_64	Architecture of the host computer. In this example a 64-bit Intel x86 platform
4.1.15-1.2.0	BSP version

It is recommended to build this toolchain on your host computer where you will do the development, but if you have a 64-bit Intel x86 based host computer you can download a pre-built version from <u>imx.embeddedartsits.com</u>.

For iMX6 and iMX7 based boards (32-bit architecture)

\$ wget imx.embeddedartists.com/common/fsl-imx-fb-glibc-x86_64meta-toolchain-cortexa9hf-vfp-neon-toolchain-4.1.15-1.2.0.sh

For iMX8 based boards (64-bit architecture)

```
$ wget imx.embeddedartists.com/common/fsl-imx-wayland-glibc-
x86 64-meta-toolchain-aarch64-5.4-zeus.sh
```

Install the toolchain. It is recommended to use the default settings (such as installation path) when installing. Replace <installation file> below with the name of the file you have downloaded or built.

```
$ chmod a+x <installation file>
$ sudo ./<installation file>
```

3.2 Hello world

To test the toolchain and make sure everything is working a simple "Hello world" application will be developed.

- 1. Open a terminal application on your host computer (the host computer in our example is lubuntu 16.04)
- Setup the toolchain environment by running the source command below. A file was installed together with the toolchain that contains all environment variables needed to be setup. The instructions below use the default installation path.

```
$ source /opt/fsl-imx-fb/4.1.15-1.2.0/environment-setup-
cortexa9hf-vfp-neon-poky-linux-gnueabi
```

3. You can verify that the environment variables have been correctly setup by running the command below. This command shows the version of the GCC compiler.

```
$ $CC --version
arm-poky-linux-gnueabi-gcc (GCC) 5.2.0
```

4. Create the "Hello world" application using a text editor. In this example we are using nano. Create the file and copy the content of the example below to that file.

```
$ cd ~
$ mkdir hello_app
$ cd hello_app
$ nano hello.c
```

- 5. Save the file (in nano use Ctrl+X followed by Y and Enter)
- 6. From the terminal where you setup the toolchain environment compile the application. We are giving the application the name "hello" using the –o argument to the compiler.

\$ \$CC -o hello hello.c

You will now have a file named "hello". Go to the next section for instructions of how to run this application on target.

3.3 Run the application on target

In the previous section the application was built. In this section the application will be copied to the target via a USB memory stick.

- 1. Connect a USB memory stick to your host computer and copy the file "hello" to that memory stick. No instructions are given here since it normally is only a drag-and-drop procedure.
- Unmount the USB memory stick from your host computer and insert it into the developer's kit. You should see output in the terminal connected to the target that looks similar to below.

```
usb 1-1.3: new high-speed USB device number 7 using ci_hdrc
usb-storage 1-1.3:1.0: USB Mass Storage device detected
scsi host4: usb-storage 1-1.3:1.0
scsi 4:0:0:0: Direct-Access SanDisk U3 Cruzer Micro 2.18 PQ:
0 ANSI: 2
scsi 4:0:0:1: CD-ROM SanDisk U3 Cruzer Micro 2.18 PQ:
0 ANSI: 2
sd 4:0:0:0: [sda] 8015505 512-byte logical blocks: (4.10 GB/3.82
GiB)
sd 4:0:0:0: [sda] Write Protect is off
sd 4:0:0:0: [sda] Write Protect is off
sd 4:0:0:0: [sda] No Caching mode page found
sd 4:0:0:0: [sda] Assuming drive cache: write through
sda: sda1
sd 4:0:0:0: [sda] Attached SCSI removable disk
```

3. The important part is the device name "sda1". Mount the USB memory stick.

mount /dev/sda1 /mnt

Copy the application to the target. It is assumed that the file was copied to the root of the USB memory stick (in step 1 above). The '~' character means that we are copying the file to the home directory.

```
# cp /mnt/hello ~
```

5. Run the application

```
# cd ~
# ./hello
Hello world
```

 If you get a "permission denied" message instead of "Hello world" add execution permissions to the application and then run it again

chmod a+x hello

4 Eclipse

Eclipse is a popular software development kit that can be used with many different programming languages. This chapter describes how to get started with Eclipse when developing C applications.

4.1 Updates to the Yocto image

The default Yocto images provided by Embedded Artists are missing some functionality needed when following the instructions in this chapter. More specifically it is a GDB server – needed for debugging and a SFTP server – needed when downloading an application to target that are missing.

The servers can be added by modifying the local.conf file in your build. See the document "Working with Yocto to build Linux" for more details about building images.

1. Open local.conf. Replace <build dir> with your build directory.

```
$ nano <build dir>/conf/local.conf
```

2. Find the IMAGE INSTALL append variable and add the lines below.

```
gdbserver \
openssh-sftp-server \
```

- Save the file and exit the editor: CTRL+X followed by Y and Enter.
- 4. Now build your image. In this example we are using a "core-image-base" build, but replace this with the image you are building.

```
$ bitbake core-image-base
```

When the image has been built don't forget to deploy the image on the target. For more information see the "Working with Yocto" document.

4.2 Install Eclipse

If you haven't already got Eclipse on your host computer follow these instructions to install it. Please note that we are using lubuntu 16.04 when writing these instructions.

NOTE: It is Eclipse **3.8.1** that was installed when writing these instructions. If you have another version of Eclipse there could be minor differences.

1. Eclipse can be installed using apt-get

```
$ sudo apt-get install eclipse
```

- Answer Y to any question. It takes a couple of minutes to install eclipse.
- Since we are doing C development we also need to install CDT (C/C++ Development Tooling)

```
$ sudo apt-get install eclipse-cdt
```

 To be able to connect to the target from within Eclipse we are going to use a plugin called Remote System Explorer. \$ sudo apt-get install eclipse-rse

4.3 Create and configure a project

Start Eclipse

Start Eclipse either from the menu (normally under "Programming") or by writing eclipse in a terminal window.

\$ eclipse &

You will be asked to select a workspace as shown in Figure 1.

•	Workspace Launcher		- + ×
Select a wor	kspace		
Eclipse Platf Choose a wo	orm stores your projects in a folder called a workspace. rkspace folder to use for this session.		
Workspace:	/home/user/workspace	•	Browse
🗌 Use this a	as the default and do not ask again Cancel		ОК

Figure 1 - Select a workspace

Click Ok and if the workspace is new or has no projects you will be presented with the "Welcome screen" that can look as shown in Figure 2. Click on the "Workbench" button at the bottom of the screen.



Figure 2 - Eclipse welcome screen

Create a project

Create a new project by going to File \rightarrow New \rightarrow Project in the menu. Choose a "C Project" as shown in Figure 3 and then click "Next".

e Ne	w Project - + ×
Select a wizard Create a new C project	
Wizards:	
type filter text	۵
 Java Project Java Project from Existing And Plug-in Project General C/C++ C Project C++ Project Makefile Project with Existin CVS 	Buildfile g Code
? <back< td=""><td>Next > Cancel Finish</td></back<>	Next > Cancel Finish

Figure 3 - New Project wizard

Enter a project name ("hello" in this example). Select project type as an "Empty project" and set toolchain to be "Cross GCC" as shown in Figure 4.

C Pr	oject - + ×				
C Project	C Project ——				
Create C project of selected type					
Project name: hello					
☑ Use <u>d</u> efault location					
Location: /home/andre/workspace/he	Browse				
Project type:	Toolchains:				
🕆 🗁 Executable	Cross GCC				
Empty Project	Linux GCC				
Hello World ANSI C Project					
Shared Library					
Static Library					
Makerile project					
Empty Project					
Show project types and toolchains of	only if they are supported on the platform				
? <back< td=""><td>Next > Cancel Finish</td></back<>	Next > Cancel Finish				

Figure 4 - Project name and type

Click "Next" and then just use the default settings in the "Select configurations" dialog as shown in Figure 5.

۲	C Project	- + ×
Select Configura	tions	
Select platforms	and configurations you wish to deploy on	
Project type: Toolchains: Configurations:	Executable Cross GCC	
🗹 🛞 Debug		Select all
🗹 🛞 Release		Deselect all
		Advanced settings
Use "Advanced s	ettings" button to edit project's properties.	
Additional confi Use "Manage cor	gurations can be added after project creation nfigurations" buttons either on toolbar or on	property pages.
?	< Back Next > Car	Finish

Figure 5 - Select configurations

When you are done click the "Finish" button as shown in Figure 6.

۲	C Project	- + ×
Cross GCC Command		
Configure the Cross G	CC path and prefix	
Cross compiler prefix:	arm-poky-linux-gnueabi-	
Cross compiler path:	i_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi	Browse
?	< Back Next > Cancel	Finish

Figure 6 - Cross compiler

Create the application

Create a new file by going to File \rightarrow New \rightarrow Source File in the menu. Enter a file name as shown in Figure 7 (hello.c in this example) and then click "Finish".

•	New Source File	- + ×
Source File		c
Create a new s	ource nie.	
Source folder:	hello	Browse
Source file:	hello.c	
Template:	Default C source template ‡	Configure
?	Cancel	Finish

Figure 7 - New source file

Copy the "Hello world" application (source code) from section 3.2 to the newly created file as shown in Figure 8.



Figure 8 - Hello world application

Configure the project

It is now time to configure the project. The path to "sysroot" is needed in several places so the first step is to create an environment variable specifying this path.

۲	rP	operties for hello			- + ×
type filter text	Environment			¢ •	>
 Resource Builders C/C++ Build Build Variables 	Configuration: Debu	ug [Active]	*	Manage Config	jurations
Environment	Environment variable	es to set			bbd
Logging	Variable	Value	Origin		Add
Settings	CWD	/home/andre/works	BUILD SYSTEM		Select
Tool Chain Editor	PATH	/opt/fsl-imx-fb/4.1.1!	BUILD SYSTEM		Edit
▼ C/C++ General	PWD	/home/andre/works	BUILD SYSTEM		Delete
 Code Analysis Documentati 		New variable		- + ×	Undefine
File Types Name:		SDKTARGETSYSRC	OT		ondernie
Formatter Value:		ortexa9hf-vfp-neo	n-poky-linux-gnueabi	Variables	
Indexer	o all configurations				
Language Ma	o all configurations				
Preprocessor Ca	ncel OK				
Project Referen					
Run/Debug Settings					-
·····,- ··· · · · · · · · · · · · · · ·					
	Appendivariables	to native environment			
		/ironment with specifie	a one		
			Rest	ore <u>D</u> efaults	Apply
?				ок	Cancel

Figure 9 - Sysroot as environment variable

A number of compiler options must be specified in order to correctly compile the application. Go to C/C++ Build \rightarrow Settings and then click on Cross GCC Compiler \rightarrow Miscellaneous. Add the line below to the "Other flags" field in this window as shown in Figure 10.

-march=armv7-a -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 -sysroot=\${SDKTARGETSYSROOT}

۲		Properties for hello - +	×
type filter text	Settings	(•
type filter text 3 Resource Builders C/C++ Build Build Variables Environment Logging Settings Tool Chain Editor C/C++ General Project References Run/Debug Settings	Settings Tool Settings PBuild Steps Settings	Build Artifact Binary Parsers Error Parsers Other flags c-fmessage-length=0-march=armv7-a -mfloat-abi=hard -mfpu=neon -mtune= Verbose (-v) Support ANSI programs (-ansi) Position Independent Code (fPIC)	
?		OK Cancel	

Figure 10 - Compiler options

Options must also be given to the linker. Go to C/C++ Build \rightarrow Settings and then Cross GCC Linker \rightarrow Miscellaneous. Add the line below to the "Linker flags" field as shown in Figure 11.

--sysroot=\${SDKTARGETSYSROOT} -mfloat-abi=hard



Figure 11 - Linker options

When creating the application you might have noticed that stdio.h was underlined (see Figure 8). Holding the mouse cursor above the question mark shows you that stdio.h cannot be found. We need to add a path to the "header files" to get rid of this warning. Go to C/C++ General \rightarrow Paths and Symbols and then add the directory f(SDKTARGETSYSROOT)/usr/include. Click the "Apply" button and then "OK".

۲		Properties for hello			- + ×
type filter text	Paths and Symbols			¢	• • -
 Resource Builders C/C++ Build Build Variables Environment 	Configuration: Debug [Active	ries 🕭 Library Paths 🖉 Source Location 🖻 References	:	Manage Co	nfigurations
Logging	Languages Inc	lude directories			Add
Settings Tool Chain Editor • C/C++ General	Assembly 🕒 S	\${SDKTARGETSYSROOT}/usr/include			Edit
 Code Analysis Documentation File Types 					Export
Formatter Indexer					Move Up
Language Mappings Paths and Symbols Preprocessor Include Project References Run/Debug Settings	① "Preprocessor Include Paths, ☑ Show built-in values	, Macros etc." property page may define additional entries			Move Down
	-1 mporcosculgam (to expo				
			Restore	e Defaults	Apply
?			(ок	Cancel

Figure 12 - Path to headers

Build the application

Now it is time to build the application. Right-click on the project ("hello") and then select "Build Project" as shown in Figure 13 below.

You can see the output of the build in the "Console window" as shown in Figure 14.

File Edit Sou	C/C++	- hello/hello.c - Eclipse Platform oiect Run Window Help	- + ×
] 🗈 🔹 🖬 🦷	🖹 🗞 🔻 🗞 🔻 🗟] 🖆 🕶 😂	▼	😰 💀 c/c++ 🕒 "
Project Expl	orer 🛛 🦳 🗖 🕼 hello.c 🕅	-	□ <u></u> M
 C hello S hello S Includes Debug B hello.c 	New Go Into Open in New Window Copy Paste Source Move Rename	<pre> an 3, 2017 ser Ctrl+C Ctrl+V Delete , char **argv) ello world\n"); F2</pre>	U J ⁴ Z ≷ ≷ ● ¥ U stdio.h • main(int, char**): int
		F5	
	Build Configurations Make Targets Index	, I Console ⊠ □ Properties	
 0° és	<u>R</u> un As <u>D</u> ebug As <u>P</u> rofile As Tgam Comp <u>a</u> re With Restore from Local History Configure	o inker i-gccsysroot=/opt/fsl-imx-fb/4.1.15-1.2.0/s get: hello ed (took 137ms)	ysroots/cortexa9hf-vfp-n

Figure 13 - Build Project



Figure 14 - Console output from a build

4.4 Run the application on target

The application is now located in your workspace under the Debug directory. For these instructions that would be in the directory /home/user/workspace/hello/Debug/. You can use the same instructions as in section 3.3 to copy the application to a USB memory stick and then to the target.

Another alternative is to use the plugin "Remote System Explorer", but first it must be configured.

Open the Remote System Explorer Perspective

Go to Window \rightarrow Open Perspective \rightarrow Other as shown in Figure 15.



Figure 15 - Change perspective

Select "Remote System Explorer" as shown in Figure 16.

•	Open Perspective	-	+	×
Hig C	/C++			
cus C	VS Repository Exploring			
🎋 D	ebug			
🐉 J	ava			
ر لي	ava Browsing			
ر ^د ه	ava Type Hierarchy			
1 P	lug-in Development			
🖽 R	emote System Explorer			
🔁 R	esource (default)			
≝ ⁰ π	eam Synchronizing			
				-
				_
	Cancel	ОК		

Figure 16 - Select Remote System Perspective

Create connection to remote system

Now it is time to configure the connection to the target. Click on the icon shown in Figure 17.

Figure 17 - Create connection to remote system

별 🛱 💀 🛃 🔻

🔲 Propertie 🛛 🔍 Remote S

Value

Property

•	New Connection	- + ×
Select Remote System Typ	pe	1
Any distribution of Linux		=0=
System type:		
type filter text		ً
🝷 🗁 General		
🔁 FTP Only		
🛆 Linux		
🖳 Local		
🖙 SSH Only		
🖙 Telnet Only (Experim	ental)	
unix Unix		
💐 Windows		
?	Back Next > Cancel	Finish

Select "Linux" as the remote system type as shown in Figure 18.

Figure 18 – Remote system type

....

Specify the IP address of the remote target and give it a description as shown in Figure 19. Please note that the IP address will most likely be different on your target.

You can get the IP address of the target by using the *ifconfig* command in a terminal attached to the target.

	New Connection	- + ×
note Linux System	Connection	
efine connection info	ormation	
arent profile:	vmlubuntu	*
Host name:	192.168.1.130	
Connection name:	192.168.1.130	
Description:	iMX Target	
Verify host name onfigure proxy settin	ngs	

Figure 19 - Host name (IP address)

Go through the wizard (click the "Next" button) and choose "ssh"-related settings as shown in Figure 20, Figure 21, Figure 22, and Figure 23.

le Ne	w Connection		- + ×
Files			
Define subsystem information			
Configuration	Properties		
□ dstore.files	Property	Value	
□ ftp.files			
✓ ssh.files			
Available Services			
A Ssh / Sftp File Service			
SSH Connector Service			
SSH Settings			
Description			
Work with files on remote system	s using the Secure Sh	ell (ssh) protocol.	
(?) < Back	Next >	Cancel	Finish
- Dutin			

Figure 20 - Subsystem information part 1

•	New Connection		- + ×
Processes Define subsystem information	I		
Configuration	Properties		
dstore.processes	Property	Value	
✓ processes.shell.linux			
Available Services			
A Shell Process Service			
Description			
This configuration allows you any contributed Shell subsyste	to work with processes	s on remote linux s	ystems using
? <ba< td=""><td>ck Next ></td><td>Cancel</td><td>Finish</td></ba<>	ck Next >	Cancel	Finish

Figure 21 - Subsystem information part 2

•	New Connection	- + ×
Shells		
Define subsystem information	n	
Configuration	Properties	
dstore.shells	Property	Value
☑ ssh.shells		
Available Services		
A Generic shell service		
👻 💐 SSH Connector Service		
SSH Settings		
Description		
Work with shells and comma protocol.	nds on remote systen	is using the Secure Shell (ssh)
? <e< td=""><td>ack Next ></td><td>Cancel Finish</td></e<>	ack Next >	Cancel Finish

Figure 22 - Subsystem information part 3

•	New	Connection		- + ×
Ssh Terminals				
Define subsystem info	ormation			
Configuration		Properties		
✓ ssh.terminals		Property	Value	
Available Services				
A SSH Terminal Ser	vice			
▼ [®] SSH Connector Se SSH Settings	ervice			
Description				
Work with terminals a protocol.	ind command	ls on remote syster	ms using the Secu	re Shell (ssh)
?	< Back	Next >	Cancel	Finish

Figure 23 - Subsystem information part 4

Click Finish. The last step before establishing the connection is to specify which user that should login. Right-click on the connection and select "Properties" as shown in Figure 24.



Figure 24 - Connection properties

Select "Host" and then change the "Default User ID" to "root" (or another user if you want to login with a different user) as shown in Figure 25.

NOTE: By default, the user "root" is not permitted to use an SSH connection. See section 5.1 how

	Properties for 1	92.168.1.130 -	+ ×
e filter text 🛛 🛛	Host	↓ ↓	•
nnector Services	Resource type:	Connection to remote system	
ost	Parent profile:	vmlubuntu	
	System type:	Linux	
	<u>H</u> ost name:	192.168.1.130	-
	Connection name:	192.168.1.130	
	D <u>e</u> fault User ID:	l [root]	
	Description:	iMX Target	
	Verify bost name		
	Configure provusati		
	configure proxy setti	iys	
	Default encoding		
	Note: This setting ca	n only be changed when no subsystem is connect	ed
	Default from rem	ote system	
	O Other: UTE-8		

Figure 25 – User ID

Connect to the target by right-click and select "Connect" as shown in Figure 26.

(200) · · · · · · ·	v v · · v ·		
Remote Syste	🛚 📽 Team 🗖 🗖	🖻 hello.c 🛱	
Local Cal Cal	ls <u>N</u> ew <u>Go</u> Into <u>G</u> oTo	<pre></pre>	o.c Autho <sto (int retu</sto
Ssh Termir	In the second secon		
	🔊 Re <u>f</u> resh	F5	
	 ৄ Rename ➢ Delete Copy ↔ Move Export Import 	F2 Delete	stem I
Propertie 🕱	<pre></pre>		
Property	C <u>o</u> nnect		
Connection st	Clear Passwords		
Default User I	P <u>r</u> operties	Alt+Enter	

Figure 26 - Connect to target

If you are using the user "root" when logging in the default password is "pass".

Enter Password - + ×				
System type:	Linux			
Host name:	192.168.1.130			
Connection name:	192.168.1.130			
User ID:	root			
Password (optional): ****				
Save user ID				
Save password				
C	ancel OK			

Figure 27 - Enter password

Copy application to target

Go to the location of the compiled application. You will find it under Local Files \rightarrow My Home \rightarrow workspace \rightarrow hello \rightarrow Debug as shown in Figure 28.

📲 Remote Syste 🛛 😤 - Team 🗖 🗖	le hello.c
🛓 🔕 (> -> @ 🖻 🚖 🌄	⊖/* * he
▼ 📑 Local	*
🝷 🔁 Local Files	*
▼ ♣ My Home	*/
🝷 🗁 workspace	#incl
🝷 🗁 hello	⊖int m
🝷 🗁 Debug	{
鳥 hello	
Image: Book and the second	}
🗟 hello.o	
🚡 makefile	
objects.mk	
🚡 sources.mk	
🚡 subdir.mk	
🖻 hello.c	
Constant	Remote
۱ 🗀 qt-sc	🔊 192.168
	root@imx6
	total 52
🗏 Propertie 🛛 🤻 Remote S	drwxr-xr-
🖪 静 💀 🛃 🎽	- rw
Property Value	- rw- r r -

Figure 28 - Local files

Right-click on the application file (hello) and select "Copy".

📲 Remote Syste	e 🛙	🗣 Team 🗖 🗖	li≥ hello.c ☎
🔹 🗿 🤃		@ 🕒 🔄 🏲	⊖/* * hello.c
* 🗁 works	pace		*
🝷 🗁 hello)		* Created o * Autho
* 🗁 Del	bug		*/
,📄 h	ello		#include <std< td=""></std<>
ii⊋ h	ello.	<u>G</u> o To	•
🗟 h	ello.	Open	(int
m 🔊	ake	Open With	print
 [0	biec		retur
la se	ourc	Re <u>f</u> resh	F5
	ubdi 🎚	🗆 Rena <u>m</u> e	F2
la hel	lo.c \$	📕 <u>D</u> elete	Delete
> Contraction (1998)	otes	🗎 <u>С</u> ору	
2017-0	1-03	✤ Mo <u>v</u> e	
	-fb-	Search	
		y <u>search</u>	tem D
all qeope	1150	Synchronize Cach	e
		Compare With	, 30 🖾
La Local She	us	Replace With	- com : - com :
		Despection	AltuEntor
Propertie 🕮		Propercies	Alt+Enter root
		🖬 🖆 🖪 🛃 🎽	-rw 1 root
Property	Valu		-rw-rr 1 root
Filter	valu		root@imx6qea.com:
EXCENSION	1		I root@imy6gea.com.

Figure 29 - Copy application

Go to the remote system and paste the application under Sftp Files \rightarrow My Home as shown in Figure 30.

	010 - 39° * 🕑 *	
] 🖗 🔻 🖓 💌 🏷	New	۶.
Bemote Syste	Go <u>I</u> nto	
E Kemote Syste	<u>G</u> o To	*
🔹 🐑 🗇	Open in New Windo Show in Table	w : d_on: J
	🖳 Moni <u>t</u> or	ithor: u
le hello.	🔊 Re <u>f</u> resh	F5
🕨 🗀 Remote	៉ Rena <u>m</u> e	F2
	X <u>D</u> elete	Delete Int argc
₩ Fsl-imx-ft	Сору	intf("H
at-opens	👔 <u>P</u> aste	turn 0;
▶ 붥 Root	⇔ Move	
🗔 Local Shells	① Move Up	
* 🕂 192.168.1.130		
🝷 🇞 Sftp Files	V More Down	
▶	P <u>r</u> operties	Alt+Enter
I → Root		
Shell Process	ses	📲 Remote System Detail
Ssh Shells		TI 102 169 1 120 S
Ssh Terminal	s	S- 192.108.1.150 ↔
		root@imx6qea.com:~# l
Propertie	🖳 Remote S 🛛 🗖 🗖	total 52 drwxr-xr-x 2 root roo

Figure 30 - Copy/paste application to target

Figure 31 shows how it looks when the application has been copied to the target.



Figure 31 - Application file on the target

Start the application

To start the application we first need to start an SSH terminal. Right-click on "Ssh Terminals" and select "Launch Terminal" as shown in Figure 32.



Alt+Enter

Figure 32 - Launch a terminal

22

Property

Port

Number of chi

Figure 33 shows the terminal window.

Properties



Set execution permissions on the application file

chmod a+x hello

Run the application

./hello

All of the above is shown in Figure 34.

1	📲 Remote System Details 🖉 Tasks 📮 Console 🖉 Terminals 🖾
	🥲 192.168.1.130 🕱
• •	root@imx6qea-com:~# ls -la total 52 drwxr-xr-x 2 root root 4096 Dec 20 01:00 . drwxr-xr-x 4 root root 4096 Nov 30 16:58 -rw 1 root root 77 Dec 20 00:06 .bash_history -rw-r 1 root root 51 Nov 30 13:20 .profile -rw-rr 1 root root 33688 Jan 4 2017 hello root@imx6qea-com:~# chmod a+x hello root@imx6qea-com:~# ./hello Hello world root@imx6qea-com:~#

Figure 34 - Run application on target

4.5 Debug the application

For more complicated applications it is really useful to being able to debug the application, that is, single step through the code and inspect variables. This section describes how to debug your application from Eclipse using GDB.

First create the GDB command file (.gdbint) in the project directory. You can do this by rightclicking on the project and then New \rightarrow File.

We need to set the path to the sysroot in this file in order for GDB to load shared libraries. The path should be the same as set in the SDKTARGETSYSROOT environment variable. Add the line below to the file.



Figure 35 - GDB command file

Go to Run \rightarrow Debug Configurations in the menu and then right-click on "C/C++ Remote Application" and select "New" as shown in Figure 36.

Debug Configurations			
Create, manage, and run configurations		Ť.	
Image: Second system Image: Second system Image: Secon	Configure launch settings from this dialog: Configure launch settings from this dialog: Press the 'New' button to create a configuration of the selected type. Press the 'Duplicate' button to copy the selected configuration. Press the 'Delete' button to remove the selected configuration. Press the 'Filter' button to configure filtering options. Configure an existing configuration by selecting it. Configure launch perspective settings from the 'Perspectives' preference page.		

Figure 36 - New debug configuration

On the "Main" tab the name of the project is specified as shown in Figure 37.

•	Debug Configurations	- + ×
Create, manage, and run configurations		Ť.
Image: Second system Image: Second system <th>Name: hello Debug Main</th> <th>Browse Browse \$</th>	Name: hello Debug Main	Browse Browse \$
Filter matched 14 of 14 items	Арру	
?	Close	Debug

Figure 37 - Debug configuration, Main tab

Go to the "Debugger" tab and set the path to the GDB debugger and GDB command file.

```
/opt/fsl-imx-fb/4.1.15-1.2.0/sysroots/x86_64-pokysdk-
linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gdb
```

/home/user/workspace/hello/.gdbinit

•	Debug Configurations	- + >
Create, manage, and run configurations		Ť.
Image: Second	Name: hello Debugger Main Main Stop on startup at: main Debugger Options Main Main Shared Libraries Connection GDB debugger: (/opt/fsl-imx-fb/4.1.15-1.2.0/sysroots/x86_64-pokysdk-lint) GDB command file: //opt/fsl-imx-fb/4.1.15-1.2.0/sysroots/x86_64-pokysdk-lint) GDB command file: //ome/user//workspace/hello/.gdbinit (Warning: Some commands in this file may interfere with the startup operation of debugger, for example "run".) Non-stop mode (Note: Requires non-stop GDB) Enable Reverse Debugging at startup (Note: Requires Reverse GDB) Force thread list update on suspend Automatically debug forked processes (Note: Requires Multi Process GDB) Tracepoint mode: Normal	Browse Browse the
Filter matched 14 of 14 items	Apply	Revert
?	Close	Debug

Figure 38 -Debug configuration, Debugger tab

Within the "Debugger" tab select "Connection". Enter the IP address of the target and port number as show in Figure 39. The port number is used in the next step and is by default set to 10000.

100		Debug Configurations	- + ×
i	Create, manage, and run configurations		TO.
	Image: Second Secon	Name: hello Debug	
2	Filter matched 14 of 14 items		Apply Revert
	?		Close Debug

Figure 39 - Connection settings for debug configuration

as shown in Figure 40.

	# gdbserver :10000 hello	
³ □ ▼	<pre> Remote System Details Tasks Console Terminals I 192.168.1.130 root@imx6qea.com:~# gdbserver :10000 hello Process hello created; pid = 961 Listening on port 10000 </pre>	
	۲۰ – ۲۰ – ۲۰ – ۲۰ – ۲۰ – ۲۰ – ۲۰ – ۲۰ –	_

Figure 40 - GDB Server

Now it is possible to start to debug the application. Click on the debug icon and then "hello Debug" as shown in Figure 41.



Figure 41 - Start a debug session

If you are asked to change perspective click the "Yes" button as shown Figure 42.



Figure 42 - Change perspective

A debug session is now started and will break at the main function as shown in Figure 43.

Debug - hello/hello.c -	Eclipse Platform		- + ×
File Edit Source Refactor Navigate Search Project Run Window He	elp		
] █ ▾ 🖩 📾 📾 🕸 ▾ O ▾ O₂ ▾] 🗶 🕪 II 🖷 🖊 3.] 레 ▾ N ▾ < 스 ▾ 스 ▾	• .e i+ ≅ .≋] @	🕭 🖋 🔻] 🦽	🗈 🏇 Debug 月 "
© Debug ⊠ ∰ # 1 → ▼ □ □	🕬= Variables 🕱 🔏 Br	eakpoints 🔐 Registers 🛽	Modules 🗖 🗖
🕆 🖸 hello Debug [C/C++ Remote Application]			
* 🎲 hello [961] [cores: 0]	Name	Туре	Value
* 🔐 Thread #1 961 [core: 0] (Suspended : Breakpoint)	⇔ argc	int	1
= main() at neuto.c:12 0x1042c	argv	char **	0x7efffda4
Image: Second state in the second			
<pre>#include <stdio.h> @ int main(int argc, char **argv) { printf("Hello world\n"); return 0; }</stdio.h></pre>			

Figure 43 - Debug session

5.1 Allow user "root" to use an SSH connection

By default, the user "root" is not permitted to login via an SSH connection. By following these instructions "root" will be permitted to login through an SSH connection. It is, however, not recommended to use on a final application, but during development it can be permitted.

1. Open the configuration file for the SSH server

nano /etc/ssh/sshd_config

2. Find the line that starts with #PermitRootLogin and remove the '#' (hash) character. If you cannot find this line just add it to the file (without the hash)

PermitRootLogin yes

- 3. Save the file and exit the editor (in nano it is Ctrl-X followed by Y and Enter).
- 4. Restart the SSH server

/etc/init.d/sshd restart