Embedded Imaging Experts since 1996

VC vision
components®

Programming Environment Setup

Software Installation, Hardware Setup and Communication for VC Z Series

Vision Components GmbH
Ottostr. 2
76275 Ettlingen
Germany
Phone: +49 (0) 7243 2167-0

WWW.VC-LINUX.COM
WWW.VISION-COMPONENTS.COM

Foreword and Disclaimer

This documentation has been prepared with most possible care. However Vision Components GmbH does not take any liability for possible errors. In the interest of progress, Vision Components GmbH reserves the right to perform technical changes without further notice.
Please notify **support@vision-components.com** if you become aware of any errors in this manual or if a certain topic requires more detailed documentation.
This manual is intended for information of Vision Component's customers only. Any publication of this document or parts thereof requires written permission by Vision Components GmbH.

Trademarks

Linux, the Tux logo, Vivado, Xilinx and Zynq, Windows XP, Total Commander, Tera Term, Motorola are registered Trademarks. All trademarks are the property of their respective owners.

The Light bulb highlights hints and ideas that may be helpful for a development.

This warning sign alerts of possible pitfalls to avoid. Please pay careful attention to sections marked with this sign.

Author: VC Support, **mailto:support@vision-comp.com**

Table of Contents

This guide is solely made for the VC Smart Cameras containing a *Xilinx Zynq* processor!
All users are strongly encouraged to read the complete document.

1 Manual Overview

Key concepts to program a VC camera system will be presented in this manual.
A full working environment is built up by following the step-by-step guide in a following section.
Aim is to get a running *Hello World!* program.

The appendix contains knowledge about compiling from shell and a section to enforce camera communication.

Further information about image acquisition and processing will be provided at the document named *Programming Tutorial* which completes the introduction to the programming of VC smart cameras.

This guide was written primarily for Windows users. However it is possible to set up the development environment under a Linux system. The differences in the installation **process will be highlighted in**

orange and indicated by a Linux logo   for an installation under Ubuntu Linux.

2 Camera Access

Only a minimum of information to use the eclipse IDE is provided here.
Deeper knowledge like IP address changing, autostarting applications etc. can be found in the appendix at the end of this document.

An Ethernet TCP/IP connection is used to communicate between the camera and the PC.
Files will be transferred over SCP or SFTP protocol via standard port 22.
Console access will be done via the SSH protocol also via standard port 22.

The initial and fallback network settings are as follows:

IP Address	**192.168.003.15**
Subnet mask	**255.255.255.0**

Due to these subnet mask settings, your PC *must* have at least one different IP address in the range of 192.168.3.[0–254]. Contact your system administrator for assistance. The initial login settings are as follows:

User Name	**user**
Password	**user**

⚠ NOTE: until further notice and to ensure full functionality use as User Name / Password: **root / root**.

The following program can be used to transfer files:
- *WinSCP*, free software (**http://winscp.net/eng/download.php**), from the selectable interfaces, we use the Commander Interface.

The following program can be used to establish a terminal connection:
- *TeraTerm*, free software (**http://en.sourceforge.jp/projects/ttssh2/releases/**).

If not already set up, install one program for each task.
We encourage you to choose *WinSCP* and *TeraTerm* at least for the first time, that our support can assist you if something is not working out-of-the-box!

The camera IP address is not limited to the IP range 192.168.3.XXX. Any IP address can be used as long as the camera and the PC work in the same subnet.

2.1 Ping Test

To test the availability of the camera you can ping for it:
   a. Connect the power and network cables to the camera, for clarity consult the hardware manual.
   b. Start a Shell on your machine by
      clicking the *Start button → All Programs → Accessories → Run*.
   c. Type the command *ping 192.168.3.15* and press enter.
On success it continuously reports *Reply from 192.168.3.15: bytes=32 time=…*, if you get the message *Request timed out* or *Destination host unreachable* check your network settings and if the camera is turned on and if you waited for at least failsafe 30 seconds for the camera to finish booting. Your firewall settings may also lead to unsuccessful connections. Ask your system administrator for assistance.

## 2.2 File Upload Test

Test file uploading by connecting to the camera IP using the SSH standard port 22. A writeable directory is, for example, */tmp/*. See the following image how it is done at *WinSCP*. You can upload any small file to test, here we used a simple text file named *test_file_transfer.txt*.
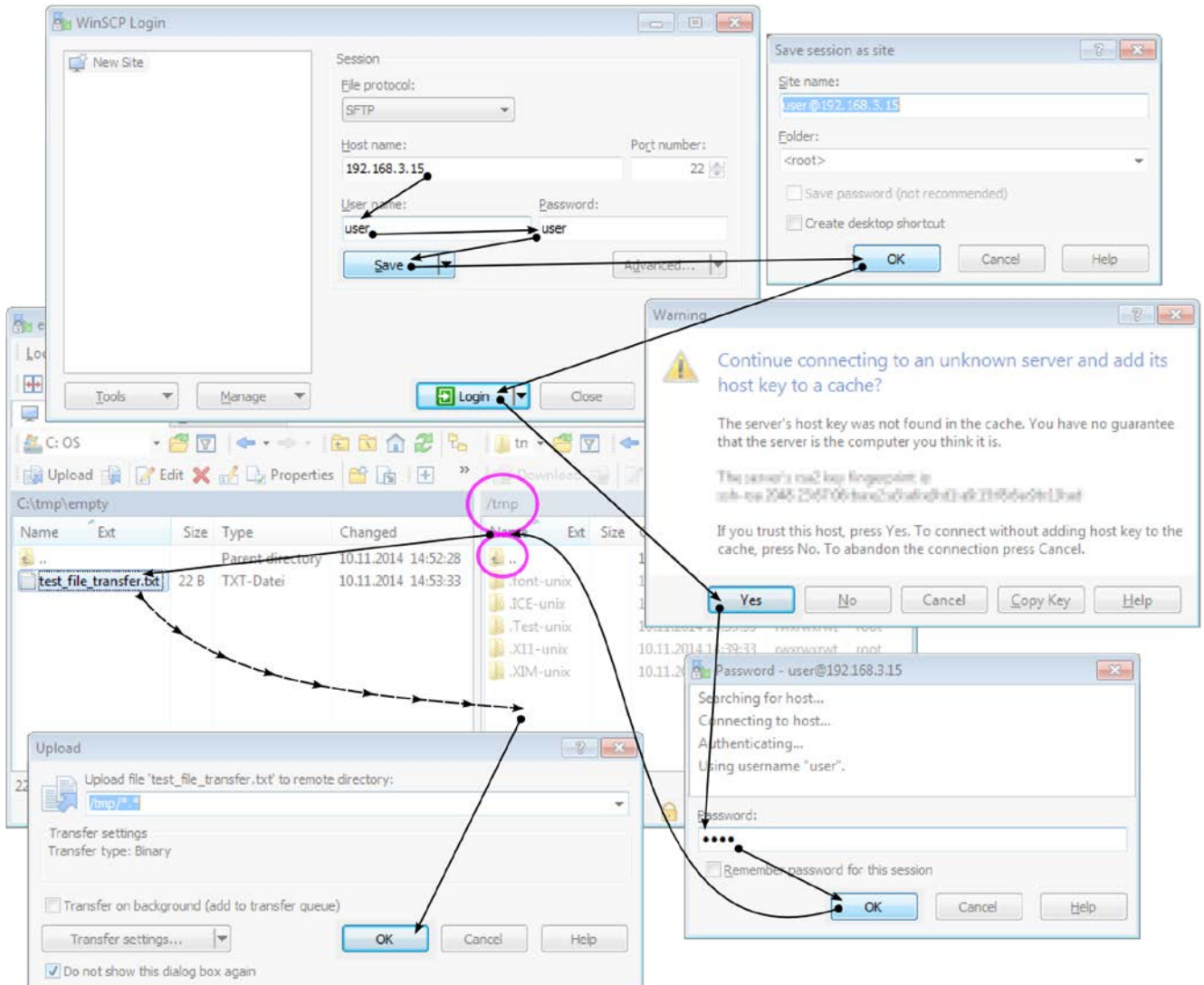
On **Linux / Ubuntu** use the "scp" command in the terminal.

2.3 Console Access Test

To gain shell access, connect to the camera via the SSH standard port 22. The following image shows how to set up a SSH connection via *Tera Term*. If you never used *Tera Term* before, a window may appear saying that no entry for the server "192.168.3.15" in your list of known hosts. Just confirm by pressing the *continue* button.
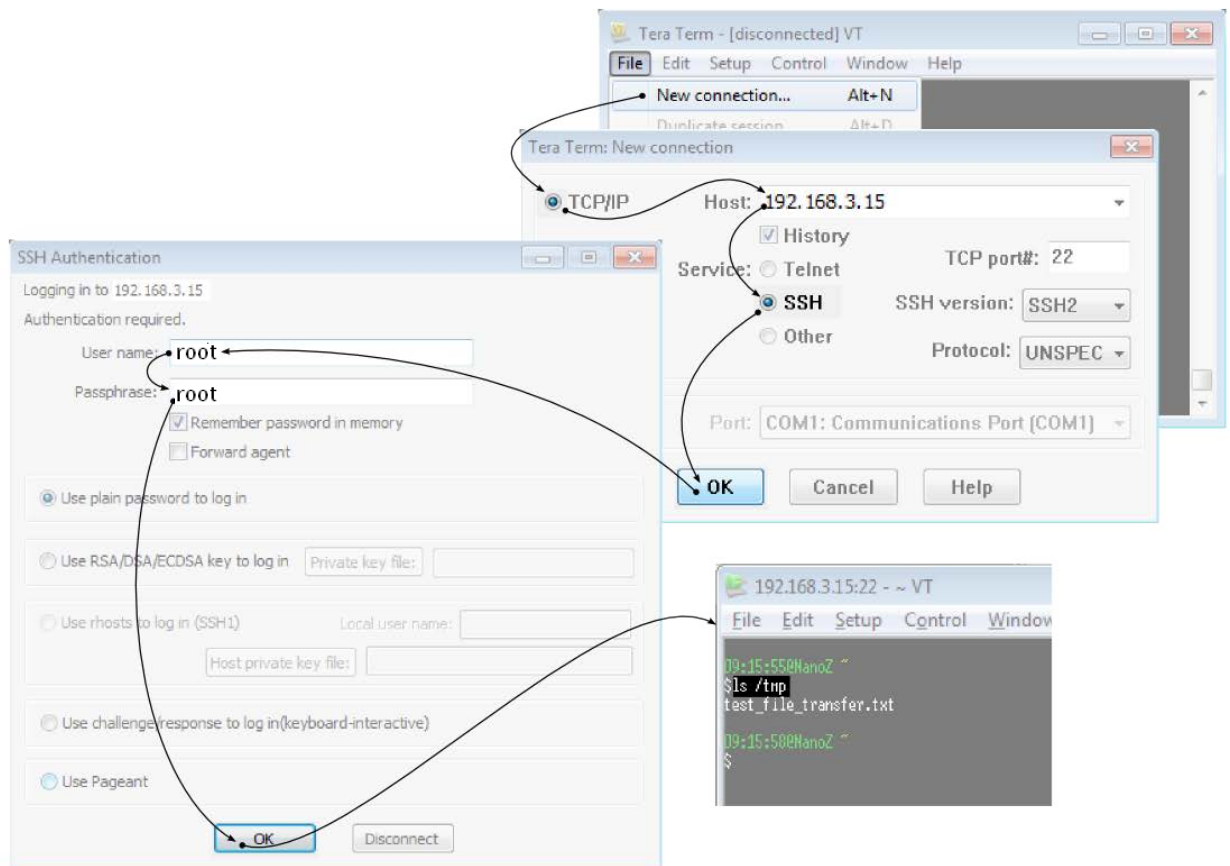
Linux / Ubuntu: run **ssh root@192.168.3.15** in the terminal.

3 Pieces of the Puzzle

There are required and optional software components to run a program at a VC Smart Camera with Xilinx Zynq processor.

3.1 Always involved Components

VC Linux

VC Linux is our bundle of a Linux kernel specifically tuned for our hardware setup and the root file system for the cameras including software packages.

libvclinux

libvclinux provides functions for image acquisition.

Linaro C Cross Compiler

Our tests showed that the Linaro compiler generates the fastest code at our platform. Hence we decided to declare it as our standard compiler and give to you instructions how to install and use it. Independent of your compiling operating system it will compile your program for our ARM Linux system. This behavior is called *cross compiling* if the compiling system is not of the same type as the executing platform.

3.2   Optional but Recommended Components

Eclipse IDE

The *Eclipse IDE* is a comfortable graphical development platform for programming Vision Components Smart Cameras. It is programmed in java. The IDE uses the GNU C cross compiler and debugger for ARM based chipsets which can be used to write and run programs on our camera system.
For most programs you are not bound to use the IDE, but it is **highly recommended**, since all support requests imply the application of the eclipse IDE and the Linaro compiler. No support is guaranteed for deviant solutions!

For an up to date list of approved *Eclipse IDE* versions, please refer to the *Support News* section in the *Support + Download* area of the VC website.

VC Image Processing Libraries

The VC Image Processing Libraries consist of an extensive range of image processing functions. They are utilized in various Industrial applications for more than 15 years and undergo continuous development.

A feature excerpt:
- Histogram Calculations, Projections, Pyramid Filters
- Kernel Filters, Sobel, Moving Averages, Correlations
- Run Length Image Processing Functions, Segmentation and Object Labelling
- Feature Calculations, for example, Area, Bounding Boxes, Fenet, Moments
- Contour Following
- Drawing Functions

To ensure compatibility and to ease the installation process the VCLIB setup is combined with the VC Linux setup in one installation.

4   Step-By-Step Guide for a Working Environment

Later, after installation and configuration of the Eclipse IDE, a demo project will be set up, the camera connection will be made, the program will be compiled, transferred and started.

Without using the Eclipse IDE you can also compile and link: Follow the same instructions to have the Compiler installed, information how to compile from the shell follow in an appendix.

We are sure that you gain **valuable** information on how the development components are interconnected by starting with bare programs and let you reenact setting the relevant parameters. Along the way we provide knowledge to you which will be good to know for your later projects.

4.1   VC Linux Setup on the target system (VC Z camera)

For compiling Vision Components uses libraries under the form of shared objects. In contrary to static libraries, these are dynamically linked at program launch. This is why they have to be present on the development platform (see next chapter) and also on the target system (in our case the VC Z camera).

This chapter explains the installation on the VC Z camera. The VC libraries are provided under the form of Debian packages, which are the standard way of installing software on Debian. For more information check **https://www.debian.org/doc/manuals/debian-faq/ch-pkg_basics.en.html**.

The camera (VCLinux OS and libraries) can be updated automatically using apt-get and VC's own Debian repository. To update the camera, execute the script **update-vclinux.sh** which can be downloaded here:
http://files.vision-components.com/VCLinux/update-vclinux.zip

Alternatively the libraries can be updated manually by following the steps 4.1.1 to 4.1.3.

4.1.1   Download

To be able to download the VC Software Package for the VC Z camera, you have to register at our website.

a.   Visit our website **http://www.vision-components.com**.
b.   Register for a *website account* by clicking *Login* followed by a
     click at *Not a member yet? click here to register*.
c.   The VC Linux software package then can be downloaded after log in at the following subsite:
     Service & Support → Download Center → Software ARM/VCLinux → VC Linux. Choose the
     newest version.
     No download available on the website yet! Temporary download link:
     http://files.vision-components.com/VCLinux/libvclinux_1.0.0_libvclib_4.0.2_debs.zip

4.1.2   Installation

The installation process needs you to be logged in as root.

    a. Extract the downloaded file after verifying its integrity.
    b. Upload the install files to the **/tmp** folder on the camera using for example WinSCP.
    c. From the camera shell, run "**dpkg –i /tmp/*.deb**".

4.1.3 Configuration

The configuration is done automatically by the Debian package manager.

The installed files are to be found under the /usr folder:

usr			
	lib		Shared objects files
	include		Include files
	share		
		doc	
		libvclib-doc	Libvclib documentation
		libvclinux-doc	Libvclinux documentation

You can check the versions of the installed packages by running the command "**dpkg –l | grep vc**":

```
08:09:24[root@VC-Z] ~
#dpkg -l | grep vc
rc  libavcodec56:armhf        6:11.4-1~deb8u1  armhf   Libav codec library
ii  libvcflib                 4.0.2            armhf   Library with Elementary Image Processing Functions (Shared Library)
ii  libvcflib-dev             4.0.2            armhf   Library with Elementary Image Processing Functions (Header Files and Static Library).
ii  libvcimgnet               0.18.0           armhf   Image Network Transfer Library for VC Smart Cameras (Shared Library)
ii  libvcimgnet-dev           0.18.0           armhf   Image Network Transfer Library for VC Smart Cameras (Header Files and Static Library)
ii  libvclib                  4.0.2            armhf   Image Processing Library for VC Smart Cameras (Documentation)
ii  libvclib-dev              4.0.2            armhf   Image Processing Library for VC Smart Cameras (Header Files and Static Library)
ii  libvclib-doc              4.0.2            all     Image Processing Library for VC Smart Cameras (Documentation)
ii  libvclinux                0.20.0           armhf   Image Acquisition Library for VC Smart Cameras (Shared Library)
ii  libvclinux-dev            0.20.0           armhf   Image Acquisition Library for VC Smart Cameras (Header Files and Static Library)
ii  libvclinux-doc            0.20.0           all     Image Acquisition Library for VC Smart Cameras (Documentation)
ii  linux-headers-3.14.51-vc-z 10             armhf   Linux kernel headers for 3.14.51-vc-z on armhf
ii  linux-image-3.14.51-vc-z  10               armhf   Linux kernel, version 3.14.51-vc-z
ii  vcgpio                    0.17.1-1         armhf   Example GPIO Manipulation for VC Cameras.
ii  vcimgnetsrv               0.17.1-1         armhf   Example Image Transfer Server for VC Cameras.
ii  vcinit                    1.0-1            all     VC init services
ii  vcio                      0.17.1-1         armhf   I/O Control for VC Cameras.
ii  vcperformance             0.17.1-1         armhf   Performance Measurement for VC Cameras.
ii  vctp                      0.17.1-1         armhf   Example Image Acquisition and Transfer for VC Cameras.
ii  vcver                     0.17.1-1         armhf   Version Information for VC Cameras.
```

The VCLinux system contains the following preinstalled tools:

vcgpio	Example GPIO Manipulation for VC Cameras
vcimgnetsrv	Example Image Transfer Server for VC Cameras
vcio	I/O Control for VC Cameras
vcperformance	Performance Measurement for VC Cameras
vctp	Example Image Acquisition and Transfer for VC Cameras
vcver	Version Information for VC Cameras

Updates for these tools are also available on our website. Direct link for the current version:
http://files.vision-components.com/VCLinux/vcapps_1.0.0_debs.zip

4.2   VC Linux Setup on the development platform (PC)

This step will copy the VC libraries for hardware access and documentation to a standard folder.

Linux / Ubuntu: a package will be available for Ubuntu. For now copy the files manually to */home/user/vc* for example.

4.2.1   Download

To be able to download the VC Software Package, you have to register at our website.

 a. Visit our website **http://www.vision-components.com**.
 b. Register for a *website account* by clicking *Login* followed by a
   click at *Not a member yet? click here to register*.
 c. The VC Linux software package then can be downloaded after log in at the following subsite:
   Service & Support → Download Center → Software ARM/VCLinux → *VC Linux*. Choose the
   newest version.
   No download available on the website yet! Temporary download link:
   http://files.vision-components.com/Libs/libvclinux_1.0.0_libvclib_4.0.2.zip

4.2.2   Installation

NOTE: the setup file is not available yet! The content of the zip file has to be copied manually to
C:\vc!

The installation process needs you to have administrator rights.
It goes without saying that you should have your system backed up, especially before updating.

 d. Extract the downloaded file after verifying its integrity.
 e. Execute the binary named *Setup_<target>_<libvclinux ver.>_<libvclib ver.>.exe*.
 f. To be able to proceed you need to read and accept the licence agreements.
 g. As Destination Directory it is recommended to choose the standard path *C:\vc*,
   so that you and our support have the same directory structure.

4.2.3   Configuration

Since the VC Linux software package only consists of passive parts like libraries, headers, documentation, etc. the main configuration is done at the Eclipse IDE.

4.2.4   Files and Directories

The following components are at the installation directory (for example ***C:\vc***):

vclinux		
	include	Header Files
	lib	Library Files
doc		
	libvclinux-doc	Camera Hardware Related Documentation
	libvclib-doc	Documentation of Functions for Image Processing and Acquisition

4.3   Linaro Compiler Setup

The compiler will be used to build your programs not only from the eclipse environment.

<div style="border: 2px solid orange;">

FOR LINUX / UBUNTU USERS

Download and install the current gcc package for armhf platform with:
```
apt-get install gcc-arm-linux-gnueabihf
```

Download and install the current g++ package for armhf platform with:
```
apt-get install g++-arm-linux-gnueabihf
```

Then proceed directly to chapter 4.3.3.

</div>

4.3.1   Download

   a.   Download the *Windows Installer* of the toolchain here:
      **http://files.vision-components.com/VCLinux/gcc-linaro-arm-linux-gnueabihf-
      4.9_win32.zip**

4.3.2   Installation

The installation process needs you to have administrator rights.
It goes without saying that you should have your system backed up, especially before updating.

   a.   Execute the binary after verifying its integrity.
   b.   To be able to proceed you need to read and accept their licence agreements.
   c.   As Destination Directory it is recommended to choose the standard path
      C:\Program Files (x86)\Linaro\gcc-linaro-arm-linux-gnueabihf-4.9-<Release Version Number>
      so that you and our support have the same directory structure.
   d.   Note the GCC Environment Settings Batch File Location at
      C:\Program Files (x86)\Linaro\gcc-linaro-arm-linux-gnueabihf-4.9-<Release Version Number>\bin\gccvar.bat.
   e.   Finish the setup dialog.

4.3.3   Configuration

The compiler, header, and library location is used later at the Eclipse IDE configuration phase.

4.3.4 Files and Directories

The following important components are at the installation directory:

bin	Compiler/Linker/Debugger Executables
arm-linux-gnueabihf	*...hf*: hard float ABI version
...	
libc	
...	
lib\arm-linux-gnueabihf	'*Shared*' Libraries
usr	
include	Header Files
lib\arm-linux-gnueabihf	'*Static*' Libraries

⚠ The suffix *hf* indicates the *hard float* build variant of the Application Binary Interface (ABI):
You cannot mix up object files or (shared or static) libraries built with different kinds of ABI.
The compiler option *–mfloat=hard* will set the type of choice later.
If a hard float application is linked statically, it should run at a soft float environment.

4.4 Eclipse IDE Setup

You can also build your program from shell (see Appendix A), but eclipse makes at least debugging a lot more comfortable.

Eclipse needs a *Java Runtime Environment* installed. If it is not already present at your system get it from the website **http://java.com**. We suppose this to be done before proceeding.

FOR LINUX / UBUNTU USERS

Download and install the Eclipse IDE package with:
`apt-get install eclipse-cdt-launch-remote`

If the Run menu does not contain the debug commands, go to menu **Window -> Customize perspective…**
Under the tab Command Groups Availability, check the following items:
- Breakpoints
- C/C++ Debug
- Debug
- Launch

Download and install the correct GDB package:
`apt-get install gdb-multiarch`

Then proceed directly to chapter 4.4.3.

4.4.1 Download

a. Visit the Eclipse website **http://www.eclipse.org/downloads**.
b. The appropriate Eclipse package is named *Eclipse IDE for C/C++ Developers*.
c. Download the package fitting the bit version of your system (32- or 64-bit).

Eclipse IDE for C/C++ Developers
175 MB   202,632 DOWNLOADS
An IDE for C/C++ developers with Mylyn integration.
Windows
32 bit | 64 bit
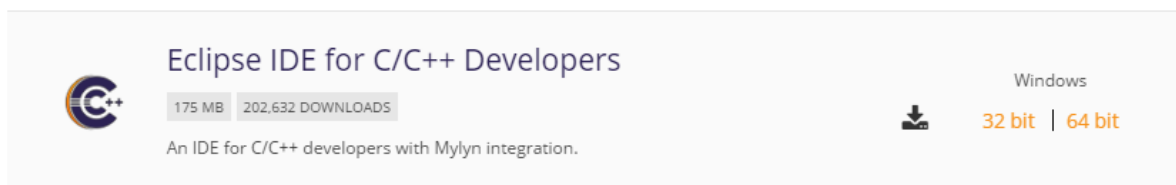
The Java Runtime Environment version and the Eclipse version should be both either 32-bit or 64-bit! A 32-bit Eclipse with a 64-bit Java for example will not work!

4.4.2 Installation

The installation process needs you to have administrator rights if you install it to the suggested folder. It goes without saying that you should have your system backed up, especially before updating.

a. Verify the integrity of the file.
b. To be able to proceed you need to read and accept the licence agreements for the package.
c. Extract the files of the archive. As Destination Directory it is recommended to choose the standard path *C:\Program Files* or *C:\Program Files (x86)* depending on the bit version downloaded which should lay the files to a directory named *eclipse*, so that you and our support have the same directory structure. It may be necessary to extract the files to a temporary folder first and then move the eclipse folder to the standard path.
d. You may want to add a link to the executable *C:\Program Files\eclipse\eclipse.exe* or *C:\Program Files (x86)\eclipse\eclipse.exe* to your start menu.

4.4.3 Configuration

At the first start of eclipse you have to select a *workspace* directory where eclipse will store your projects, we suggest the path *C:\vc\eclipse-workspace* (**Linux / Ubuntu:** */home/user/workspace*). After application a *Welcome* tab is displayed. It can be closed by just clicking the *X* symbol of the tab.

⚠ *Important* is the correct configuration of your Windows Internet Explorer proxy server settings! Ensure that any proxy server is bypassed for connections to the camera IP. You can get to the configuration panel by calling your *Windows Control Panel*, followed by *Network and Internet* and *Internet Options*. Then choose the Tab named *Connections*, click on the *LAN Settings* button and add the Camera IP at the *Do not use proxy server for adresses beginning with* field after clicking the *Advanced* button.
Also ensure that your firewall does not block any traffic to the camera IP, and your PC is in the same subnet as the camera, which means, that it has an IP address of the form *192.168.3.XXX*.

4.4.3.1   Adding a Camera as a Remote System to Eclipse

The following images show how to add a camera device as a remote system.
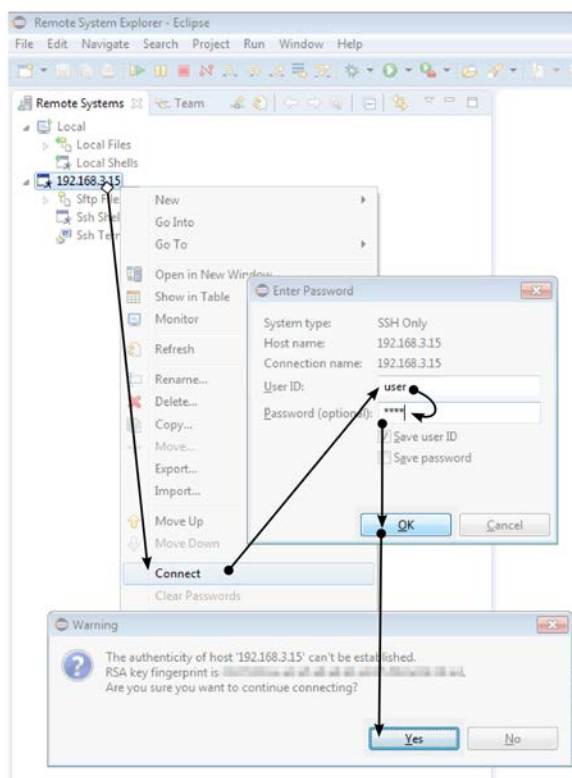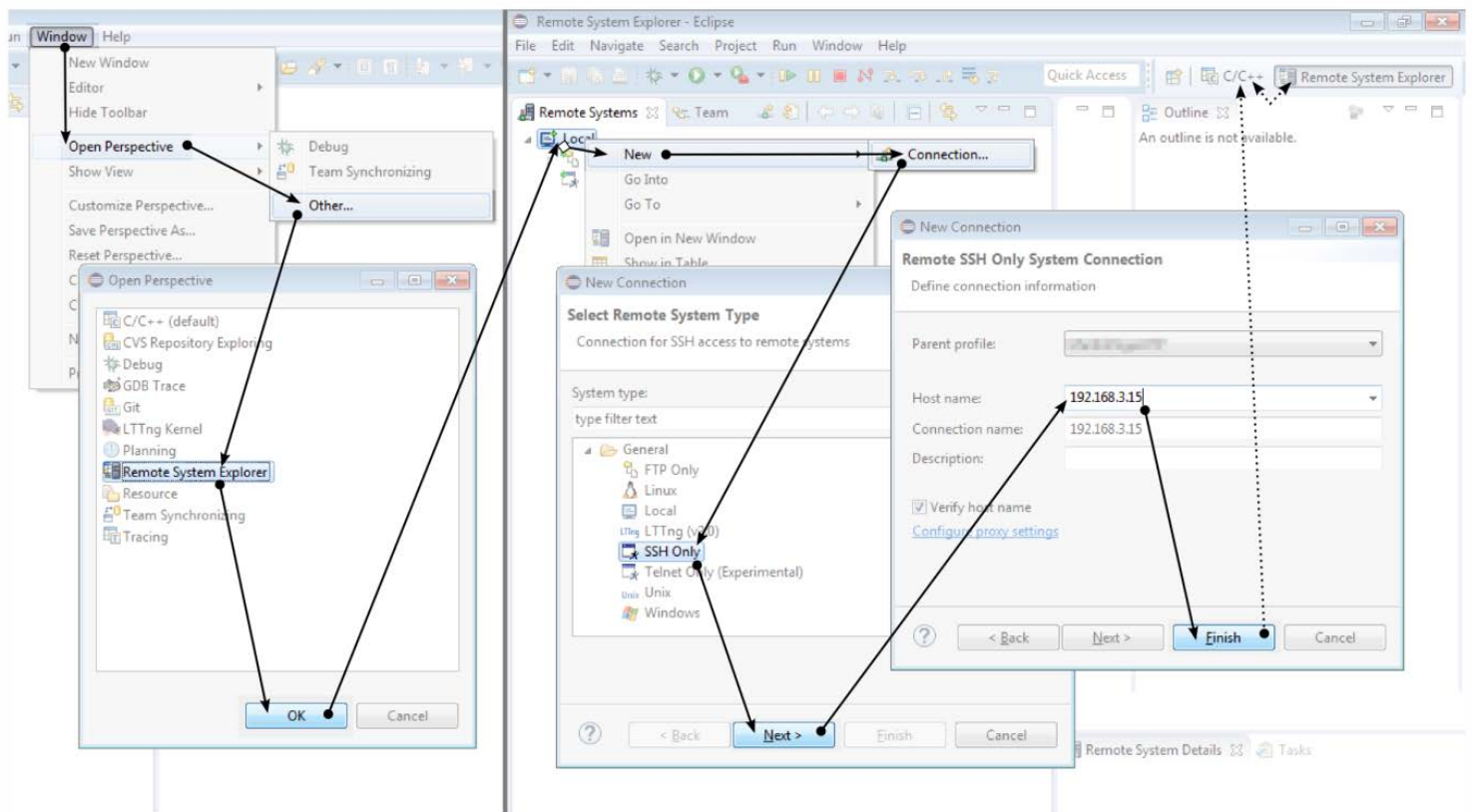
NOTE: until further notice please use as User Name / Password: **root / root**.

If the error message *Failed to connect sshd on "192.168.3.15:22"* is displayed, check your proxy server and/or your firewall settings (also the Network settings in Eclipse)!

4.4.3.2   Debugger Template Setup: Communication between the SDK and a VC Camera

The following Image describes the main setup of an environment used for debugging.
It is meant to be duplicated by you for each of your projects. The applied settings provide the target IP and information how to transfer data or access the console as well as the debugger used by the client machine.
The path of the GDB Debugger is given by the location of the Linaro GCC and its version number, for example: *C:\Program Files (x86)\Linaro\gcc-linaro-arm-linux-gnueabihf-4.9-2014.09\bin\arm-linux-gnueabihf-gdb.exe* (**Linux / Ubuntu:** */usr/bin*).

On **Linux / Ubuntu** the correct path and command is:
/usr/bin/gdb-multiarch --eval-command="set architecture arm"

Later at the *Hello World* example the project which builds the executable, as well as the executable path and filename on the target camera system, will be added to a duplicate of this template.
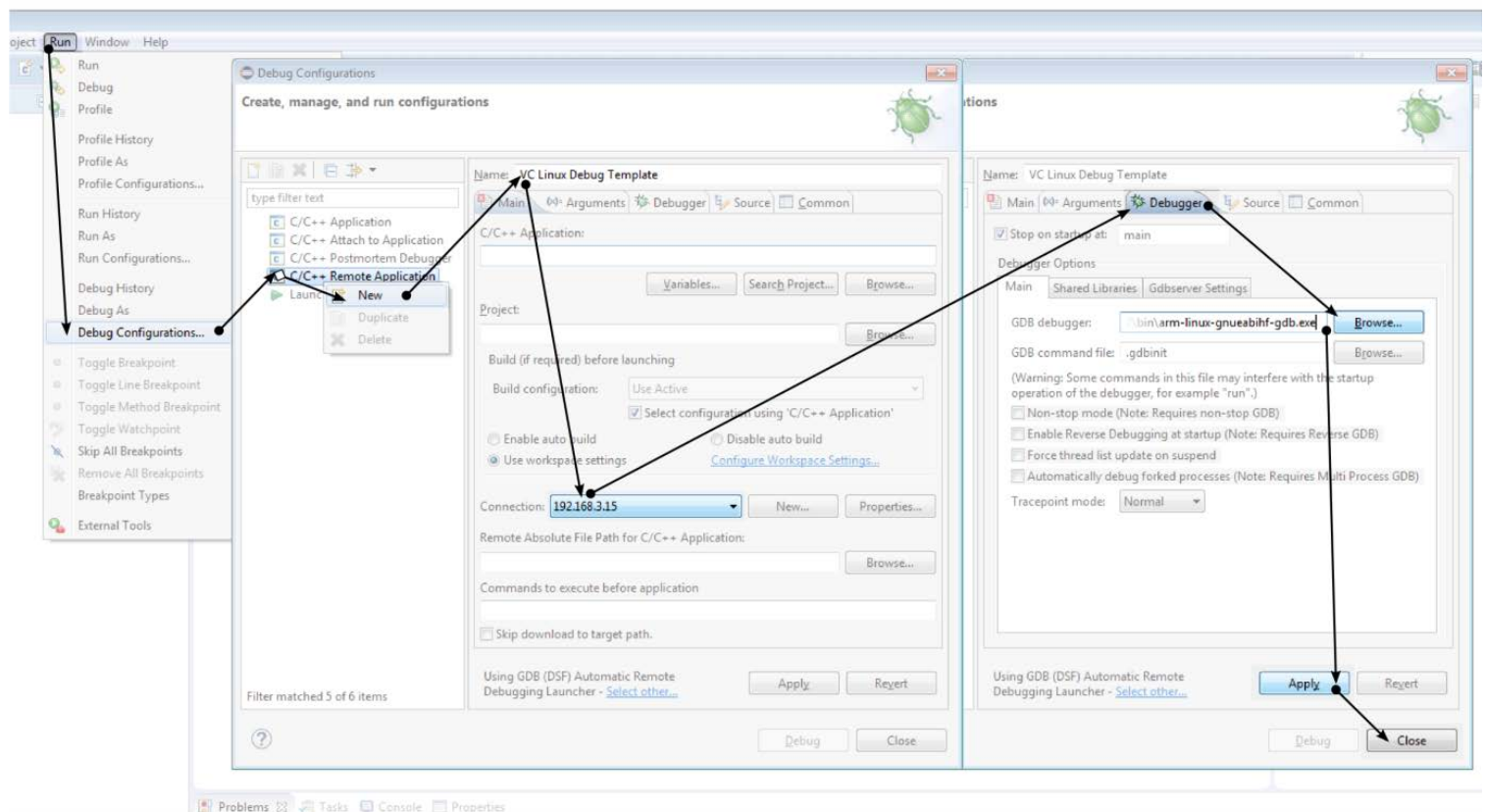
4.4.4   Files and Directories

The folder of the eclipse program contains nothing really of interest.

Configuration changes of eclipse and new plugins will be kept in a subdirectory named '*.eclipse*' at the user's home path *C:\Users\<User Name>* if the actual eclipse installation directory is not writeable (which is the case if you have installed it to the standard path *C:\Program Files\eclipse*).

The selected workspace folder contains a directory named '*.metadata*'. All workspace settings you configure will be kept in there (for example also the just generated Debug Configuration). Do not modify anything in the folder directly. You should also not copy the folder directly since this would not work correctly. If you want to, e.g. for sharing the settings with a colleague, then use the *File→Export→General→Preferences* dialogue at the Eclipse IDE.

Each new C project like the following *Hello World!* program add a folder to the workspace directory. The project directory includes the files '*.cproject*' and '.project'. The '*.cproject*' file contains C project settings like the compiler parameter or library paths.

4.5 *Hello World!* Program

4.5.1 Set Up a New C Project

The next image displays the steps to be done for generating a new C project.
Verify your project type is *Executable* and your toolchain is *Cross GCC.*
The cross compiler path is the path to our previously installed Linaro GCC. Control the path twice,
since a false path will lead to an *arm-linux-gnueabihf-gcc not found in PATH* build error.
If the IDE asks you to switch to the so-called *C/C++ Perspective* , accept it.
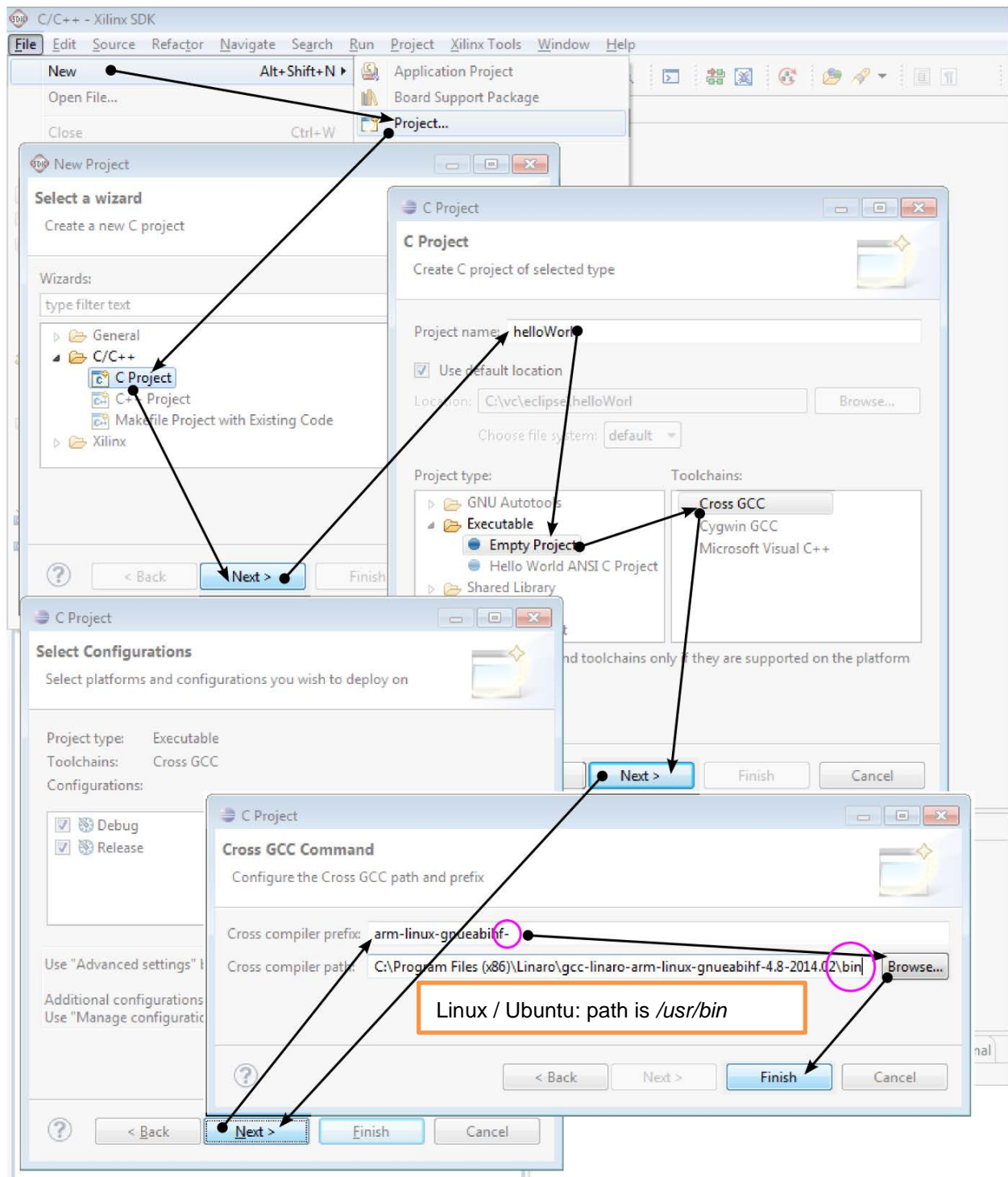
4.5.2   Using the Internal Builder

Ensure the internal builder is being used. This step is only for reasons to have a guaranteed working environment. If you know and use *make*, you may ignore this step.
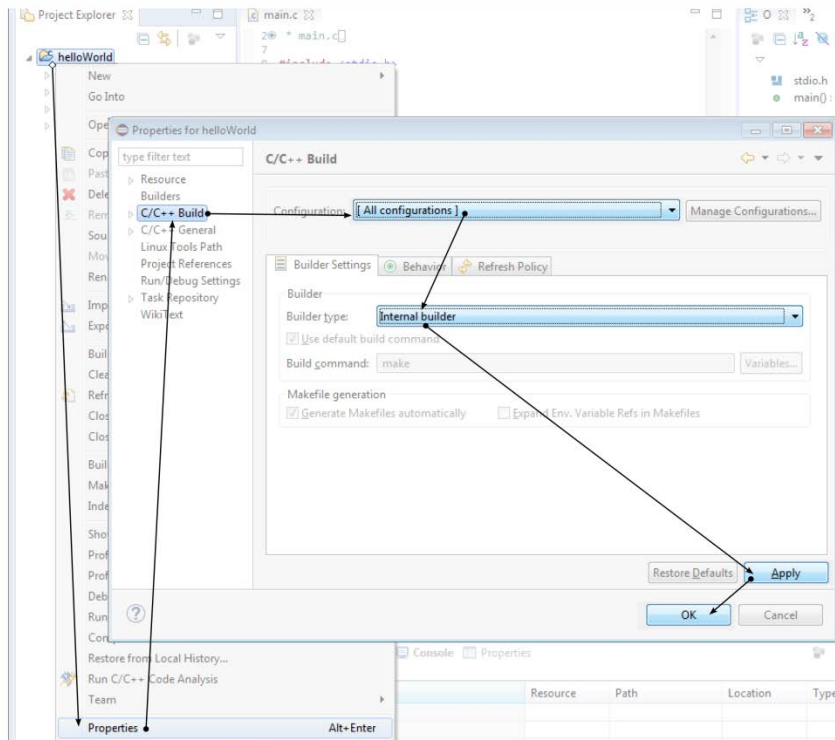
4.5.3   Adding a New *main.c*

Right click on the *helloWorld* Project Folder to open the menu. Don't forget to save the new file!
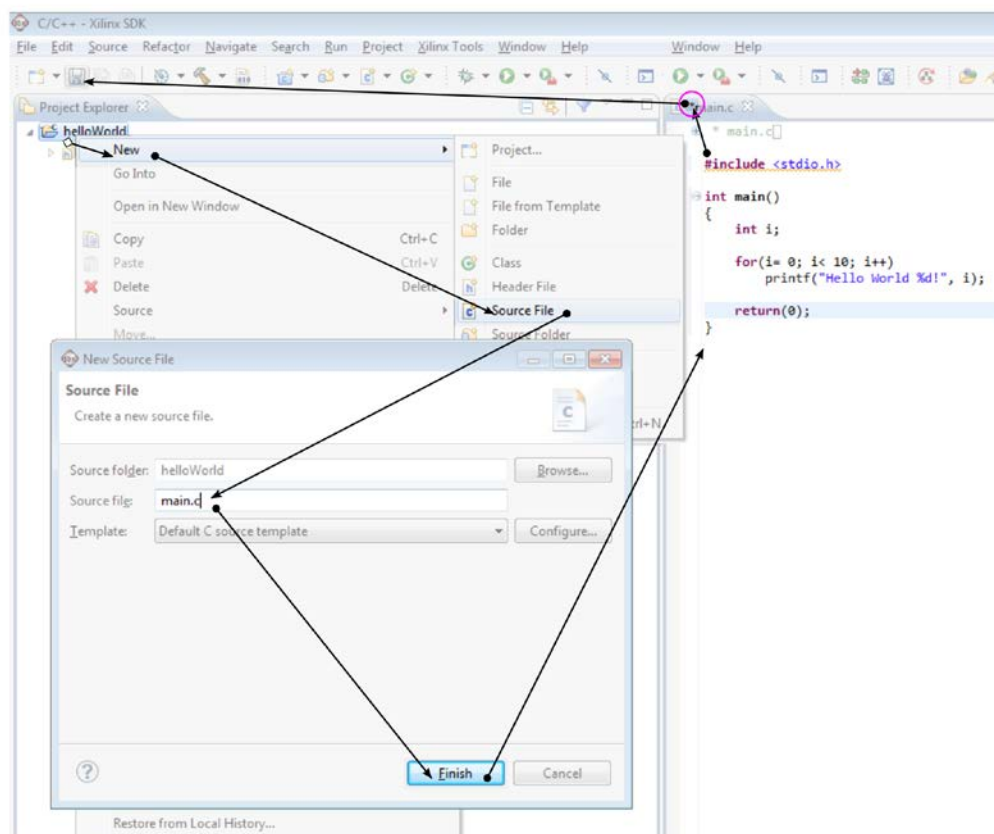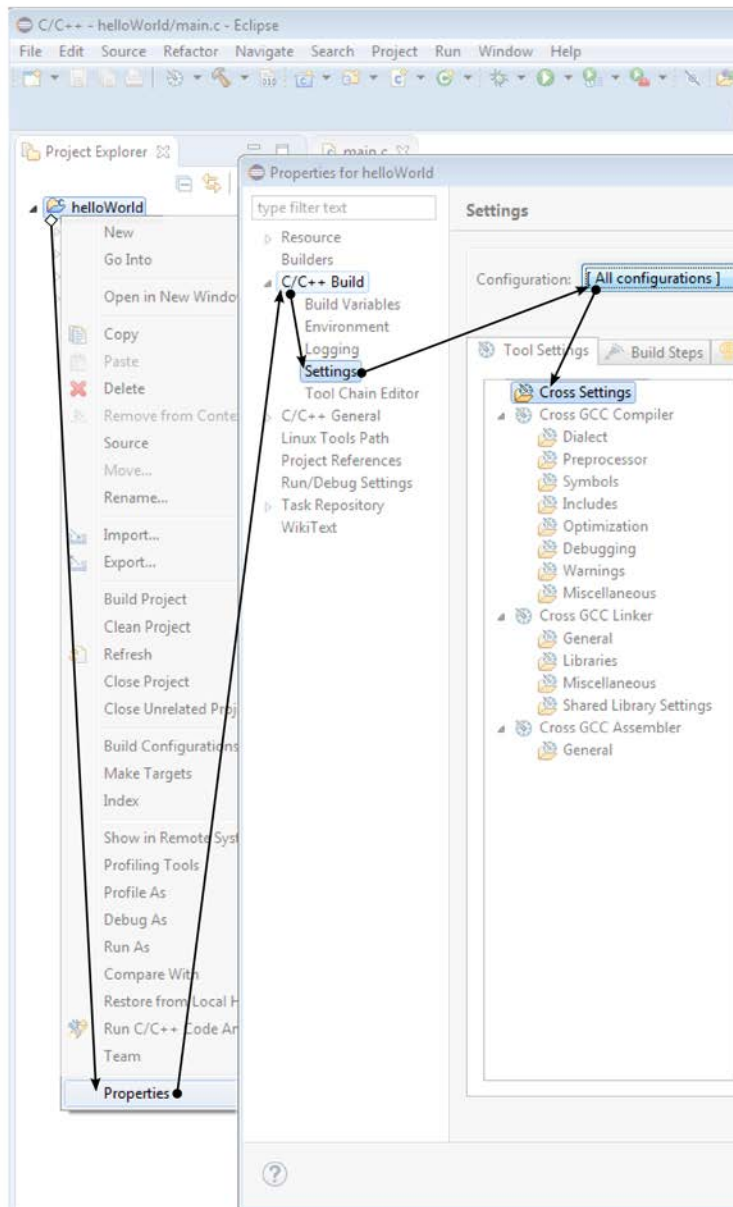
4.5.4 Building the Program

Remember to save the file after changing it!
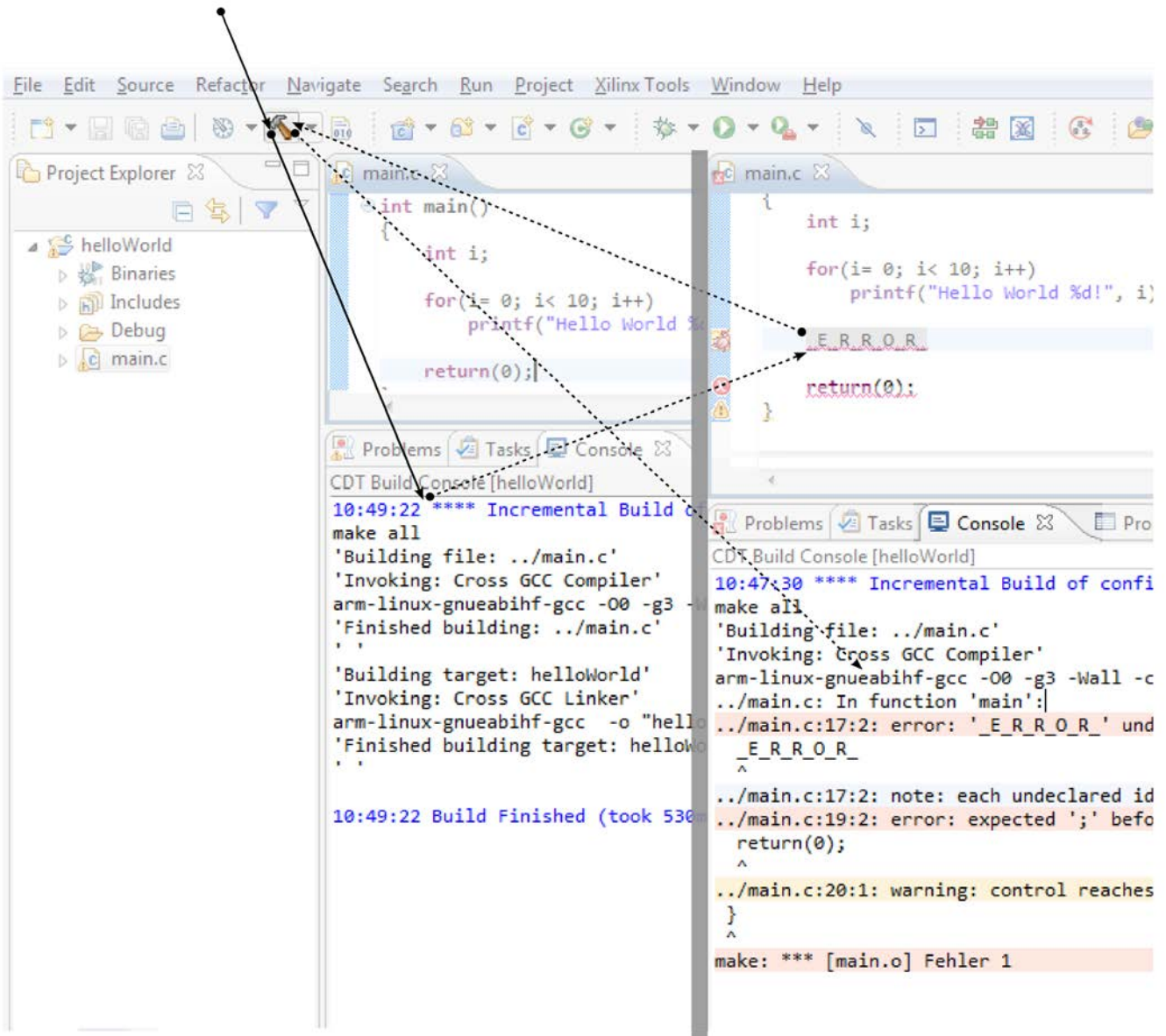
4.5.5 Linking VC Libraries & Optimization

The following image guides you through the settings necessary to use the VC Libraries and provide information of the chipset used to enable code optimization. The paths visible at this image may be replaced depending on your VC library version.

This step is necessary to be done after setting up a new C project since there is no polite way to store the information. As workaround you could store a project like the hello world program as blueprint and duplicate it instead of re-setting it up. However we recommend the manual way to avoid problems.

Prefix: arm-linux-gnueabihf-

Path: C:\Program Files (x86)\Linaro\gcc-linaro-arm-linux-gnueabihf-4.9-2014.09\bin

Include paths (-I)

"C:\vc\vclinux\include"

Optimization Level: Optimize most (-O3)

Other optimization flags

For debugging use "**None (–O0)**"

Other flags: -c -fmessage-length=0 -mcpu=cortex-a9 -mfpu=neon -ftree-vectorize -mfloat-abi=hard

☐ Verbose (-v)
☐ Support ANSI programs (-ansi)
☐ Position Independent Code (-fPIC)

Libraries (-l)

vcimgnet
vclinux
vclib
vcflib
m
rt

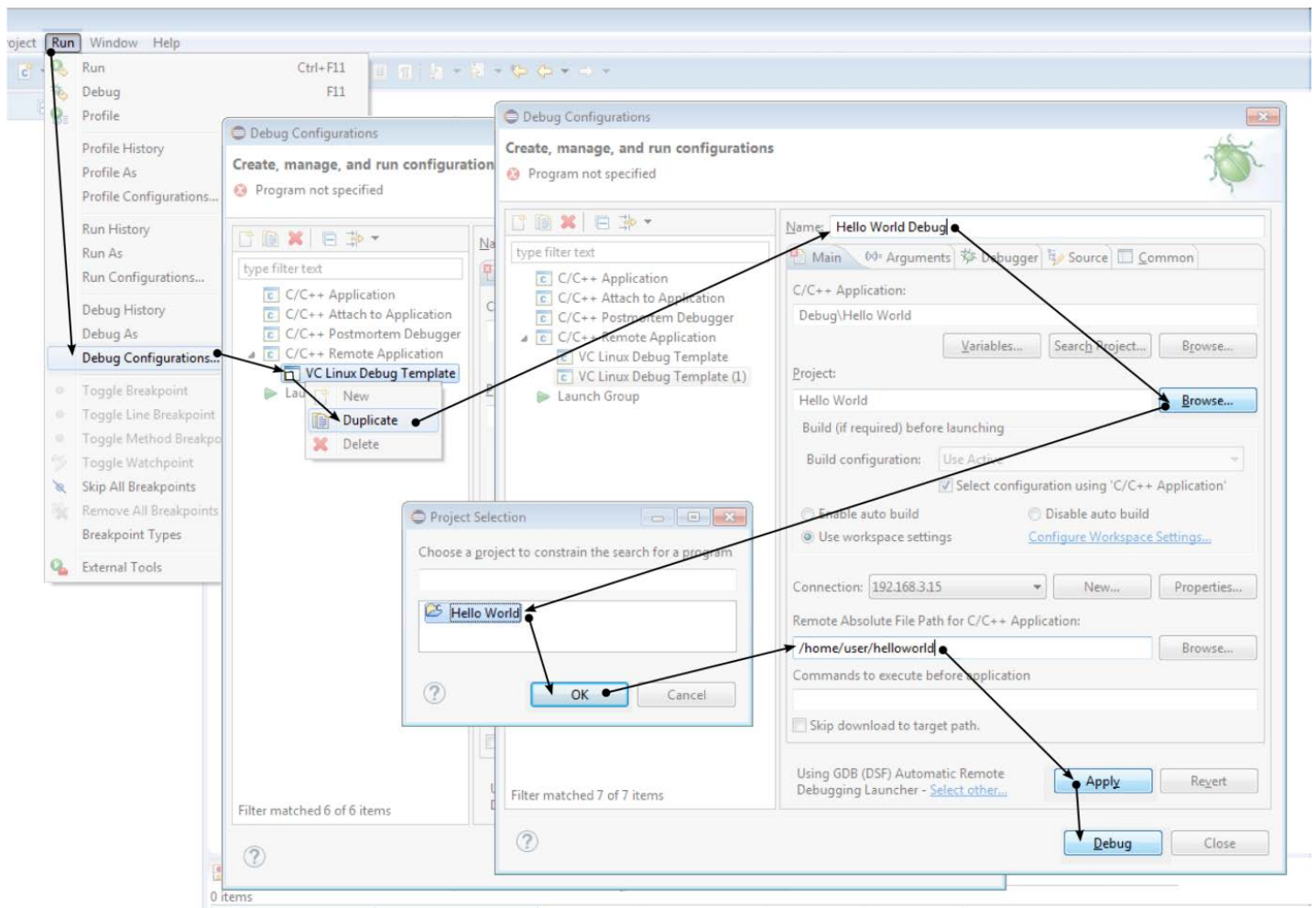Library search path (-L)

C:\vc\vclinux\lib

⚠ It is important to link the math library ("m") and the runtime library ("rt") in addition to the VC libraries!
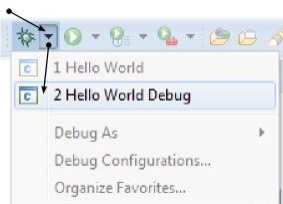
Paths may vary regarding your installation and version setup.

4.5.6   Program Execution and Debugging

We now add project specific information to a duplicate the previously added Debug Communication Setup Template and start a debug session.


After clicking the *Debug* button the IDE asks to switch to the so-called *Debug Perspective* – Accept it.

You then will see the following window. The program has been modified slightly to see the debug capability. First double-click at the left blue coloured border at the height of the *printf()* line. This sets a breakpoint. The Play/Pause button will continue execution until the next breakpoint has been reached. At its way down, the variable *i* changes its value by assignment which can be seen by the yellow coloured background at the *variables* tab.

Proceed taking a *single step without stepping through function internals for debugging* (if you'd have wished so, the left button beneath would have done the job). The *printf()* generates a line on the console as can be seen at its tab. If there is nothing happening at the console, remember, that it is line buffered. So if you forgot to add a *\n* at the end of the printf-string, the content won't be written to console out! Since nothing more is at this function, quit by clicking the *Stop Debugging* button. To switch back to the *Programming Perspective* press the *C/C++* Button at the right.

More information about debugging in eclipse can be found at the eclipse documentation.
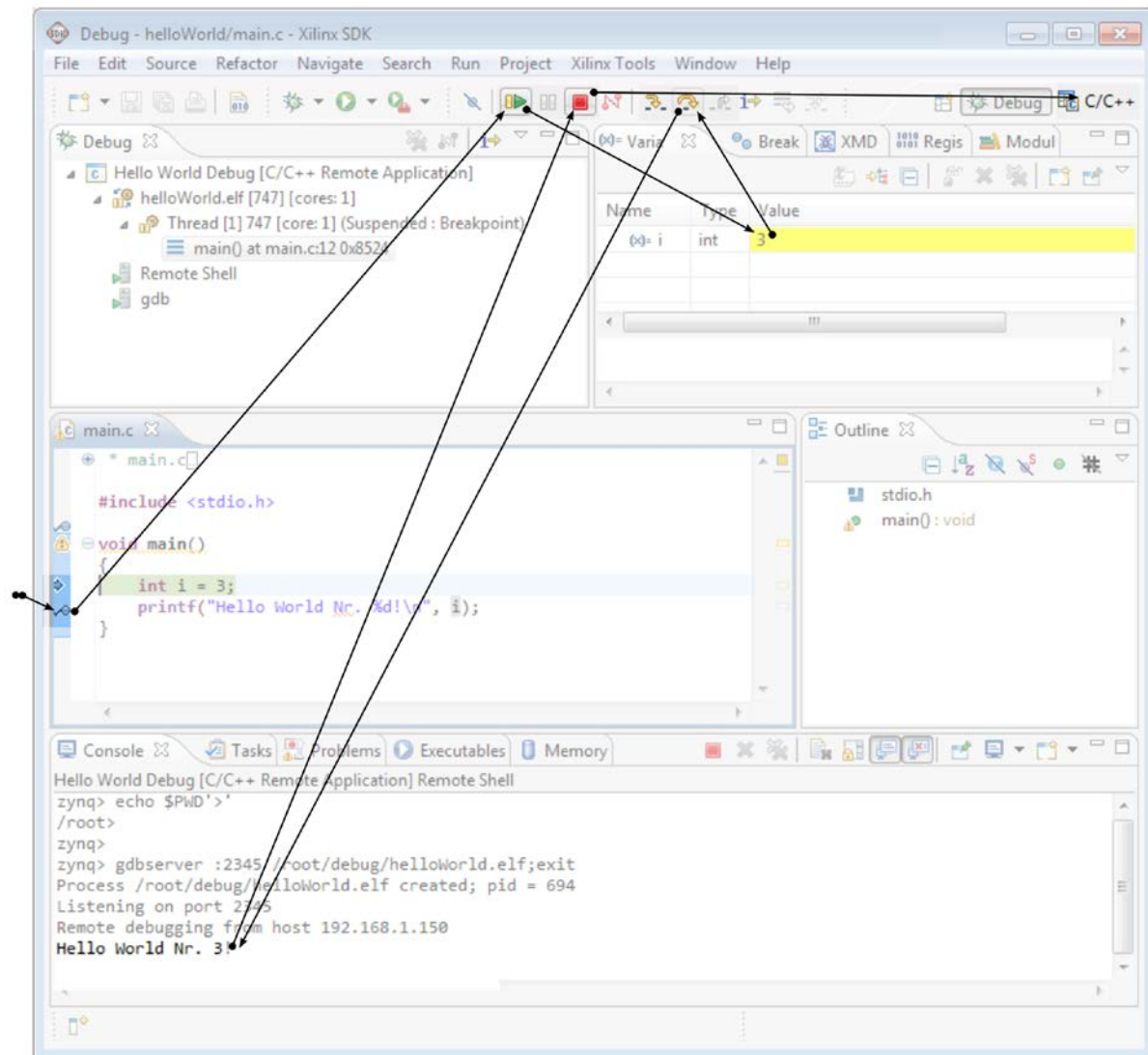
There are behaviours you should know based on the nature of the underlying debug mechanism: As a first step eclipse copies the executable to the folder you configured at the debug configuration panel under *Remote Absolute File Path for C/C++ Application*. If a previous debugging process is not cancelled correctly, this executable is still being running and thus non-overwriteable, which results in a I/O error at transfer time. The solution is the same as at the error message which tells that the *gdbserver* could not be started: After eclipse normally copied the executable to the system it starts the program *gdbserver* at the camera which itself runs your program. At your eclipse machine, eclipse starts the program *gdb* being the client component which connects to the *gdbserver* at your camera via ethernet. Since another instance is still running in the error case, you have to end the old process first to be able to proceed with a new debugging session. Therefore open a SSH Console using *TeraTerm* and type the command *ps* to get a listing of all currently running processes. There will be a line with
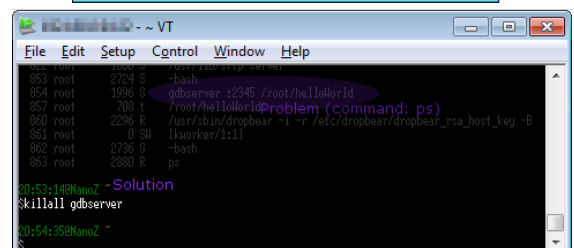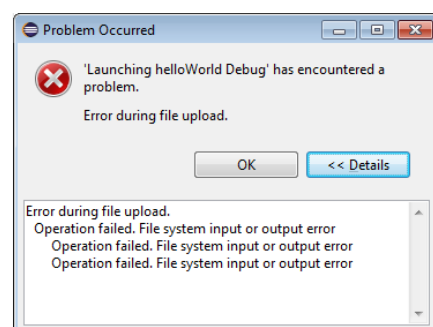
gdbserver followed by your binary location. The line starts with a *PID* number, the process ID. Each running program at the camera has such a unique processes ID. You can use the *PID* of your process to terminate it by using the following command: *kill -15 PID*. It sends a request to the gdbserver to quit. In case this does not succeed, you can tell the linux kernel to terminate the process by sending the command *kill -9 PID*.

4.6   Next steps: programming the VC Z cameras

With this working environment, you can now start programming your VC Z camera. For help you can refer to the library documentation which can be found in your installation directory, and also online.

4.6.1   Online library reference

Link for the VCLinux online documentation:
http://www.vision-components.com/fileadmin/external/documentation/software/lib/libvclinux/latest/html/index.html

The VCLinux online documentation contains some short programming examples showing the possibilities of image capture, I/O management and image transfer using vcimgnet.

Link for the VCLib online documentation:
http://www.vision-components.com/fileadmin/external/documentation/software/lib/libvc-base/latest/html/index.html

The VCLib online documentation is a reference for VC's image processing library.
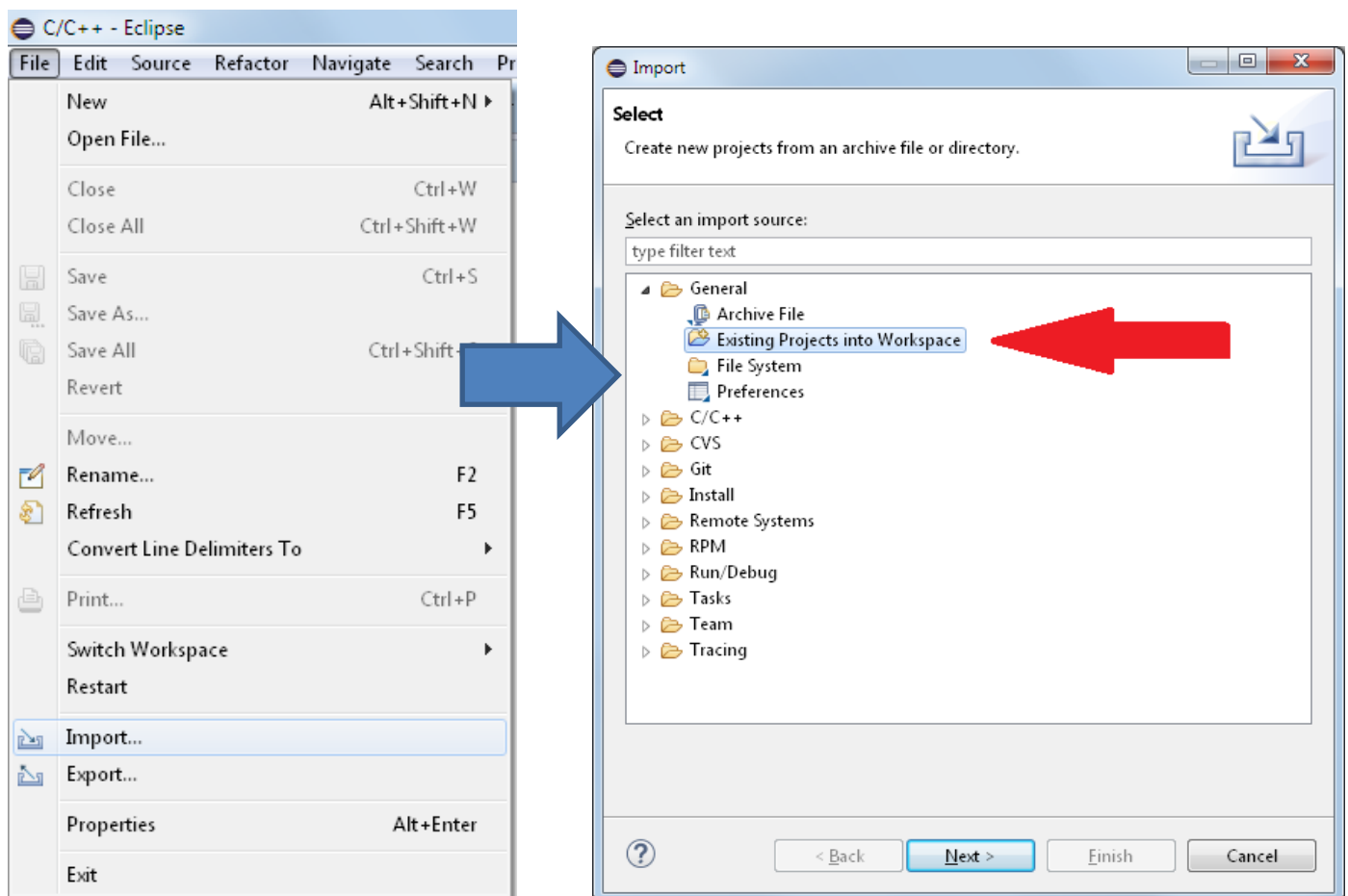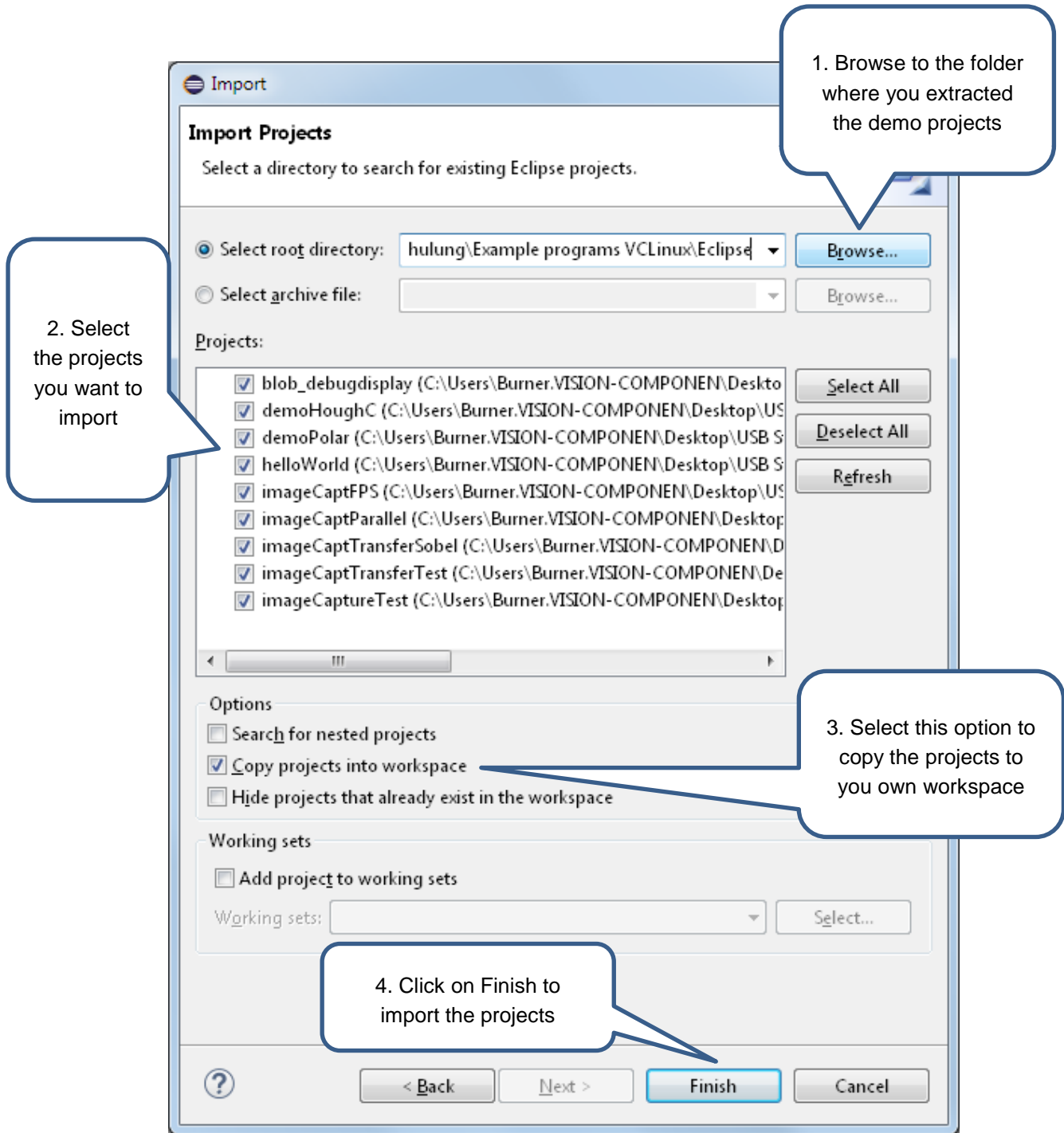
4.6.2 Example programs

Some example programs / projects are available for download:
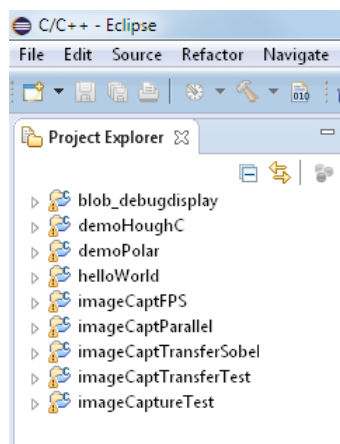**http://files.vision-components.com/Support/Eclipse_Example_Projects_VC_Z.zip**

The Projects can be imported directly into Eclipse using the import function in the menu File →
Import…

Under "General" choose "Existing Projects into Workspace":

1. Browse to the folder where you extracted the demo projects

2. Select the projects you want to import

3. Select this option to copy the projects to you own workspace

4. Click on Finish to import the projects

Import

Import Projects

Select a directory to search for existing Eclipse projects.

○ Select root directory: hulung\Example programs VCLinux\Eclipse   Browse...

○ Select archive file: Browse...

Projects:

- ☑ blob_debugdisplay (C:\Users\Burner.VISION-COMPONEN\Deskto
- ☑ demoHoughC (C:\Users\Burner.VISION-COMPONEN\Desktop\US
- ☑ demoPolar (C:\Users\Burner.VISION-COMPONEN\Desktop\USB S
- ☑ helloWorld (C:\Users\Burner.VISION-COMPONEN\Desktop\USB S
- ☑ imageCaptFPS (C:\Users\Burner.VISION-COMPONEN\Desktop\US
- ☑ imageCaptParallel (C:\Users\Burner.VISION-COMPONEN\Desktop
- ☑ imageCaptTransferSobel (C:\Users\Burner.VISION-COMPONEN\D
- ☑ imageCaptTransferTest (C:\Users\Burner.VISION-COMPONEN\De
- ☑ imageCaptureTest (C:\Users\Burner.VISION-COMPONEN\Desktop

Select All

Deselect All

Refresh

Options

☐ Search for nested projects

☑ Copy projects into workspace

☐ Hide projects that already exist in the workspace

Working sets

☐ Add project to working sets

Working sets: Select...

?   < Back   Next >   Finish   Cancel

The projects then appear in the Project Explorer:

C/C++ - Eclipse

File   Edit   Source   Refactor   Navigate

Project Explorer

- ▷ blob_debugdisplay
- ▷ demoHoughC
- ▷ demoPolar
- ▷ helloWorld
- ▷ imageCaptFPS
- ▷ imageCaptParallel
- ▷ imageCaptTransferSobel
- ▷ imageCaptTransferTest
- ▷ imageCaptureTest

Appendix A: Compiling from Shell

To compile and link your source code the following command can be executed
(you may want to use a batch or makefile):

"C:\Program Files (x86)\Linaro\gcc-linaro-arm-linux-gnueabihf-4.9-2014.09\ bin\arm-linux-gnueabihf-gcc-4.9.2"	Compiler Call
-ggdb	Optional for Debug
-O3	Optimization
-mcpu=cortex-a9 -mfpu=neon -ftree-vectorize -mfloat-abi=hard	Target System
-I"C:\Program Files (x86)\Linaro\gcc-linaro-arm-linux-gnueabihf-4.9-2014.09\ arm-linux-gnueabihf\libc\usr\include"	GCC Header Path
-IC:\vc\vclinux\include	VC Header Path
-I.\	Your Header Paths
-oFILEOut	Output Filename
FILEIn1.c FILEIn2.c	Input Files
--static	Link Libs Statically
-L"C:\Program Files (x86)\Linaro\gcc-linaro-arm-linux-gnueabihf-4.9-2014.09\ arm-linux-gnueabihf\libc\usr\lib\arm-linux-gnueabihf"	GCC Library Path
-LC:\vc\vclinux\lib	VC Library Path
-L.\	Your Library Paths
-lvclinux	Link VC Library
-lvclib	Link VC Library
-lvcflib	Link VC Library
-lm	Link GCC math
-lrt	Link GCC Runtime

To invoke a debug session by hand, copy the binary to your camera, open a console at the camera, e.g. via TeraTerm and start the gdb server application which, for example, listens on port 1234 and allows any IP to connect:

gdbserver :1234 ./FILEOut	GDB Server Call

Now you can connect to the server from your local machine by calling

"C:\Program Files (x86)\Linaro\gcc-linaro-arm-linux-gnueabihf-4.9-2014.09\ bin\arm-linux-gnueabihf-gdb"	GDB Client Call

and run the following commands at the *(gdb)* prompt:

|	
file ./FILEOut	Load Symbols
target remote 192.168.3.15:1234	Connect to Server

You may then
- list the source code including line numbers using the *l* command,
- insert breakpoints using the *b* command followed by the line number,
- single step through the program by using the *s* command,
- continue execution until at breakpoint using the *c* command,
- print a value of a parameter by using the *p* command, e.g. *p i*
- set new values to a parameter by assigning it to them, e.g. *i=22,*

- backtrace the stack by the command *bt*,
- add a parameter to be displayed continuously by using the *watch* command, or
- leave the program by using the command *quit*.

Refer to the GDB documentation for other tasks.

Appendix B: Communication Error Resolving
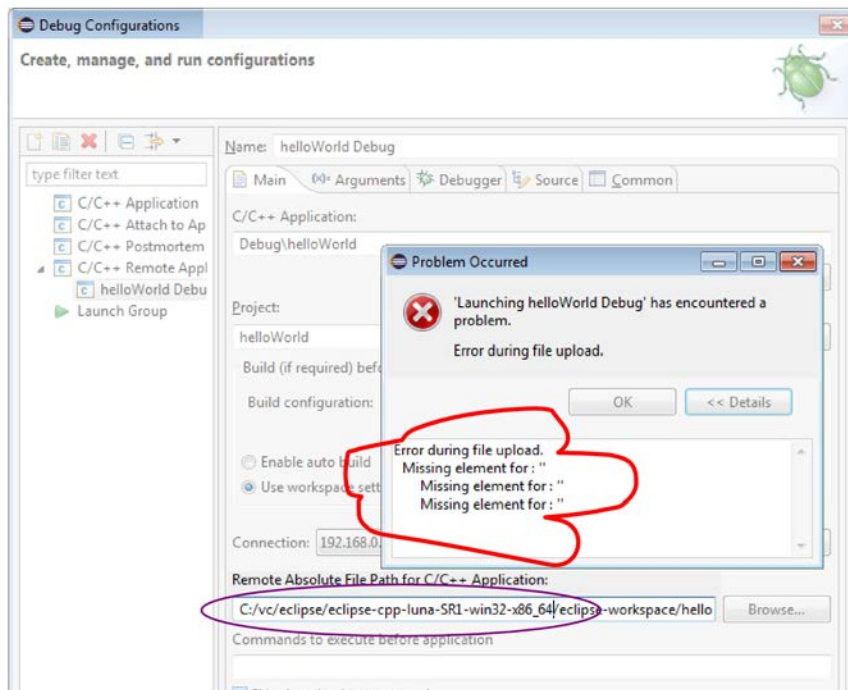
If you get the error message above, it is probably due to a wrong path entered for "Remote absolute file path" in the Debug Configuration window.

Appendix C: Changing the IP address and DHCP

To change the IP address of a Z series camera edit the file **vcsetip.scr** (under /root) and modify the parameter **ipaddr**:

```
ipaddr    192.168.3.15
netmask   255.255.255.0
gatewayip 192.168.3.254
serverip  192.168.3.35
dnsip     192.168.3.254
dnsip2    192.168.3.254
addip     setenv bootargs ${bootargs}
"ip=${ipaddr}:${serverip}:${gatewayip}:${netmask}:${hostname}:${ethdev}:off
:${dnsip}:${dnsip2} "
```

Other network parameters like subnet mask, gateway and DNS addresses can also be modified here. Do not modify the last line of the script.

Then run the script **vcsetip.sh** to apply the changes. Reboot the camera.

To activate the DHCP client, run the script **vcsetdhcp.sh**. Reboot the camera.

Appendix D: Starting programs automatically

To start a program automatically, please proceed like this:

- create a script file **user_init.sh** in the folder **/etc/vcinit/**

- write in it your program calls

- the script user_init.sh has to be executable. This can be done with the command **chmod +x user_init.sh**

- the script **/etc/vcinit/vcinit.sh** calls the script user_init.sh at startup if it is available.

Pay attention to the path for the programs to be executed.

Appendix E: Camera image transfer

An image viewing application is provided by Vision Components on every nano Z camera. It consists of two shell commands on the camera:
- **vcimgnetsrv**: this program is an image server. It runs in the background and transfers the content of a specific memory area over Ethernet to a client software on a PC
- **vctp**: this program does a continuous image capture and copies the captured image in the memory area for vcimgnetsrv to transfer

On the PC side the client software **ATXClient** (alternative: Python script **vcimgnetclient.py**) receives the images and display them. The PC clients can be downloaded here:
**http://files.vision-components.com/Support/vcimgnet_PC_clients.zip**

Follow these steps:

1. Install the client program on the PC by registering the ATXControl control in the windows command line with **"regsvr32.exe atxcontrol.ocx"**. On Windows 7 execute cmd.exe with administrator rights!
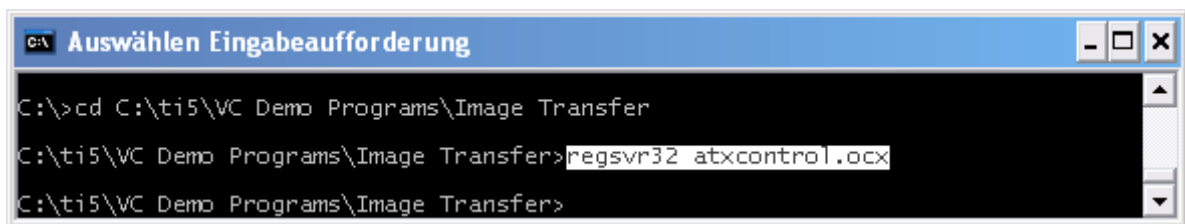
Registering atxcontrol.ocx

2. On the camera, run the program vcimgnetsrv in the background with the command **"vcimgnetsrv &"**.

3. Start the ATXClient program and enter the correct parameters (IP address of your camera under **Server IP**, resolution under **DX and DY**). Check the box "live mode" to get continuous images.

4. Click on **"Get Image"** to start the image transfer. At this point you should see a moving bar in the middle of the image (meaning the client is connected to the server on the camera but no images are being captured yet):
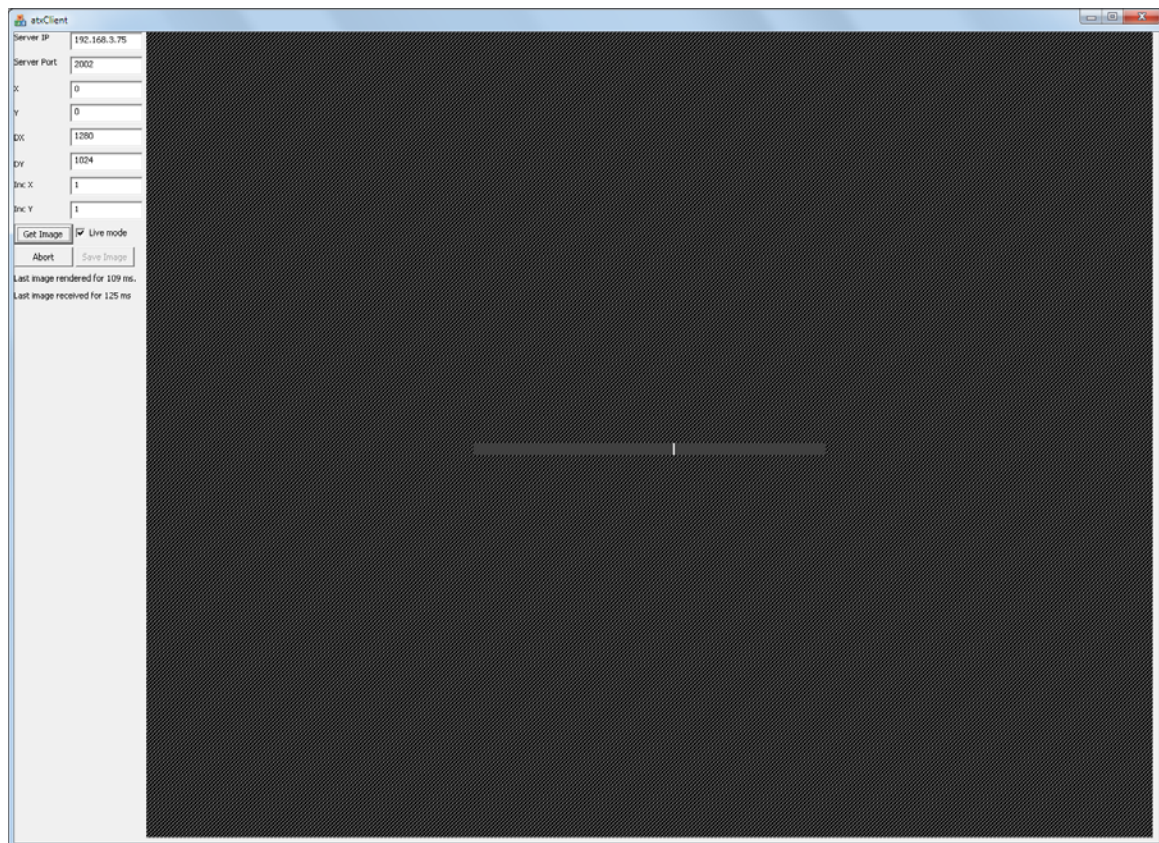
5. To start capturing start the program vctp from the camera shell with **"vctp"**. The live image should then be transferred:

Alternatively to the ATXClient, the Python client **vcimgnetclient.py** can be used. This client needs a Python 2.7.x (https://www.python.org/downloads/release/python-2710/) and PyGTK (http://www.pygtk.org/downloads.html) installation on the PC.

The server **vcimgnetsrv** can be used from any other user program by using the vcimgnet programming interface and is useful for visual debugging. More information can be found in the online documentation: **vcimgnetsrv documentation**.