

User Guide

CC-IDE User Guide

1.1

Scope

This document contains the user guide of Integrated Development Environment for ChipCraft processor classes based on supported architectures. Installation requirements and basic usage are described. The document covers project creation, compilation and debugging with build-in simulator as well as the FPGA or ASIC hardware.

Contents

1. Installation	3
1.1 Download	3
1.2 Requirements	3
1.2.1 Operating system	3
1.2.2 Java 8	3
1.2.3 Python	4
1.2.4 Python pyserial module	4
2. Using CC-IDE	5
2.1 Starting and closing IDE	5
2.2 Creating a project	7
2.3 Building a project	12
2.3.1 Perspectives	16
2.4 Project properties	16
2.5 Debugging	18
2.6 IDE preferences	23
3. Troubleshooting CC-IDE	26
4. Revision History	27



1. Installation

1.1 Download

ChipCraft's Integrated Development Environment (CC-IDE) is available at <https://github.com/chipcraft-ic>, as part of ChipCraft's Software Development Kit (CC-SDK). Separate SDK packages can be downloaded for each supported architecture and operating system. Each package contains libraries, C headers and sources for given architecture, development tools with cycle-accurate processor simulator and graphical user interface IDE based on Eclipse CDT (Eclipse for C++ Developers).

1.2 Requirements

1.2.1 Operating system

CC-IDE is distributed for Linux and Windows. Both versions require 64-bit host architecture.

Supported Linux distributions: CentOS 7, Ubuntu 20.04.

Supported Windows versions: 10.

CC-IDE could work on different versions of Linux and Windows, however the company does not guarantee that.

1.2.2 Java 8

CC-IDE requires Java 8 runtime environment, 64-bit edition.

On **Windows** you can install 64-bit Oracle JRE manually after downloading it from Oracle website:

<https://www.oracle.com/java/technologies/javase/javase8u211-later-archive-downloads.html>

On **Linux** you can install OpenJDK 8 using your favorite package manager. In Ubuntu 20.04 run in terminal:

```
sudo apt-get install openjdk-8-jre
```

Please make sure you use 64-bit version 8 of JRE software. CC-IDE could work on newer versions however the company does not guarantee that. To verify your version of JRE run:

```
java -version
```



1.2.3 Python

CC-IDE uses Python scripts as part of its development flow, for example for debugging firmware downloaded to ChipCraft devices or running via a simulator. It is recommended to use latest version of Python 2.

On Windows you can install it manually after downloading from official Python website:

<https://www.python.org/downloads/>

On Linux install it using your favorite package manager. On Ubuntu 20.04:

```
sudo apt-get install python2
```

CC-IDE could work on different versions of Python, including Python 3, however the company does not guarantee that.

1.2.4 Python pyserial module

CC-IDE uses the `pyserial` Python module to connect with devices through serial port. Install it using your favorite Python package manager, e.g.

```
pip install pyserial
```

If pip command cannot be found try:

```
python2 -m pip install pyserial
```



2. Using CC-IDE

CC-IDE is an Integrated Development Environment provided by ChipCraft for developing software for devices using ASICs produced by the company. It is based on Eclipse CDT, version 4.16 (2020-06), using the same basic usage and concepts. This manual is describing only the custom elements added over Eclipse CDT and basic features important for embedded software development for ChipCraft's products.

Eclipse CDT documentation can be found on: <https://www.eclipse.org/cdt/documentation.php>.

2.1 Starting and closing IDE

To start CC-IDE run `ccide.exe` (Windows) or `ccide` (Linux) executable file in CC-IDE's `ide` subdirectory.

During the first start IDE will ask to select a workspace directory as can be seen on figure 2.1. This directory is used as a default location for all new projects. Any empty directory will do. You can change it later using `File` menu. If you are going to work with this directory for a longer time it is recommended to check **Use this as the default and do not ask again** option.

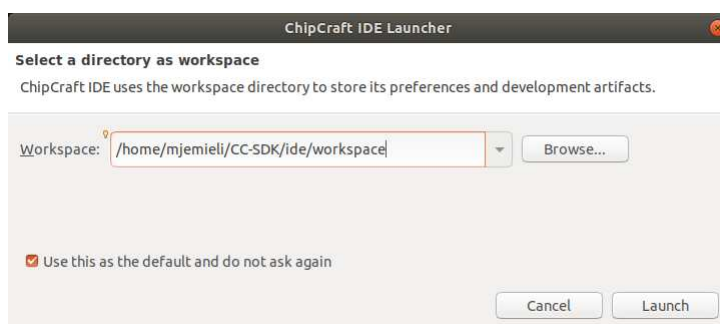


Figure 2.1. Workspace directory selection window.

After starting the IDE you should see empty IDE window (figure 2.2)



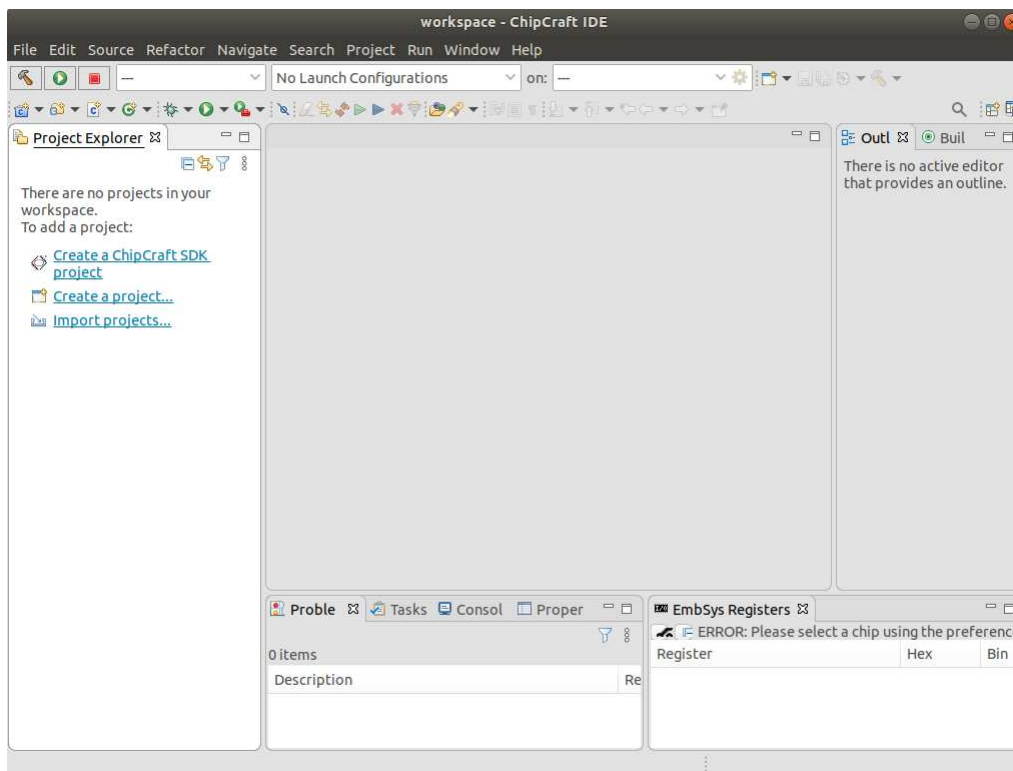


Figure 2.2. Empty CC-IDE window.

To close the IDE select **File -> Exit** from menu or click a cross symbol on the window titlebar. In the latter case you will be asked for confirmation (figure 2.3).

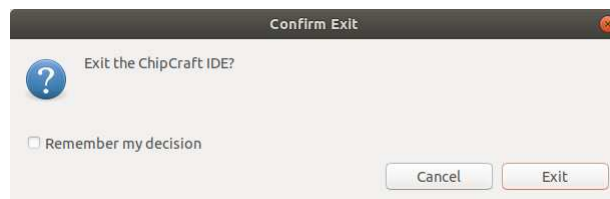


Figure 2.3. Confirmation on closing CC-IDE.



2.2 Creating a project

To create a new project select **File -> New -> ChipCraft SDK Project** from main window menu (figure 2.4).

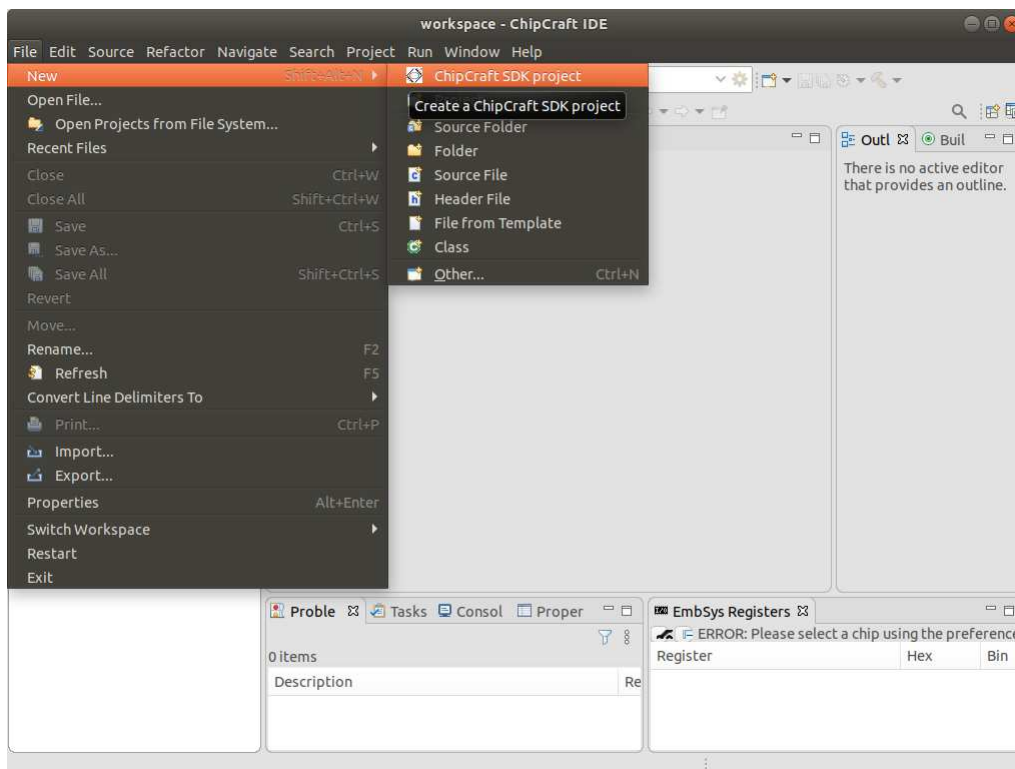


Figure 2.4. File -> New menu.

First page of New Project wizard will show up (figure 2.5). Specify a project name. Project location is auto-generated using workspace path and project name. If you want you can override it by unselecting **Use default location** checkbox.

After filling the fields click **Next**.

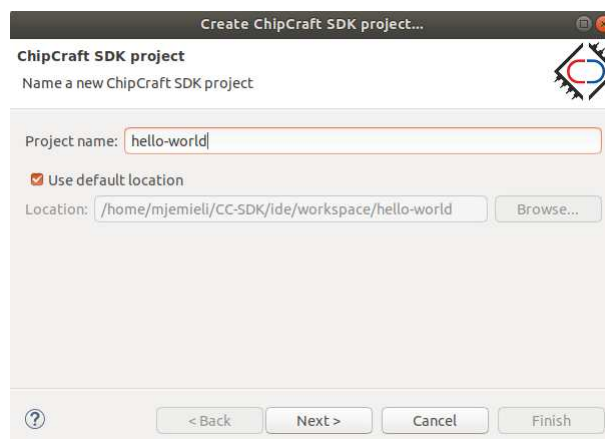


Figure 2.5. First page of New Project wizard.

In the second page of New Project wizard (figure 2.6) you can provide basic configuration of the project.



The **SDK** field allows customization of CC-SDK location. By default the current directory is used. If environmental variable `CHIPCRAFT_SDK_HOME` is set, then its value is used. Note that if proper CC-SDK directory is not detected, it's not possible to choose toolchain and board.

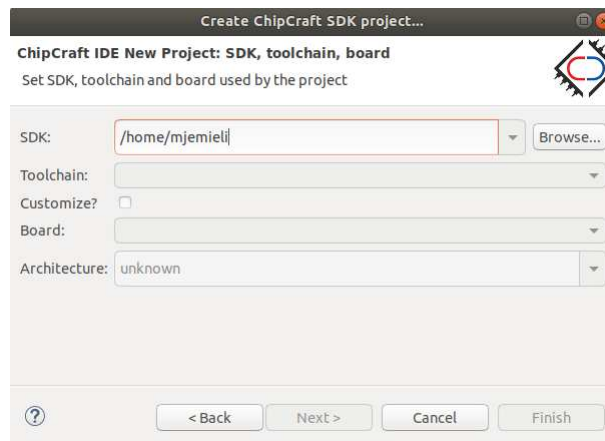


Figure 2.6. Second page of New Project wizard, SDK.

After proper CC-SDK directory is given, the **Toolchain** and **Board** fields become active and prompt the user to choose one of the options (figure 2.7).

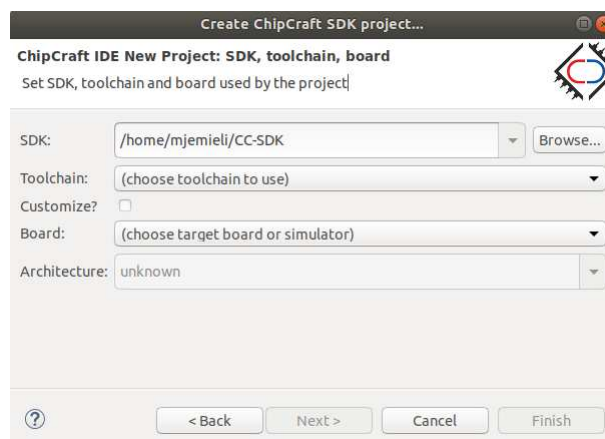


Figure 2.7. Second page of New Project wizard, proper SDK.

To choose the CC-SDK directory, users can also browse directory structure using the **Browse...** button. When a directory is selected, press the **Open** button to use it (figure 2.8).

Next, choose a toolchain from list of toolchains available in the chosen CC-SDK (figure 2.9). Each toolchain generates valid binaries for one of supported architectures. Currently ChipCraft distributes SDK packages for single architectures, with single toolchain supporting that architecture. In the future SDK packages for multiple architectures may become available. The IDE is designed to be future-proof and will work with multiple toolchains, for example allowing to use different toolchain versions for a single supported architecture or multiple toolchains for multiple architectures. Alternatively, start by choosing the board you want to develop for. CC-SDK package may contain boards for multiple architectures (figure 2.10).

Note that when you choose a toolchain, it will filter the selection of available boards to those that fit the toolchain architecture (figure 2.11). This ensures your code will compile properly for given toolchain and board combination.



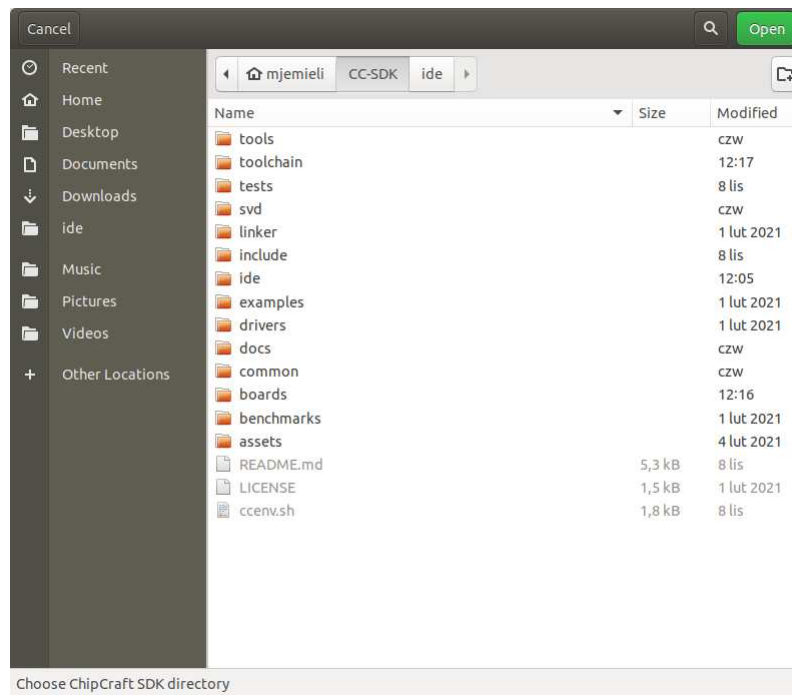


Figure 2.8. Second page of New Project wizard, browsing for SDK.

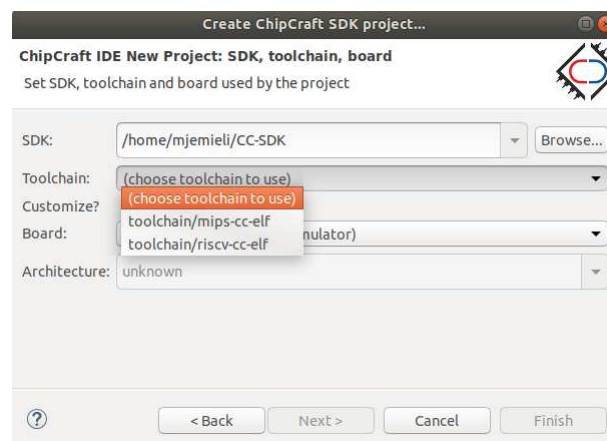


Figure 2.9. Second page of New Project wizard, listing toolchains.

Similarly, choosing a specific board will filter available toolchains to those that will generate proper code for that board (figure 2.12).

It's possible to proceed further with New Project wizard when both a toolchain and a board are chosen (figure 2.13).

Sometimes it's necessary to customize the resultant firmware's binary by changing the linker script used by the toolchain. Since this isn't usually necessary, the option is left for power users and can be enabled by ticking the **Customize?** box. It will then show a list of linker scripts suitable for chosen toolchain available by default or allow to provide your own (figure 2.14).

Similarly to choosing SDK, user can browse for custom linker script by pressing the **Browse...** button. When the proper file is selected, press **Open** to set it (figure 2.15).

Or, if you know the path to your custom linker script, write it directly in the **Linker script** field (figure 2.16).



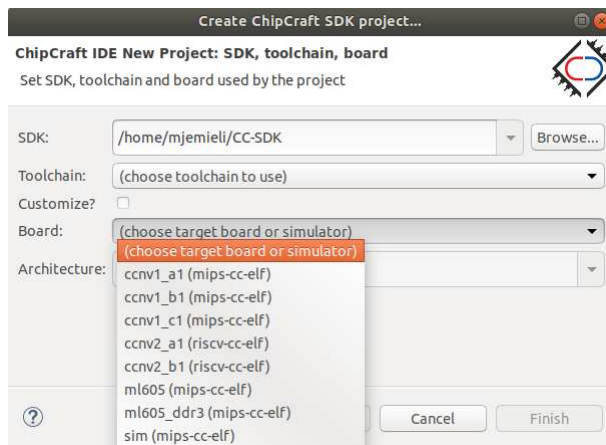


Figure 2.10. Second page of New Project wizard, listing boards.

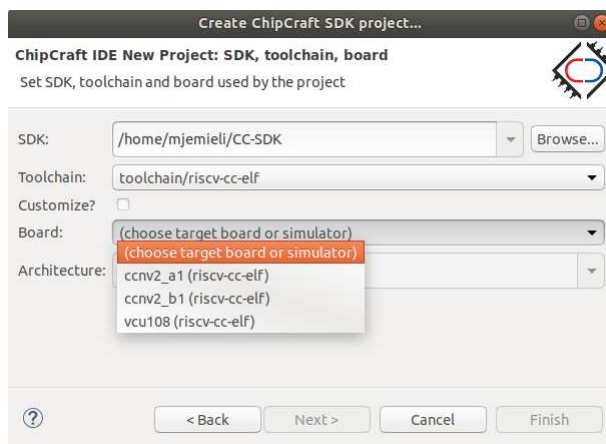


Figure 2.11. Second page of New Project wizard, filter by toolchain. Choosing toolchain for one architecture filters available boards to this architecture

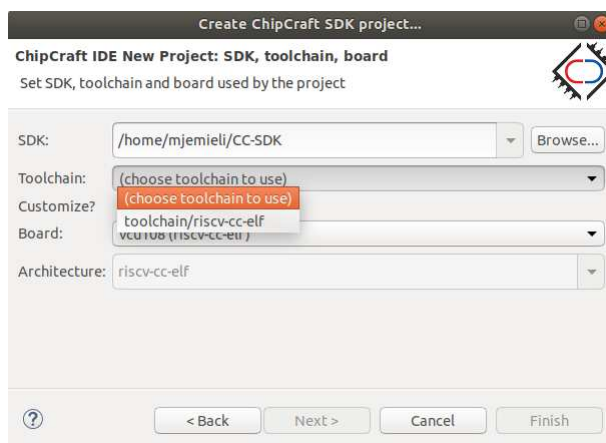


Figure 2.12. Second page of New Project wizard, filter by board. Choosing board for one architecture filters available toolchains to this architecture

On the third page of New Project wizard, the user can set input/output communication channels used when developing software for the board. Communication happens over UART serial port. The wizard automatically fills out default baud rates for a board. Usually those don't need to be changed (figure 2.17).



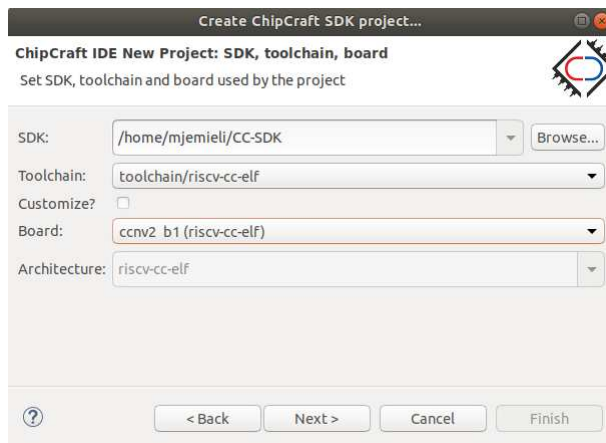


Figure 2.13. Second page of New Project wizard, toolchain and board chosen.

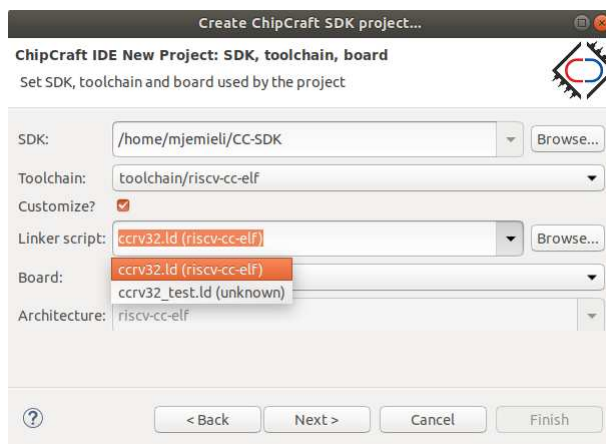


Figure 2.14. Second page of New Project wizard, customize linker script.

For physical board select proper debug and UART port. The first one is used for sending debugger commands and receiving results. It's usually used by the toolchain debug facilities. The second one is used as input/output stream by user programs. On Linux ports usually follow a pattern: `/dev/ttyUSBn`, however due to many different configurations the wizard will show all available serial ports. On Windows ports follow a different pattern: `COMn`. CC-IDE will show all connected ports as combo-box suggestions (figure 2.18).

A custom path can also be directly given in the appropriate field (figure 2.19). Note: for simulator-based project, neither debug nor UART ports are used. Any value can be given in this case.

Finally, if you have a hardware JTAG device compatible with the board you're using, tick the **JTAG** box (figure 2.20). This will set up the project to use JTAG instead of serial port communication which is usually much faster and reliably. Please contact ChipCraft for choosing the proper JTAG device.

After configuring input/output devices, click **Finish**. New project will be generated and the IDE will open *main.c* file (figure 2.21). It will implement a trivial *Hello world!* firmware.



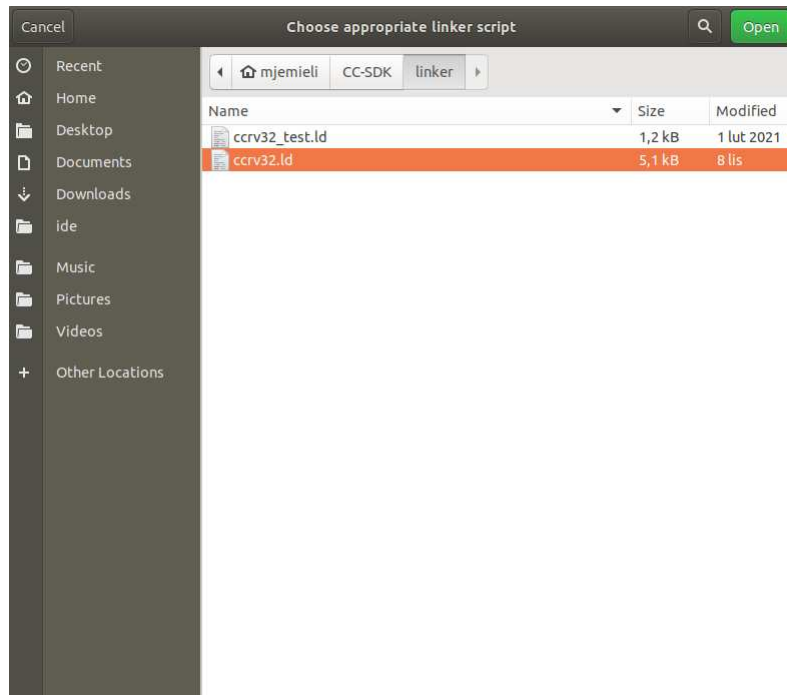


Figure 2.15. Second page of New Project wizard, browse for custom linker script.

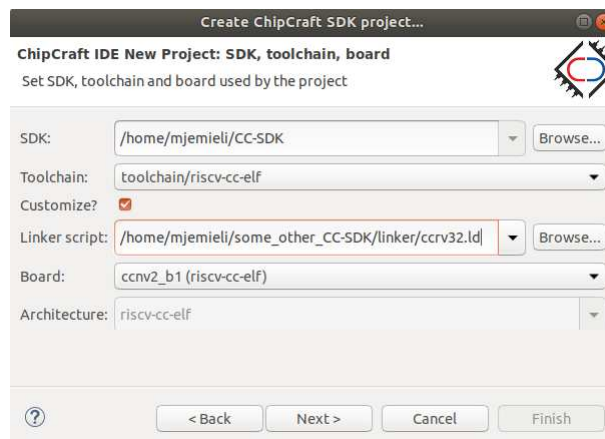



Figure 2.16. Second page of New Project wizard, path to custom linker script.

2.3 Building a project

To build a project click on a build button . You can also select option **Project -> Build All** from menu.

CC-IDE has dedicated toolbar for most used actions when working with CC-SDK projects. It is visible on figure 2.22.

Toolbar has following buttons:

- Clean - removes build artifacts from the project directory;
- Build - builds the project;
- Rebuild - cleans, then builds the project;
- Program - write the application into device's non-volatile memory;



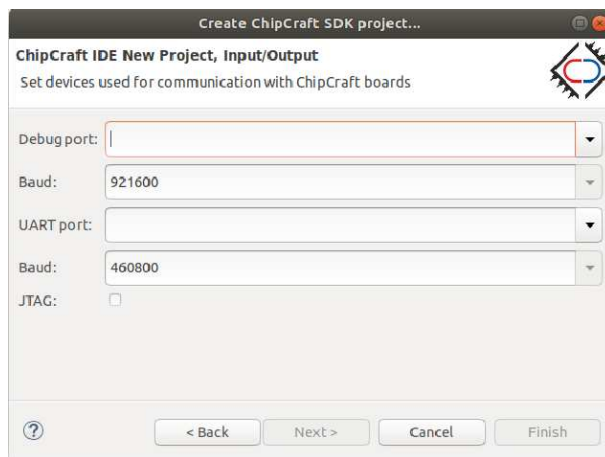


Figure 2.17. Third page of New Project wizard, default baud rates.

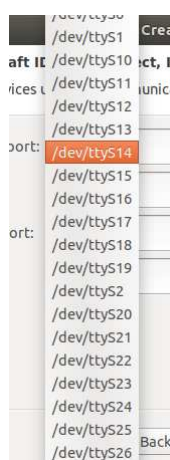


Figure 2.18. Third page of New Project wizard, all detected serial ports.

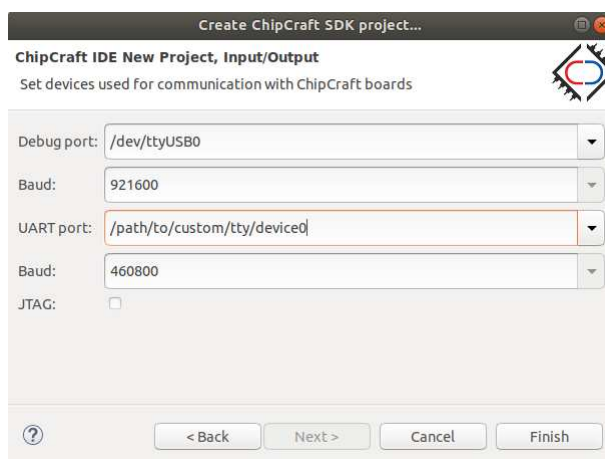


Figure 2.19. Third page of New Project wizard, custom path to serial device.

- Flash - write the application into device's volatile memory (RAM);
- Reset - resets the device;
- Terminal - opens serial terminal to communicate with the device.



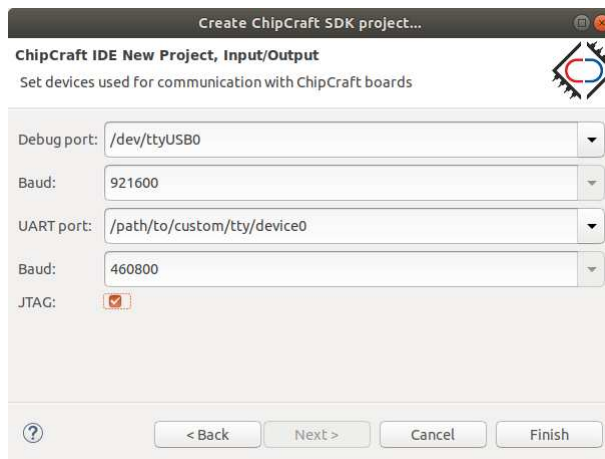


Figure 2.20. Third page of New Project wizard, JTAG support enabled.

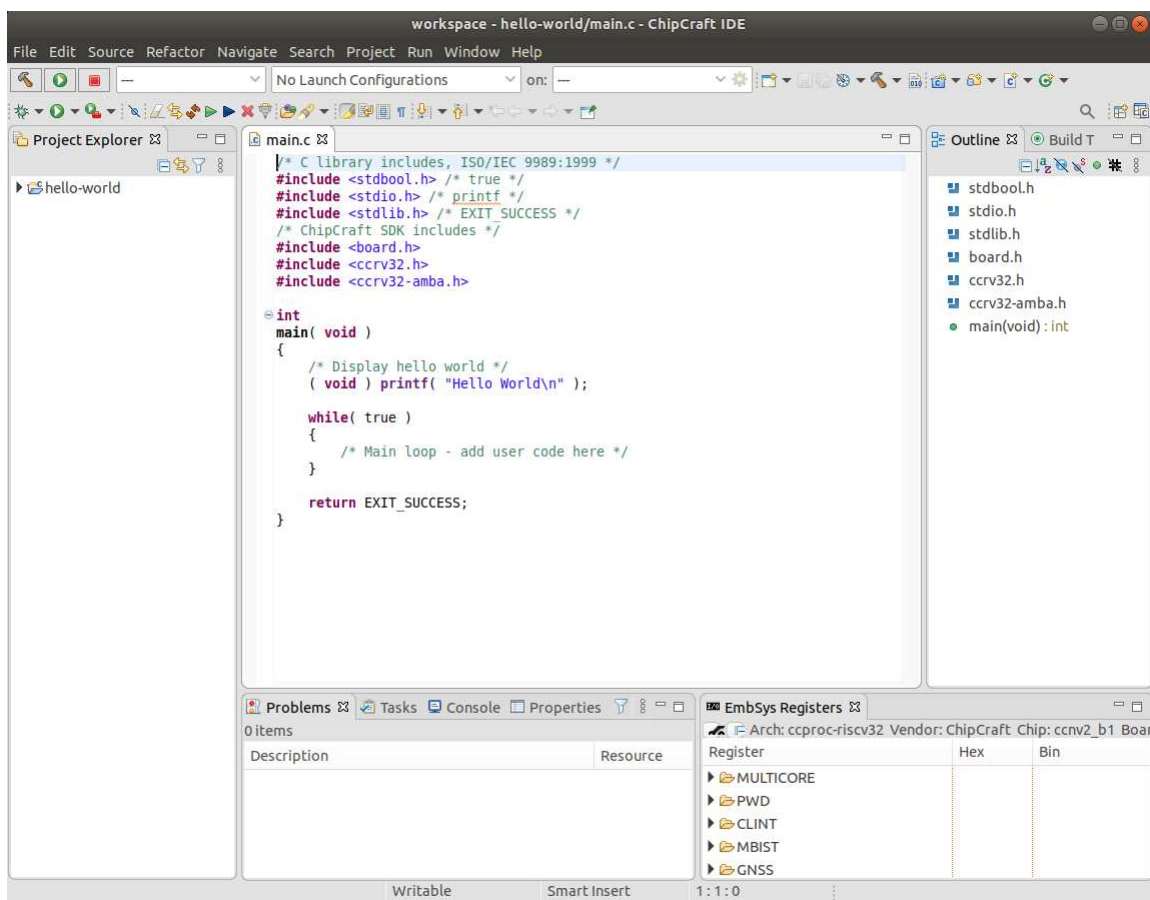


Figure 2.21. CC-IDE main window after project creation.

All toolbar buttons work in context of current project. It is determined from selected element in project tree or from file currently open in editor window (depending on input focus).

Projects created by CC-SDK Project wizard contain the `common` virtual folder (visible on figure 2.23). All files in this folder are symbolic links to corresponding files in CC-SDK. You should not modify them because all projects share them. If you need to make changes you should copy a file to your project and remove from `common` folder.



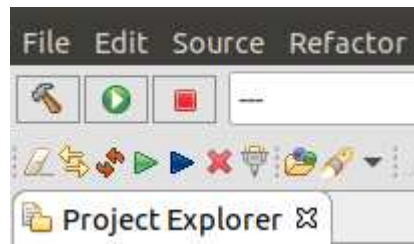


Figure 2.22. CC-IDE Toolbar.

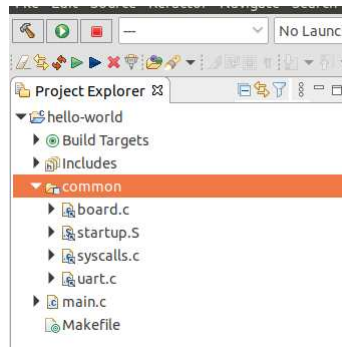


Figure 2.23. Linked source files.

Custom Makefile actions can be configured in CC-IDE as Build Target items. They are visible in Project Explorer (figure 2.24) and a dedicated view (figure 2.25). By default the New Project wizard adds basic Build Targets for resetting and programming a device.

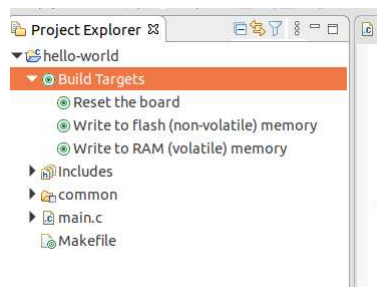


Figure 2.24. Build targets in project tree.

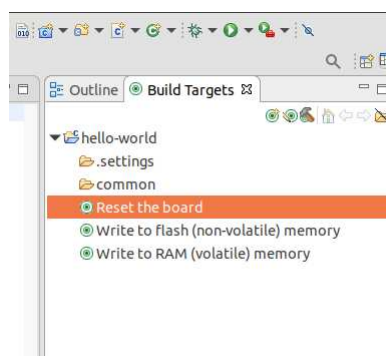


Figure 2.25. Build targets view.



2.3.1 Perspectives

CC-IDE supports perspective concept. User can customize the main IDE window by adding views and changing their position. All such actions happen in context of a perspective. IDE can have multiple perspectives optimized for different tasks, like code development, debugging and so on. By default CC-IDE uses two perspectives: **C/C++** and **Debug**. User can create more if needed. List of all visible views and their positions are persisted in perspective so when IDE is restarted they are in the same place.

2.4 Project properties

Project properties can be changed by clicking RMB¹ on the project in the Project Explorer view and selecting *Properties* from a context menu.

All project properties are grouped in categories visible in a tree control in the left side of the window. Categories can be filtered by providing a query text inside field above the category tree. To save changed properties and close the window click **Apply and Close** button at the bottom.

Options directly responsible for building the project are in *C/C++ Build* category.

C/C++ Build -> Build Variables page allows to set custom variables accessible from some CC-IDE input fields. They can be used in a form of placeholders: `${xxx}`. They are useful if the same string is needed to be repeated multiple times in configuration. To determine if input field supports variables check if *Variables...* button is nearby.

C/C++ Build -> Environment page (figure 2.26) allows to set environment variables used during the build process. They are accessible from *Makefile* scope. New Project Wizard configures multiple environment variables starting with `CHIPCRAFT_SDK_` prefix. They should not be changed nor removed using this page.

CC-IDE uses simplified project build, where compiler and linker invocations are hard-coded inside the *Makefile*. User is expected to modify the Makefile to change them and the flags used during build. The project settings page in **C/C++ Build -> Settings** contains only parsers configuration. This is due to flags and options required to be set for proper firmware build. Relying on single *Makefile* included from CC-SDK is simpler and less error-prone in case any changes are necessary - in case the build process needs to be updated, it only requires new version of CC-SDK to work correctly for all projects, instead of requiring to correct project properties for every project.

Ability to set separate CC-SDK directory for every project and replace linked common files strikes good balance between being approaching for new users and accommodating for power users.

C/C++ General -> Paths and Symbols page is used to configure include directories, C preprocessor definitions and libraries used by the linker.

Includes tab (figure 2.27) allows configuration of include paths. Paths are added independently for used programming languages (e.g. C, C++, Assembly). Environment and build variables can be used in path definition.



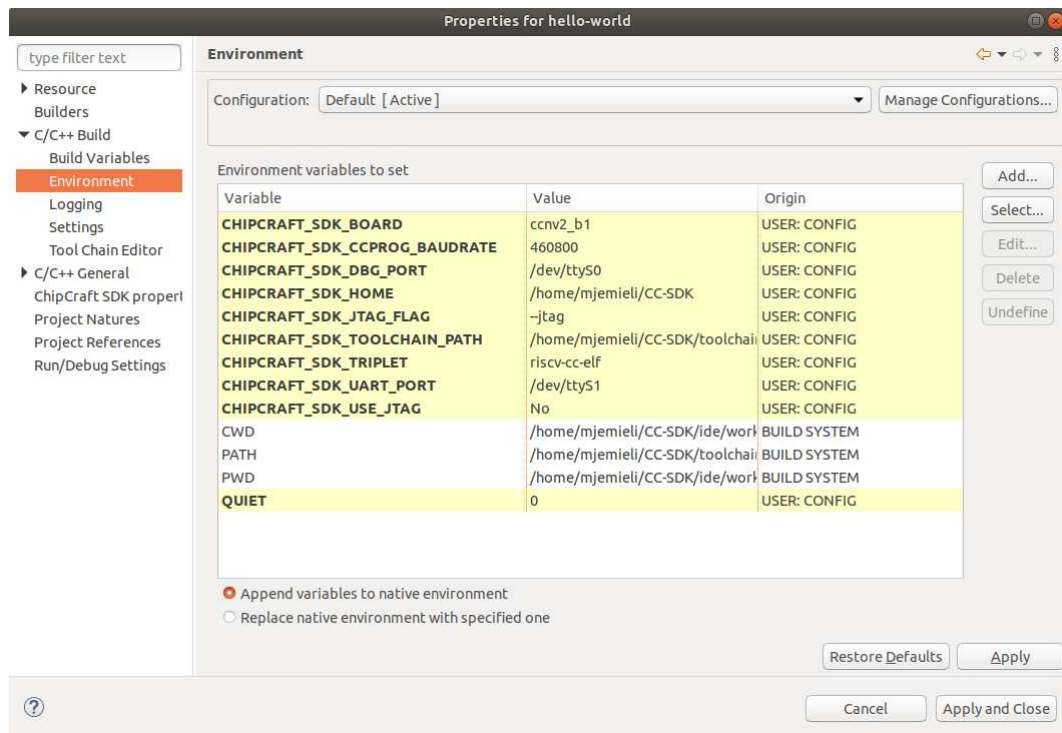


Figure 2.26. Environment variables configuration.

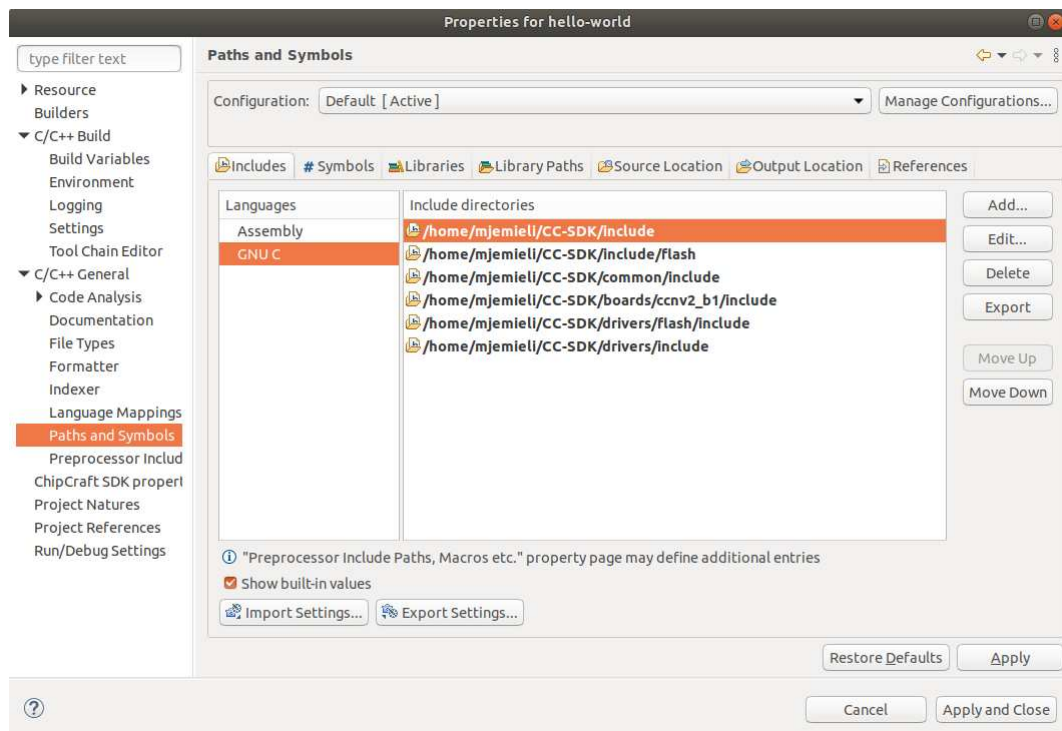
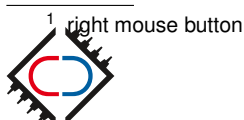


Figure 2.27. Include directories configuration.

CC-IDE adds multiple paths when generating the project (they start with `${CC-SDK_HOME}`). They should never be deleted.

Symbols tab (figure 2.28) allows configuration of C preprocessor definitions. Similarly to *Includes* tab you can use environment and build variables during symbol definition. Definitions are independently defined for each language (C, C++).



CC-IDE adds some required symbols when generating the project (like `BOARD`). Please do not delete them from configuration.

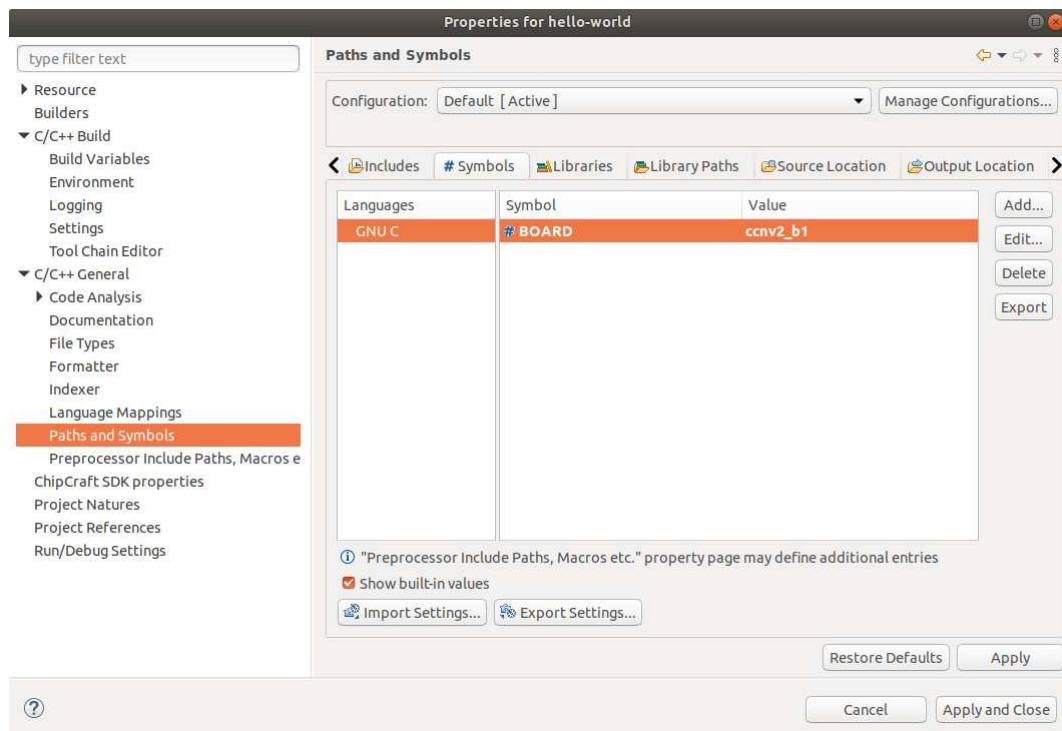


Figure 2.28. Preprocessor symbols configuration.

CC-SDK specific configuration can be found in **ChipCraft SDK properties** page (figure 2.29). It has the same options as you encounter during new *ChipCraft SDK Project* wizard. CC-SDK home location, toolchain, board, debug and UART (stdio) ports can be configured. It is useful if configuration provided during project generation needs to be changed.

2.5 Debugging

To debug an application in CC-IDE a launch configuration has to be created. There are two supported configuration types: *ChipCraft SDK Device* and *ChipCraft SDK Simulator*. The first one is used with a dedicated ASIC or FPGA board connected to the computer using a serial port or JTAG device. The second one uses simulator software included in CC-SDK package.

Launch configuration can be run in one of two defined modes: **Run** (for quick running without attaching a debugger program) and **Debug** (full featured debug mode). Depending on launch mode different perspective is used.

Launch configuration can be created by clicking on *Launch Configurations* list in the toolbar and selecting *New Launch Configuration* option. This shows a window allowing to choose the type of launch configuration (figure 2.30). In next window (figure 2.31) specific launch configuration options for given launch configuration are displayed. The tab *ChipCraft Debug Server* allows setting additional options for debug server component responsible for communication between physical device and the debugger, which is a part of toolchain included in CC-SDK. The tab also allows



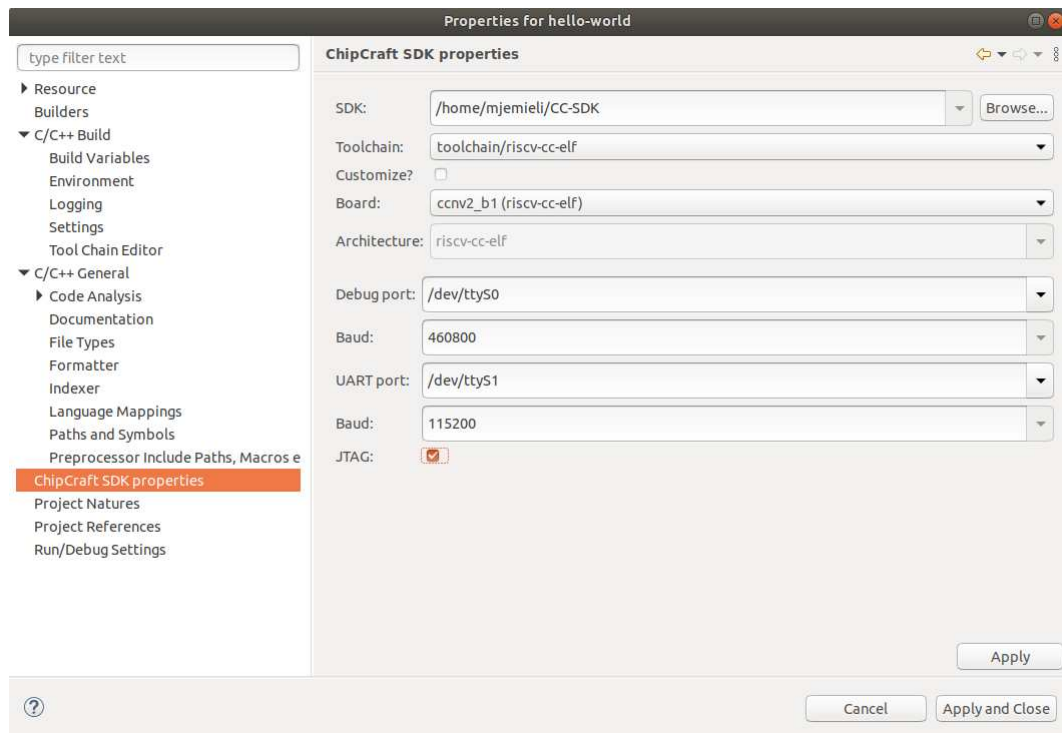


Figure 2.29. Project CC-SDK specific properties.

setting the Python executable to be used for running debug server scripts. Usually the defaults don't need to be changed.

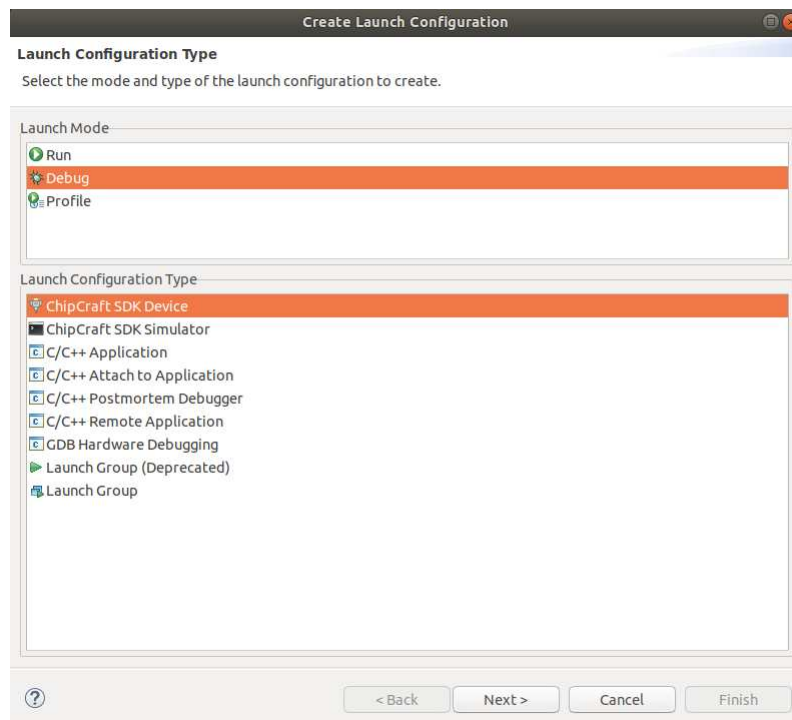


Figure 2.30. Launch configuration creation.

Launch configuration's *Startup* tab (figure 2.32) contains settings important for properly starting the debugging session and sets up the behaviour of target platform after debugger connects to it. The debugger initially tries to



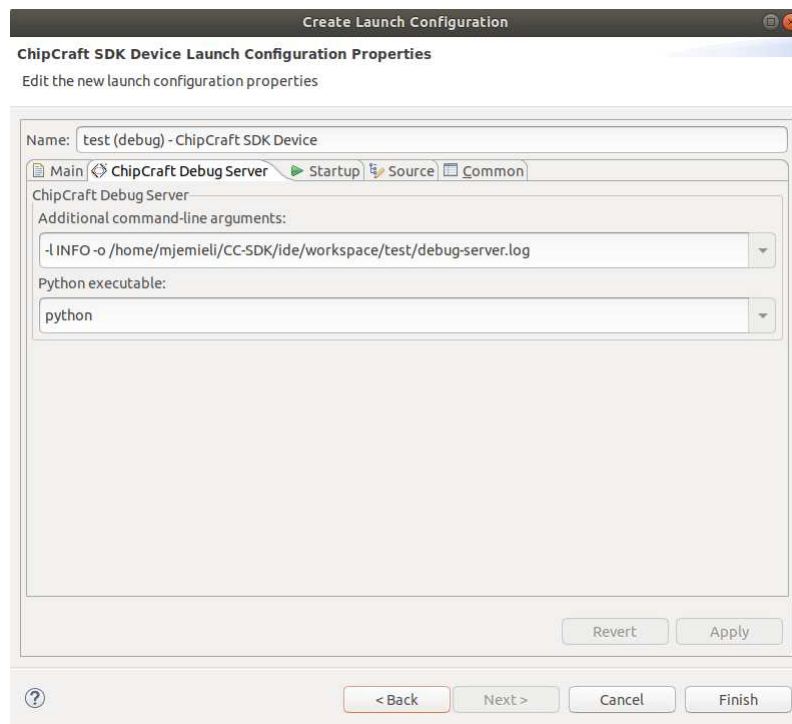


Figure 2.31. Launch configuration options.

set a breakpoint at address or symbol given in the *Run Commands* section. However, firmware execution may have already passed the given address, which means debugger will not stop where requested. By checking the *Reset and Delay* and *Halt* boxes in the *Initialization Commands* section, the user ensures that execution of the software will reset to start address and halt immediately after debugger connects, so it can then resume execution until requested initial breakpoint is hit. The debugger will initially pause execution whenever it connects to target platform. Checking the *Resume* box in the *Run Commands* section ensures the execution will automatically continue after debugger connects. The program can then be manually paused by the user.

Note that due to UART driver implementation in some target platforms, debugging a hardware device without JTAG will be unable to set up a breakpoint before certain function in firmware's startup routine is reached. To reflect that, new launch configurations for debugging a hardware device without JTAG by default disable the *Reset and Delay*, *Halt* and *Resume* options. If you know the hardware and firmware you use supports setting breakpoints before the main function is reached, check their respective boxes in launch configuration's *Startup* tab. Both simulator and JTAG allow the debugger to stop anywhere, so for those configurations these options remain enabled.

After launch configuration is created it can be executed by selecting **Debug/Run** button in launch configuration window or by selecting created launch configuration from context menu of **Debug/Run** buttons in the toolbar.

Launch configuration with default options can be started by using a shortcut. To do so click RMB on the project in *Project Explorer* window, then highlight **Debug As/Run As** submenu and select suitable configuration type (figure 2.33): *SDK application* uses a dedicated ASIC or FPGA board, while *SDK simulation* uses simulator software.

Please note that if project uses *sim* board, only the *ChipCraft SDK Simulator* configuration should be used. If project uses ASIC or FPGA board, only the *ChipCraft SDK Device* configuration should be used. Binary files produced



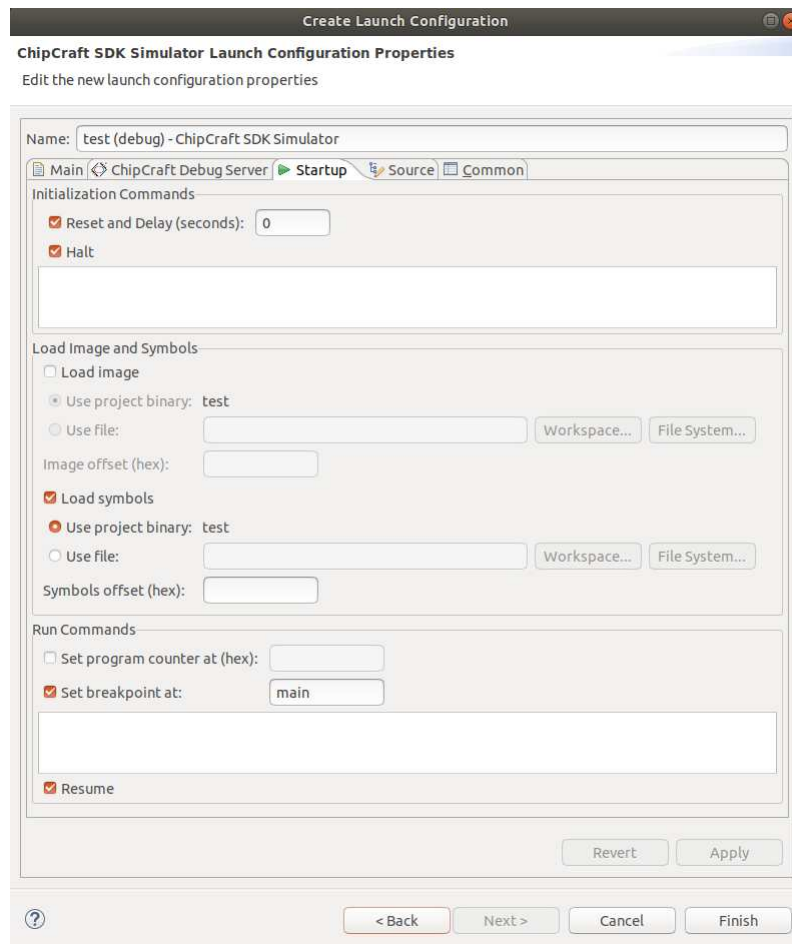


Figure 2.32. Launch configuration startup options.

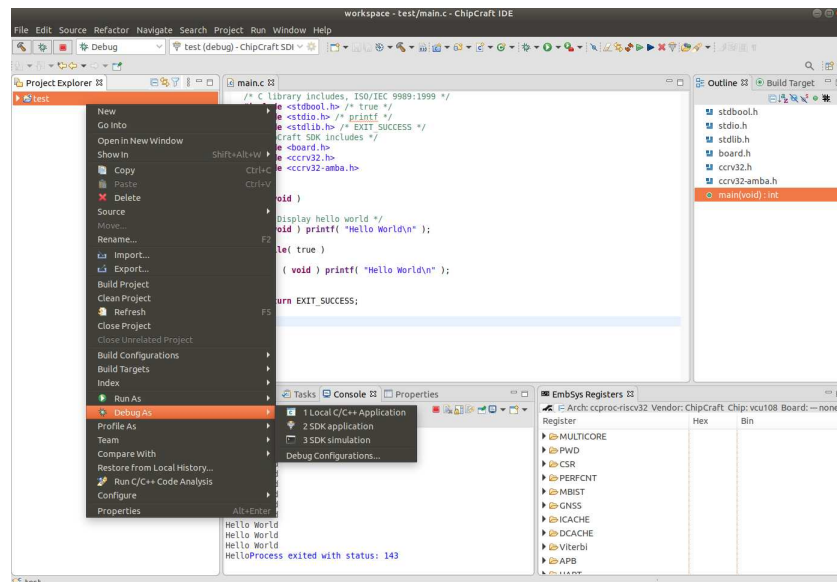


Figure 2.33. Debug launch shortcut.

for simulator are different than binary files produced for other boards so both launch configurations cannot be used interchangeably.



After debug configuration is launched, the IDE debug perspective is opened (figure 2.34). The IDE may ask about switching to new perspective the first time that happens. In **Debug** view (figure 2.35) all launches are visible in a tree. Applications utilizing multiple cores are displayed with a list of threads instead of cores. Every thread has associated stack-trace (if program is halted).

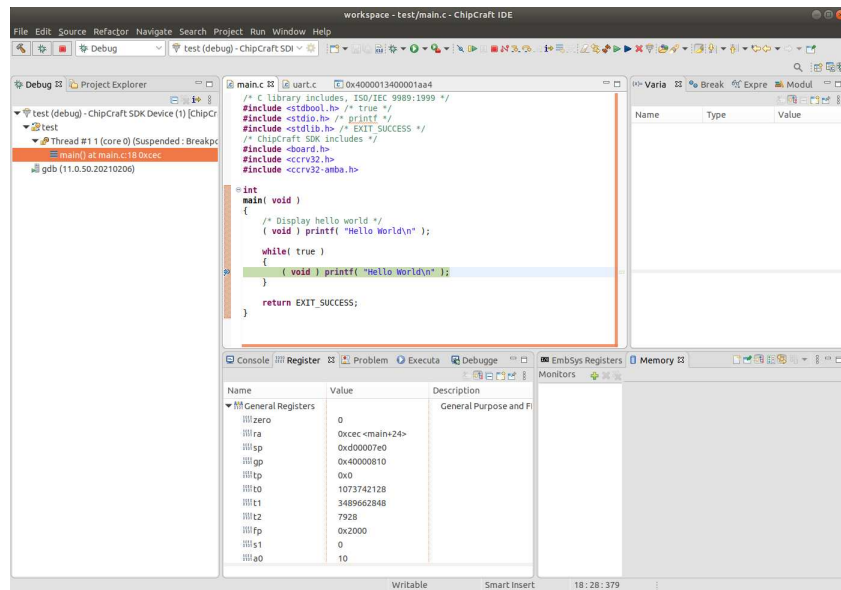


Figure 2.34. Main window during debugging.

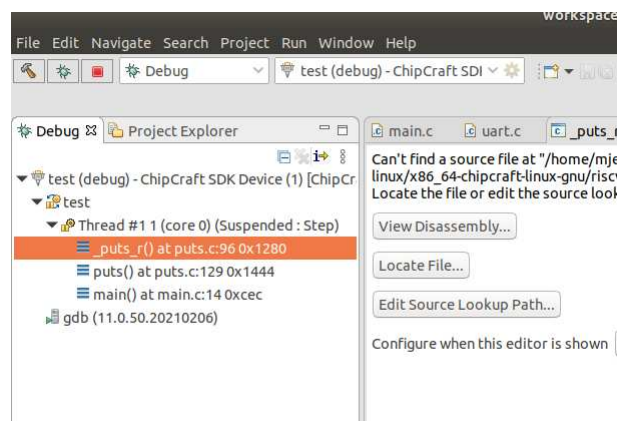


Figure 2.35. Debug process tree.

To access standard input/output streams from debugged program use the **Console** view. In case of *ChipCraft SDK Simulator* configuration type, the console with simulator output should already be visible. In case of *ChipCraft SDK Device* configuration, serial terminal has to be opened manually by clicking **Terminal** button in CC-SDK toolbar (figure 2.22). Console can be used both for reading program output and sending text to program input (figure 2.36).

EmbSys Registers view displays all registers of debugged device. Each register and in case of some registers groups of bits in registers can be modified independently.

CC-IDE utilizes a third-party plugin, *EmbSysRegView* to achieve this. The plugin settings are tied to project and automatically change to the board chosen during project creation. The settings will also automatically change when switching between projects using different boards. It's also possible to manually set the board used. To do this, users should click the wrench icon. Then in *EmbSysRegView* plugin properties a proper board can be selected.



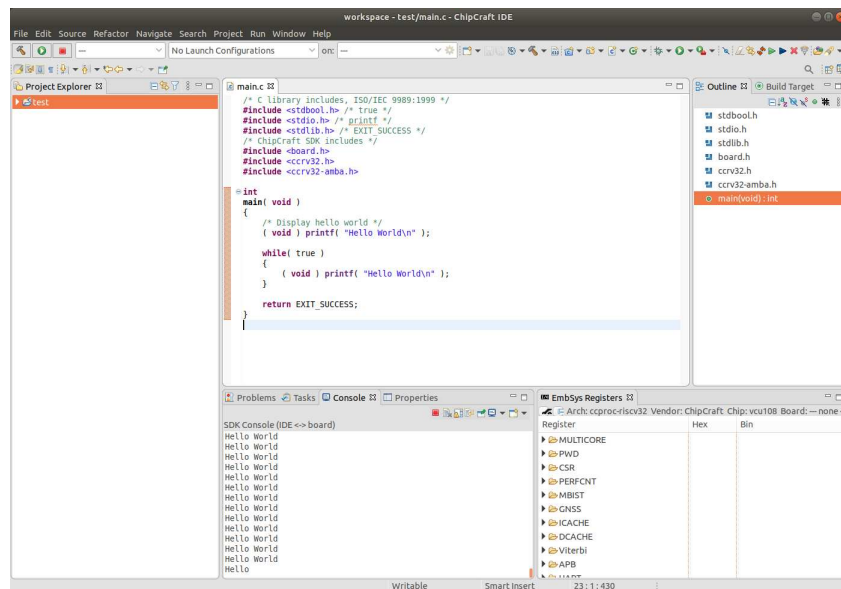


Figure 2.36. Debug UART console

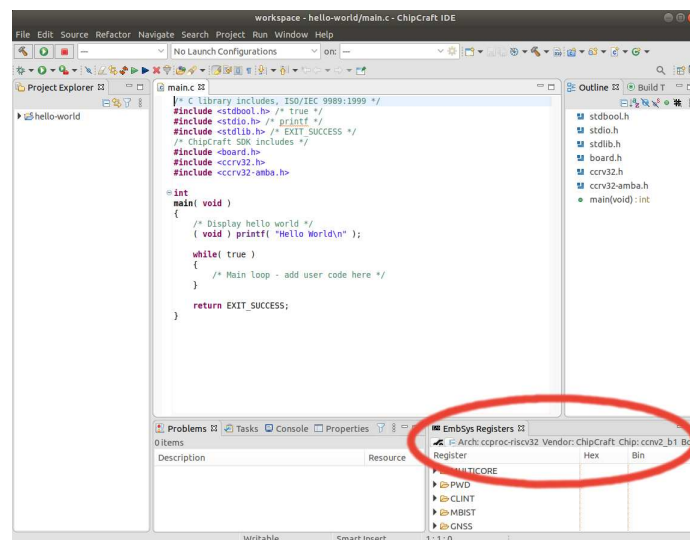


Figure 2.37. EmSysRegView plugin.

To load contents of a register user has to double click on the register itself or a containing element (that is, entire peripheral). Register values are displayed in red font if they changed after last program execution. To modify register content double click on register value cell and enter a new value. Note: it only works if program is already halted.

2.6 IDE preferences

To open CC-IDE preferences window select menu **Window -> Preferences**. Most of preferences are the same as in Eclipse CDT.



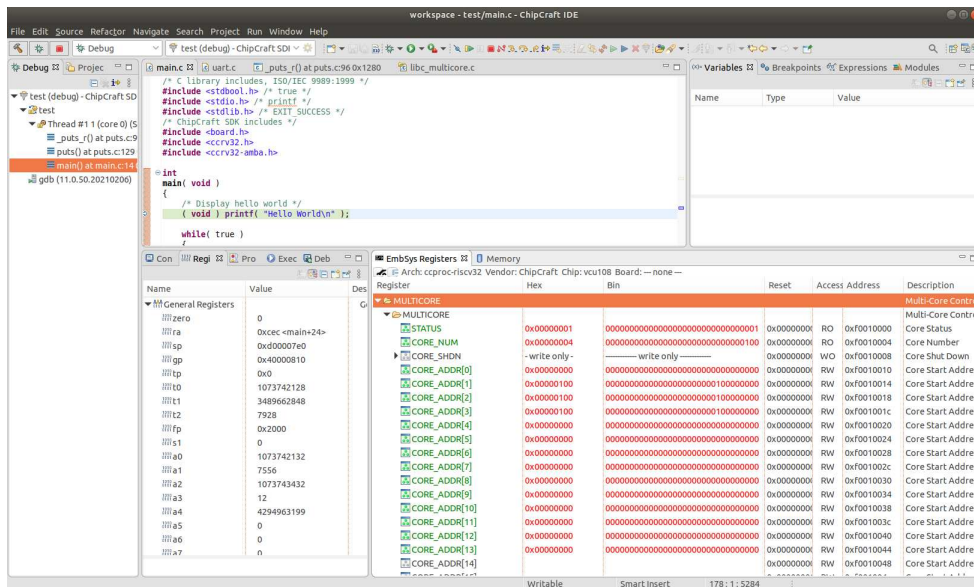


Figure 2.38. Registers view.

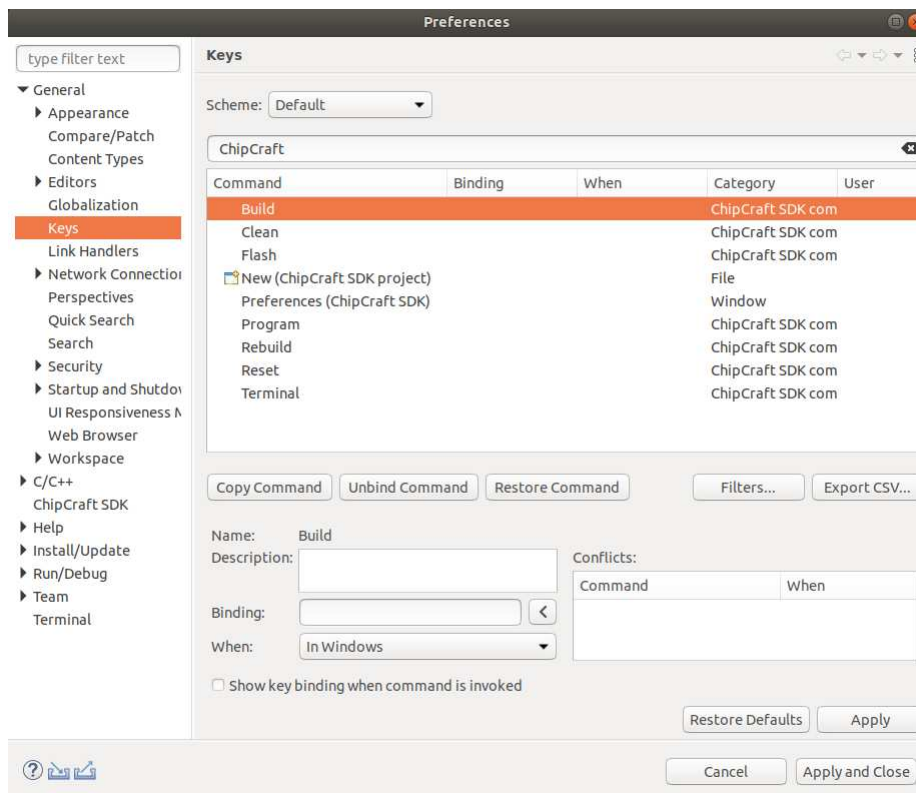


Figure 2.39. Hotkeys preferences.

In **General** -> **Keys** page (figure 2.39) you can configure hotkeys for all CC-IDE commands. CC-IDE specific commands are placed in a group named *ChipCraft SDK commands*. The commands can be filtered by the name of this group.

In **ChipCraft SDK** page you can configure logging level of custom ChipCraft components for Eclipse. This may be useful if facing some problems with the IDE itself. Setting log level to `DEBUG` will be most verbose, while `NONE` will



disable logging from ChipCraft components for IDE altogether. The logging level is reset to `WARNING` every time CC-IDE starts. Logs are written to standard Eclipse log file, which will be expanded upon in the next section.

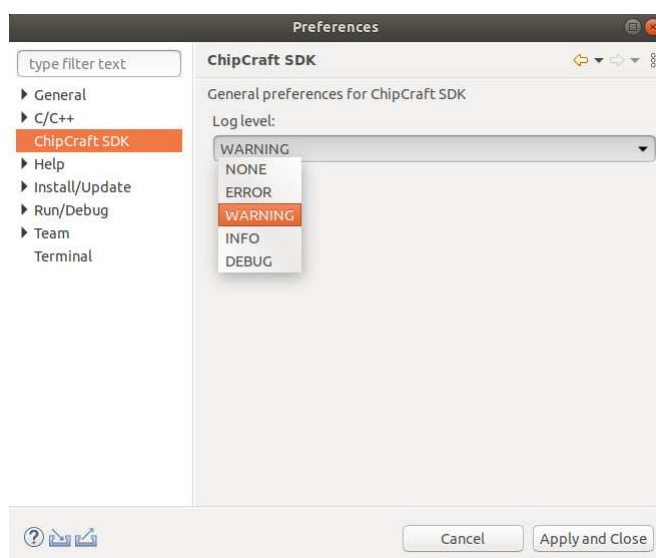


Figure 2.40. Global ChipCraft SDK preferences.



3. Troubleshooting CC-IDE

ChipCraft does its best to make sure CC-IDE works well on all supported platforms but not all hardware and software configurations are available for testing. Please report any bugs to ChipCraft to help make our product better. Information in this section can be useful to get more details about possible errors.

In case of errors Eclipse platform log file is worth checking for the cause. The easiest way to read its contents is to open **Help > About Eclipse Platform > Configuration Details**. This prints out a great number of details about the environment and also concatenates the log file. It is great for including in a bug report. The log file itself should also be accessible in the chosen workspace, in `.metadata/.log` file.

In case of debugging issues it is useful to enable GDB traces console. To do it go to **Window -> Preferences -> C/C++ -> Debug -> GDB**. Then activate **Show the GDB traces consoles with character limit** and set a high limit. After you save the changes there should be console named "gdb traces" available in Console View console selector during debugging session. All commands sent to GDB and all responses from GDB (possibly errors) should appear there.

If you still can't locate an issue happening during debugging you can enable debug mode of `dbgserver.py` script which is translating GDB commands to OCD commands. To do it go to launch configuration properties and ensure the following arguments are present in the debug server command line:

```
-l DEBUG
```

By default the debug level of `dbgserver.py` is set to `INFO`. Changing it to `DEBUG` makes it more verbose. Any errors that occur during initialization of `dbgserver.py`, before it can log any output, will be visible in Eclipse platform default log file, when global ChipCraft SDK log level is set to `DEBUG`.



4. Revision History

Doc. Rev.	Date	Comments
1.1	11-2022	Update due to new IDE release.
1.0	03-2019	First Issue.





ChipCraft Sp. z o.o.

Dobrzańskiego 3 lok. BS073, 20-262 Lublin, POLAND

www.chipcraft-ic.com

©2022 ChipCraft Sp. z o.o.

CC-IDE-UserGuide-Doc_171122.

ChipCraft®, ChipCraft logo and combination of thereof are registered trademarks or trademarks of ChipCraft Sp. z o.o. All other names are the property of their respective owners.

Disclaimer: ChipCraft makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. ChipCraft does not make any commitment to update the information contained herein.