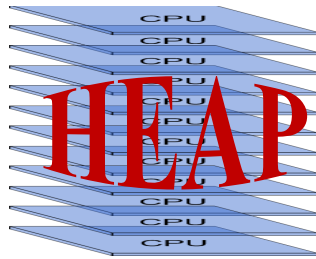


Information and Communication Technologies (ICT) Programme

Project N°: FP7-ICT- 247615

**HEAP**



---

Deliverable D3.4

## **Data dependency visualization tool**

User Manual and Tutorial

---

**Author(s):** Mihai T. Lazarescu (PoliTo) / Joeri van Ruth (ACE)

**Status -Version:** V1.4

**Date:** 23 January 2012

**Distribution - Confidentiality:** Public

**Code:** HEAP\_D3.4\_V1.4\_20120123

**Abstract:** In this deliverable there is a description of the data dependency visualization sub-toolset

© Copyright by the HEAP Consortium



## Disclaimer

This document contains material, which is the copyright of certain HEAP contractors, and may not be reproduced or copied without permission. All HEAP consortium partners have agreed to the full publication of this document. The commercial use of any information contained in this document may require a license from the proprietor of that information.

The HEAP Consortium consists of the following companies:

No	Participant name	Participant short name	Country	Country
1	ST Microelectronics	STM	Co-ordinator	Italy
2	Synelixis Solutions Ltd	Synelixis	Contractor	Greece
3	Thales Communications	Thales	Contractor	France
4	ACE Associated Compiler Experts B.V.	ACE	Contractor	Netherlands
5	Compaan Design	Compaan	Contractor	Netherlands
6	Politecnico Di Torino	PoliTo	Contractor	Italy
7	ATHENA Industrial Systems Institute	Athena	Contractor	Greece
8	Universita Degli Studi Di Genova	UniGe	Contractor	Italy
9	SingularLogic	SiLo	Contractor	Greece
10	Uppsala Universitet	Uppsala	Contractor	Sweden



## Document Revision History

Date	Issue	Author/Editor/Contributor	Summary of main changes
2011—09—19	1.1	Mihai T. Lazarescu	1 <sup>st</sup> draft. No deliverable no. yet
2011—09—20	1.2	Joeri van Ruth	2 <sup>nd</sup> draft. No deliverable no. yet
2011—12—14	1.3	Mihai T. Lazarescu	Updated for tracer written in C.
2012—01—21	1.4	Mihai T. Lazarescu	Dependency viewer and analysis tools details.
2012—01—23	1.4	Joeri van Ruth	Document edit and final review



## Table of contents

<b>1. Introduction .....</b>	<b>7</b>
<b>2. Tool Chain.....</b>	<b>7</b>
2.1. <i>Source Code Analysis and Instrumentation.....</i>	7
2.2. <i>Run-Time Data Dependency Tracer Library.....</i>	7
2.3. <i>Data Dependency Visualization.....</i>	8
2.4. <i>IDE.....</i>	9
<b>3. Demo Virtual Machine.....</b>	<b>10</b>
3.1. <i>Load the Demo Project.....</i>	10
3.2. <i>Run the Demo Analysis.....</i>	13
3.3. <i>Run the Data Dependency Visualization .....</i>	15
<b>4. New Project.....</b>	<b>18</b>
<b>5. NetBeans Project for HEAP extensions and ZGRViewer.....</b>	<b>20</b>
<b>6. Excerpts of Code::Blocks Documentation on Creation of a New Project .....</b>	<b>22</b>
6.1. <i>The project wizard .....</i>	22
6.2. <i>Changing file composition.....</i>	23
6.2.1. <i>Adding a blank file.....</i>	23
6.2.2. <i>Adding a pre-existing file.....</i>	25
6.2.3. <i>Removing a file .....</i>	25



## **Abbreviations**



## 1. Introduction

The open-source flow for the visualization of the execution parallelism provides:

- an IDE for the development of C language-based software projects
- a graphical visualization program that displays and allows the exploration of data dependencies, with automated cross-references to the source code in the IDE.

The companion tools for data dependency profiling are reported in D3.2:

- a program to analyse the developer C source code and to generate a functional model instrumented with code for data collection during program execution
- a library to analyse the data gathered during program execution, at run-time, and to generate a compact representation of the data dependencies between program instructions.

A free software virtual machine was configured with the whole chain as a means to achieve a consistent distribution able to demonstrate the tool functionality and receive valuable feedback for its further development.

## 2. Tool Chain

The tool chain and the virtual machine make use only of free software tools.

The changes to the tools as well as the virtual machine configuration provided are considered a beta release. Please provide feedback to improve it.

### 2.1. Source Code Analysis and Instrumentation

The source code analysis and instrumentation tool is based on the **CIL** platform<sup>1</sup> (**C** Intermediate Language). CIL is written in **ocaml**<sup>2</sup> and provides a high-level representation along with a set of tools that facilitate the analysis and the source-to-source transformations of C programs.

CIL is both lower-level than abstract-syntax trees, by clarifying ambiguous constructs and removing redundant ones, and also higher-level than typical intermediate languages designed for compilation, by maintaining types and a close relationship with the source program. The main advantage of CIL is that it compiles all valid C programs into a few core constructs with a very clean semantics. Also CIL has a syntax-directed type system that makes it easy to analyse and manipulate C programs. Furthermore, the CIL front-end is able to process not only ANSI-C programs but also those using Microsoft C or GNU C extensions.

A new code analysis and instrumentation module was written for the HEAP project. Some suitable existing CIL modules were merged and extended to implement the required functionality for code analysis and annotation.

### 2.2. Run-Time Data Dependency Tracer Library

The tracer library was written first in **Perl**<sup>3</sup> to allow fast prototyping of data structures and algorithms for analysis. Once the structure was consolidated, it was fully rewritten in C to reduce the run time

---

<sup>1</sup> <http://sourceforge.net/projects/cil/>

<sup>2</sup> <http://caml.inria.fr/ocaml/>

<sup>3</sup> <http://www.perl.org/>



(about 16x speed increase with respect to the Perl version and about **450x slower** than the normal application run).

The tracer library is linked with the instrumented application program under analysis to obtain an executable program. To perform the execution analysis, this program should be run using the same inputs as the normal (not annotated) application program. Data dependency is collected during program execution and at the end a summary file is generated that contains all the data needed to represent graphically the dependencies and the cross-references with the source program in the IDE.

### **2.3. Data Dependency Visualization**

This tool chain component is based on the free software program **ZGRViewer**.<sup>4</sup> It is a graph visualizer implemented in Java and based upon the **Zoomable Visual Transformation Machine**.<sup>5</sup>

ZGRViewer is specifically aimed at displaying graphs expressed using the DOT language from AT&T GraphViz and processed by programs *dot*, *neato* or others such as *twopi*. It is designed to handle large graphs and offers a zoomable user interface (ZUI), which enables smooth zooming and easy navigation in the visualized structure.

In the latest version it can provide:

- overview + detail views;
- focus+context magnification with Sigma Lenses views;
- graphical fish-eye focus+context distortion views;
- navigation along graph edges with Link Sliding;
- navigation from node to node with Bring & Go.

The tool chain includes the latest stable release (version 0.8.2), thus the features may differ from the latest development version.

Several classes were developed to integrate ZGRViewer with the HEAP tool chain. Its code and operation were analysed thoroughly to find the best way to integrate it to the HEAP flow to both simplify the integration, the debug, and maintenance of the integration as well as of the whole tool chain.

Special attention was given to the following objectives:

- limit as much as possible the changes to the original ZGRViewer code, both as entity as well as number. All changes to the original code were well tagged and documented to simplify future updates of the ZGRViewer code from the project (porting to newer versions of ZGRViewer)
- define a flexible format for the transfer of the data output by the tracer at the end of the annotated program execution. An XML template was defined such way to allow for easy data structure creation in the viewer and also be flexible enough to easily accommodate future extensions or other modifications
- define a bidirectional Inter-Process Communication subsystem to allow the communication of commands and data between the viewer and the IDE. The IPC is based on UNIX pipes and ASCII commands and data. It allows an easy integration with almost any IDE that is able to implement the other end of the IPC, reducing to a bare minimum the compatibility requirements between the viewer and the IDE

<sup>4</sup> <http://zvtm.sourceforge.net/zgrviewer/news.html>

<sup>5</sup> <http://zvtm.sourceforge.net/>





- a powerful graph driver for the GraphViz “dot” layout engine that generates an easy to understand dependency graph colourisation and a suitable node and arch layout in most cases
- an efficient cross-reference mechanism between the source code in the IDE and the nodes of the data dependency graph, including pop-up windows with data and keyboard short cuts to facilitate and speed up the graph exploration;
- an efficient internal data representation of the graph able to both deal with a large number of nodes and edges and also to simplify the graph transformation algorithms.

The HEAP-specific code was grouped under the “IDE” package with sub-packages for generic IPC classes, Code::Blocks-specific IPC classes, graph and IR classes, and utility classes. Both the ZGRViewer project and the HEAP additions make up a single project under the NetBeans IDE (see annex 5).

Most of the functions and graphic interface are presented in detail in the following sections.

## 2.4. IDE

The IDE functionality is provided by **Code::Blocks**.<sup>6</sup> Code::Blocks is a well established cross-platform IDE that supports projects in C/C++/D languages. It runs on Linux, OS X, and Windows platforms providing by design a consistent look, feel, and operation mode. It is written in C++ using the **wxWidgets**<sup>7</sup> library and is designed to be very extensible and fully configurable.

The IDE functionality was extended to suit the specific requests of the HEAP project, in particular the interface with the graph viewer based on the Java project ZGRViewer.

The recommended way to extend the functionality of the IDE is by writing plug-ins. The plug-ins can be written either in an IDE-specific scripting language or in C++, like its core. To decide what language to use it was considered that writing a script requires less time than writing C++ code, but the scripting interface did not allow to spawn threads, feature needed to handle the IPC with the view via UNIX pipes. Consequently, the plug-in had to be written in C++.

The plug-in implements all the functionality needed to integrate the ZGRViewer-based graph viewer into customized for HEAP in the Code::Blocks IDE to offer the designer a **single interface** for controlling the development and optimisation of the project. To this end, the plug-in extends the IDE functionality with:

- ability to create the communication pipes and to spawn the viewer process upon the developer command using a specific HEAP menu
- automatically connect to the viewer process using the pipes to establish the first contact with the viewer after its initialization
- implement the IPC listener and transmitter, and the IPC protocol for exchanging commands and data with the viewer
- implementing the interaction with the IDE elements (e.g., the editors, the menus, the project information) necessary to provide the developer the needed visual feedbacks, such as:
  - ⤴ open and select the line requested by the developer through the viewer interface (for instance to display the line represented by a graph node)
  - ⤴ retrieve and send to the viewer the number required of context lines for a given source code line that are used to display on the graph the source code context for a given node

<sup>6</sup> <http://www.codeblocks.org/>

<sup>7</sup> <http://www.wxwidgets.org/>



- ⤴ provide the block folding information for all project files to the viewer to collapse the corresponding nodes
  - ⤴ send the viewer the refresh command upon developer click on the “update” item in the HEAP menu
- ability to end the viewer process and remove the communication pipes upon developer command using the HEAP menu
- interact with the IDE graphical elements, such as main and context menus.

Most of these functionalities and graphical items of the HEAP plug-in are presented in detail in the following sections.

### 3. Demo Virtual Machine

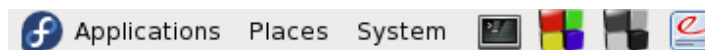
A **Linux**<sup>8</sup> **VirtualBox**<sup>9</sup> virtual machine (VM) was configured to reliably support the functionality of the tool chain. Its installation is described in D3.2, where the reader is now referred.

The users defined on the virtual machine are:





- **root** with the password: **Demo11HEAP**  
This login can be used to perform administration tasks on the VM, if required.
- **heapdemouser** with the password: **Demo11HEAP**  
This login is used for all tool chain-related activities.

Demo Project


The buttons to launch the applications of interest are exposed for convenience on the top panel of the workspace, right next to Fedora menus:



From left to right, they are:

-  opens a terminal window;
-  opens Code::Blocks IDE;
-  discards a hanged instance of the Code::Blocks IDE;
-  displays the user manual of the distribution.

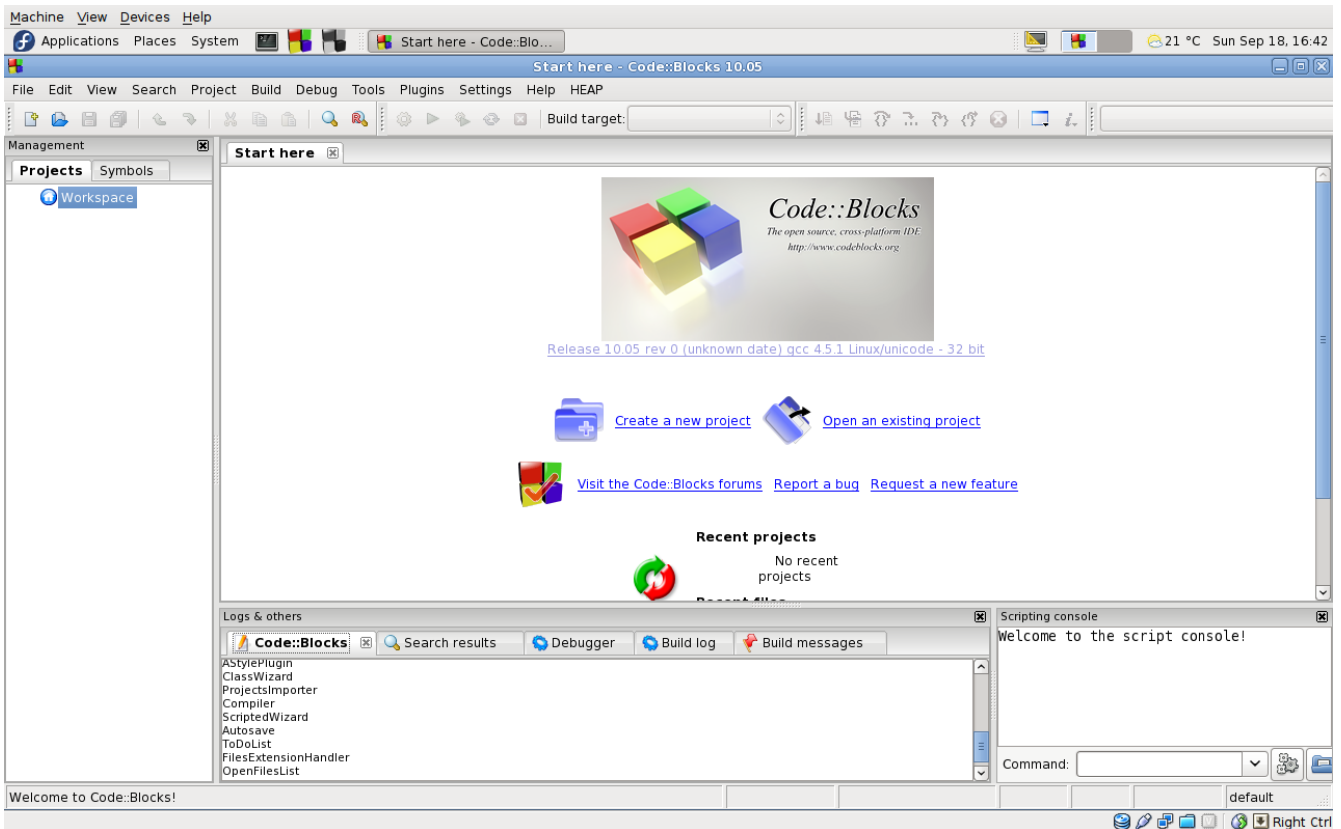
#### 3.1. Load the Demo Project

Click on the Code::Blocks button () to start the IDE:

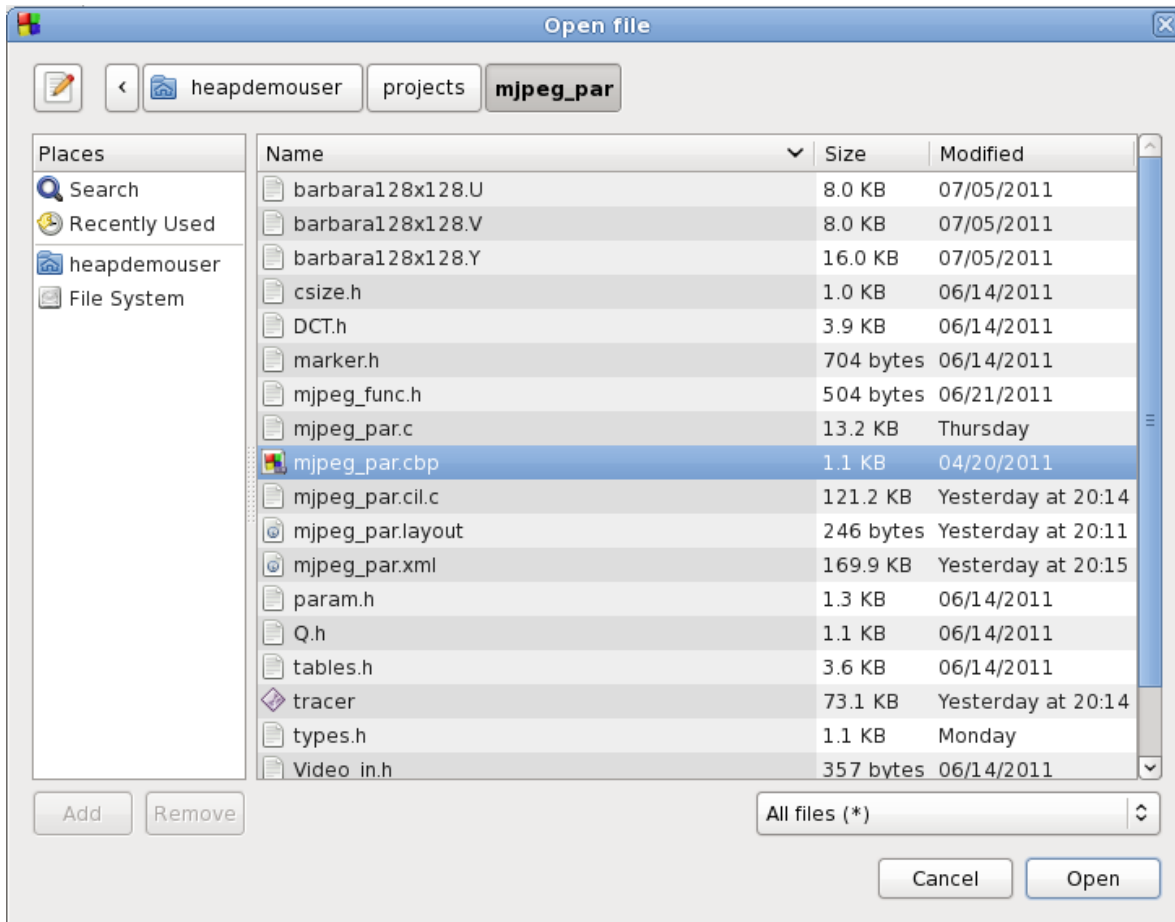
<sup>8</sup> <https://secure.wikimedia.org/wikipedia/en/wiki/Linux>

<sup>9</sup> <http://www.virtualbox.org/>

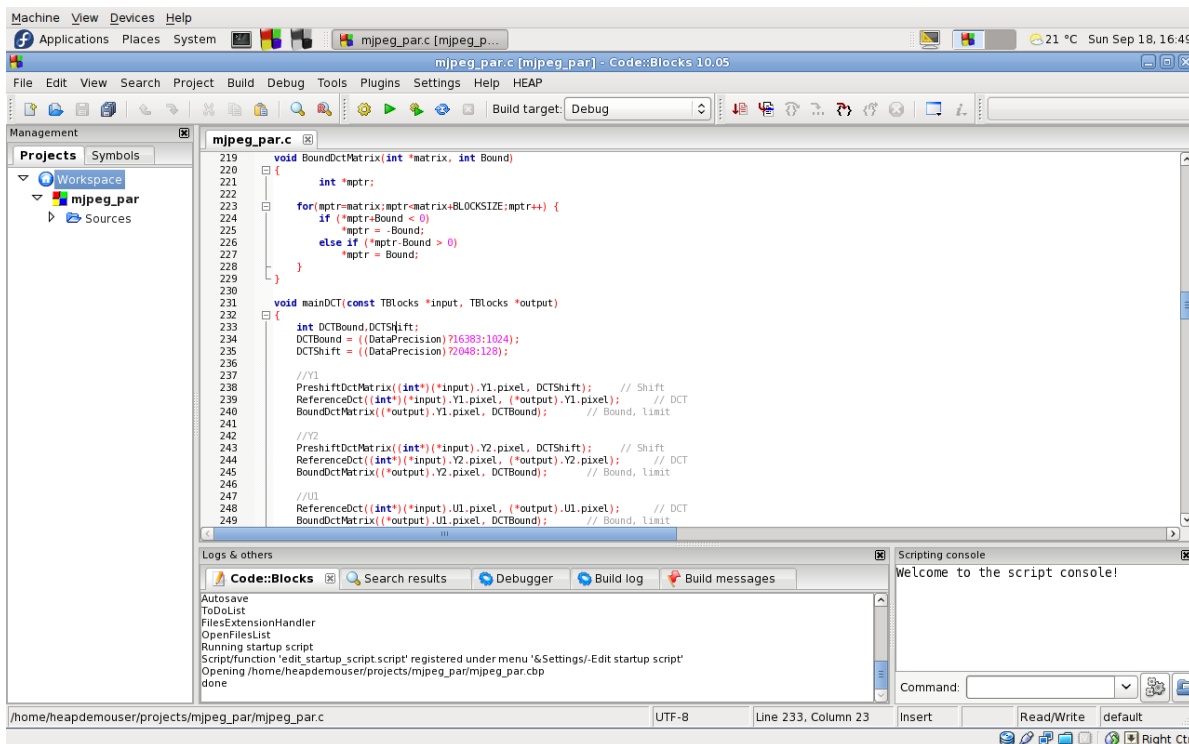
FP7-ICT-247615 - HEAP  
Free Software-Based Flow for the Visualization of the  
Parallelism in the Program Execution -- User Manual and Tutorial



Select “**File**” from the top menu, then click on “**Open...**”. In the file chooser window that opens navigate to “**heapdemouser/projects/mjpeg\_par**”, select “**mjpeg\_par.cbp**” and click on “**Open**”:



The “mjpeg\_par” project will open:

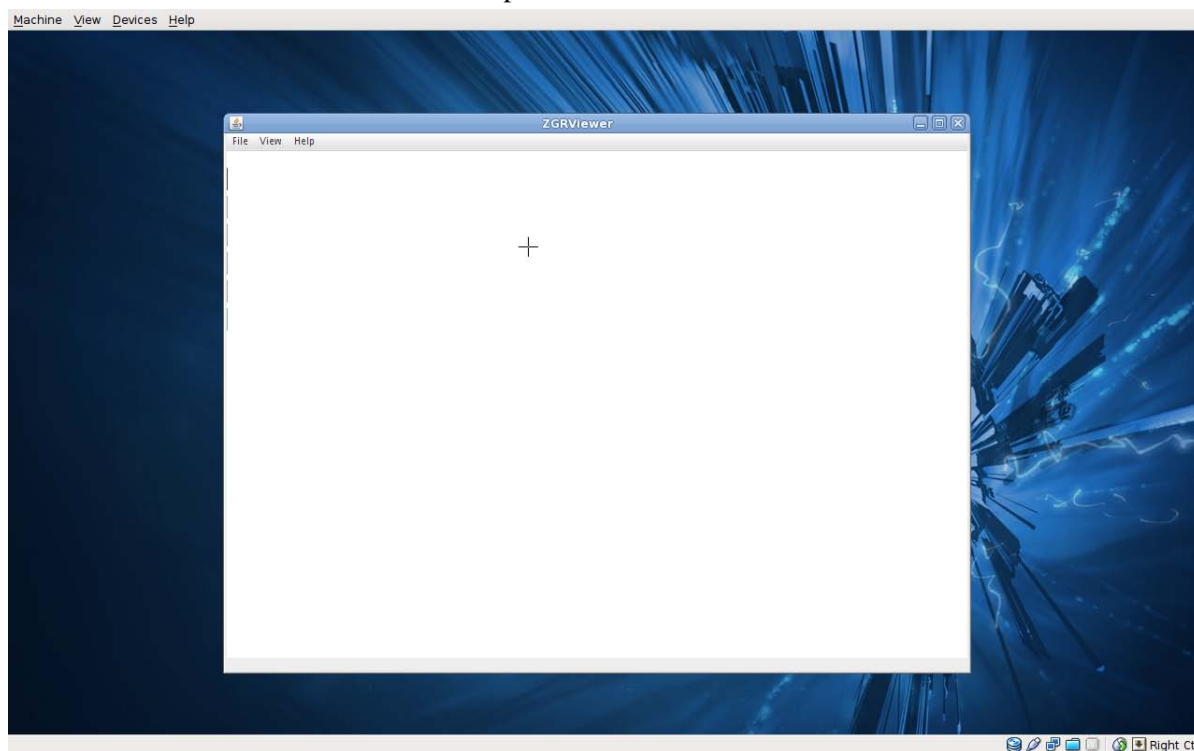




Now start the visualization program by clicking on the “HEAP” entry of the top menu and then on “Run”:



The ZGRViewer visualizer window will open:



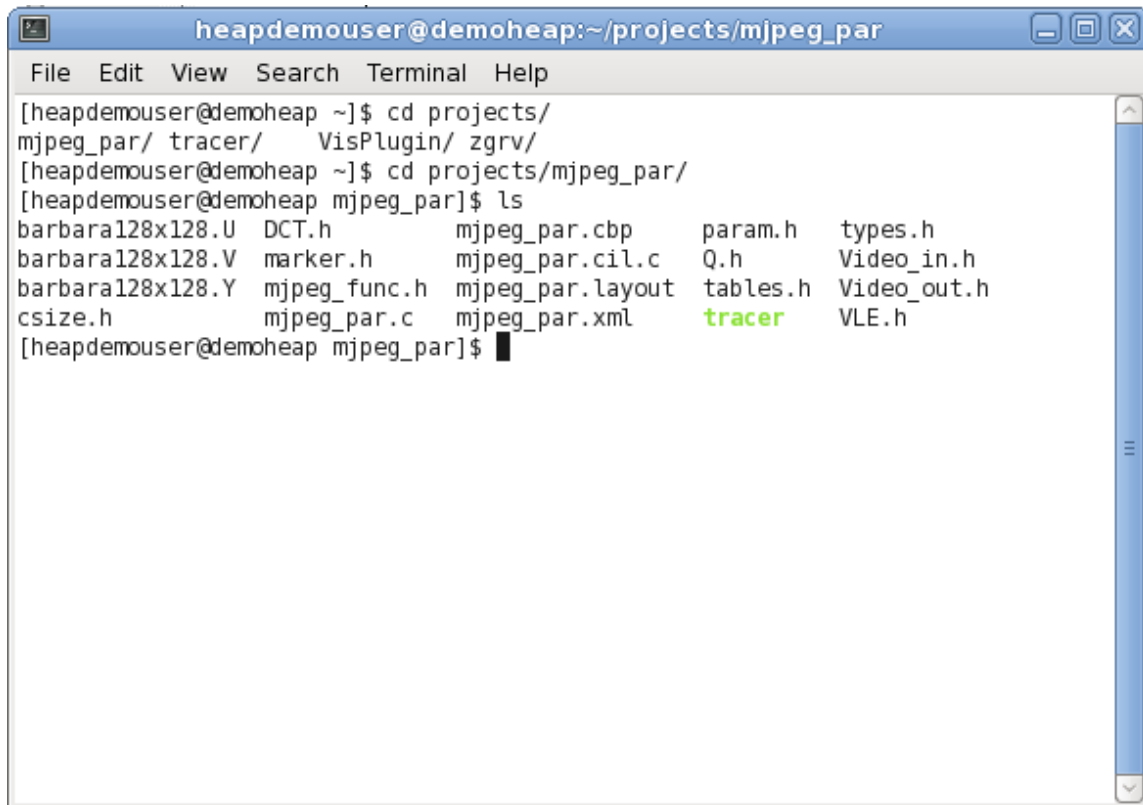
Arrange the IDE and the ZGRV windows on the screen to have a clear view of both. If you have two monitors attached to the host machine you may wish to move the ZGRV window on the second monitor of the VM and then move this VM second monitor window on the second physical monitor of the host.

### 3.2. Run the Demo Analysis

The analysis tool chain is run from the command line. A script is provided that loosely glues together the whole chain.

*Note: the instrumented program runs about **450 times slower** than the native run.*

Open a terminal window by clicking on the icon in the top panel of the workspace and go into the directory of the `mjpeg_par` project of the IDE:

A screenshot of a terminal window titled "heapdemouser@demoheap:~/projects/mjpeg\_par". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal shows the following commands and output:

```
[heapdemouser@demoheap ~]$ cd projects/  
mjpeg_par/ tracer/ VisPlugin/ zgrv/  
[heapdemouser@demoheap ~]$ cd projects/mjpeg_par/  
[heapdemouser@demoheap mjpeg_par]$ ls  
barbara128x128.U DCT.h mjpeg_par.cbp param.h types.h  
barbara128x128.V marker.h mjpeg_par.cil.c Q.h Video_in.h  
barbara128x128.Y mjpeg_func.h mjpeg_par.layout tables.h Video_out.h  
csize.h mjpeg_par.c mjpeg_par.xml tracer VLE.h  
[heapdemouser@demoheap mjpeg_par]$
```

In this directory run the **tracer.sh** script with arguments:

```
tracer.sh -- mjpeg_par.c
```

where:

- -- (double dash) ends the command line options that are passed to the compiler and linker;
- **mjpeg\_par.c** is the name of the source file to analyse:



```

heapdemouser@demoheap:~/projects/mjpeg_par
File Edit View Search Terminal Help
[heapdemouser@demoheap ~]$ cd projects/mjpeg_par/
[heapdemouser@demoheap mjpeg_par]$ ls
barbara128x128.U  DCT.h          mjpeg_par.cbp  param.h        types.h
barbara128x128.V  marker.h       mjpeg_par.cil.c  Q.h           Video_in.h
barbara128x128.Y  mjpeg_func.h  mjpeg_par.layout  tables.h      Video_out.h
csize.h          mjpeg_par.c   mjpeg_par.xml   tracer        VLE.h
[heapdemouser@demoheap mjpeg_par]$ tracer.sh -- mjpeg_par.c
+ /home/heapdemouser/projects/tracer/cil-1.4.0/bin/cilly --save-temps --noWrap -
-noPrintLn --dooneRet --dosimplify --doimarw -c mjpeg_par.c
gcc -D_GNUCC -E -DCIL=1 mjpeg_par.c -o ./mjpeg_par.i
/home/heapdemouser/projects/tracer/cil-1.4.0/obj/x86_LINUX/cilly.asm.exe --out .
./mjpeg_par.cil.c --noWrap --noPrintLn --dooneRet --dosimplify --doimarw ./mjpeg_
par.i
gcc -D_GNUCC -E ./mjpeg_par.cil.c -o ./mjpeg_par.cil.i
gcc -D_GNUCC -c -o ./mjpeg_par.o ./mjpeg_par.cil.i
+ rm -f mjpeg_par.i mjpeg_par.o mjpeg_par.cil.i
+ gcc mjpeg_par.cil.c -lavl -lxml2 -lheap -o tracer
+ rm -f mjpeg_par.cil.o
+ ./tracer
W: no arg 1 for instruction 19 (main())
W: no arg 2 for instruction 20 (main())
+ test -s model.xml
+ test model.xml = mjpeg_par.xml
+ mv model.xml mjpeg_par.xml
[heapdemouser@demoheap mjpeg_par]$

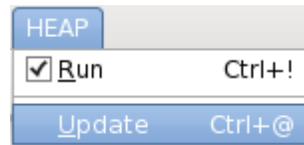
```

where:

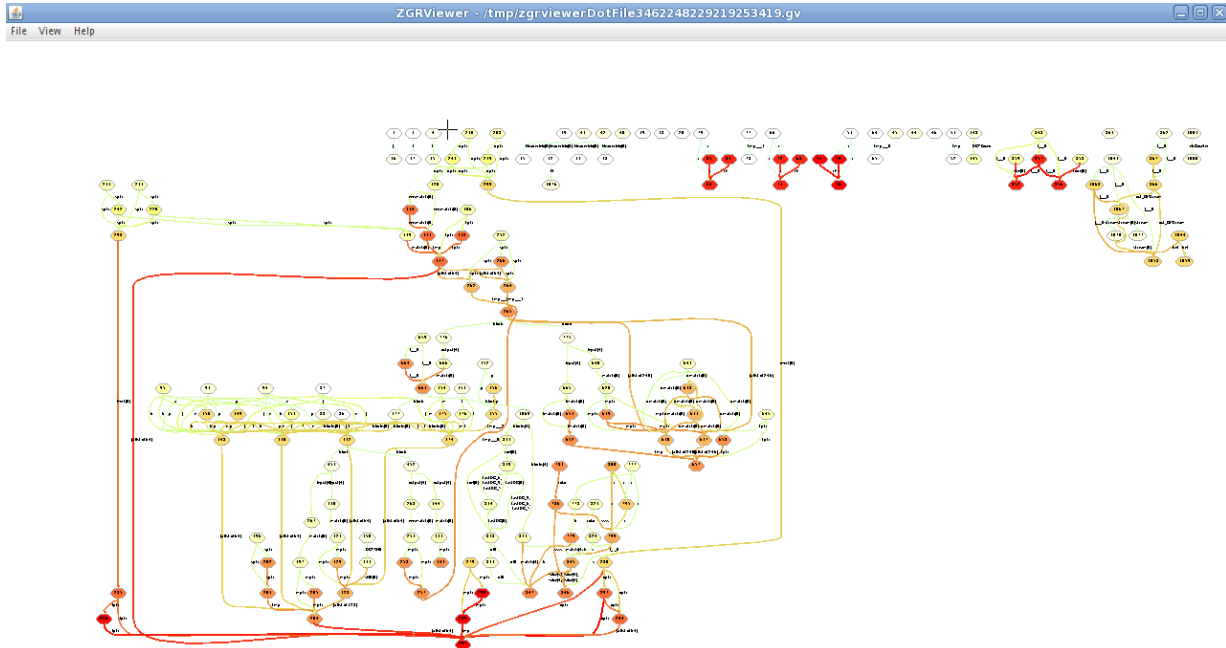
- + **/home/heapdemouser/projects/tracer/cil-1.4.0/bin/cilly** is the starting command for CIL compilation
- the three **gcc** compilations that follow are part of the *cilly* run and generate the instrumented model of the user program, **mjpeg\_par.cil.c**
- + **rm -f mjpeg\_par.i mjpeg\_par.o mjpeg\_par.cil.i** cleans the temporary files from the directory
- the next **gcc** run compiles the CIL model (mjpeg\_par.cil.c) and links it with the data dependency tracer library (libheap) and other system libraries (libxml2, libavl)
- the **rm** command cleans the temporary files from the directory
- the data dependency tracer is then run. It actually runs the user program instrumented for data dependency tracing together with the data dependency tracer
- finally, the **mv** command renames the file with the generated data to the name expected by the ZGRViewer-based visualizer.

### 3.3. Run the Data Dependency Visualization

After each operation that can affect the visualization (e.g., change the folding in the IDE editor, update the visualizer data) the visualizer should be informed on the update. Access the HEAP menu on the top menu of the IDE and click on the “**Update**” entry:



Notice how the visualizer window displays the data dependency among program instructions:



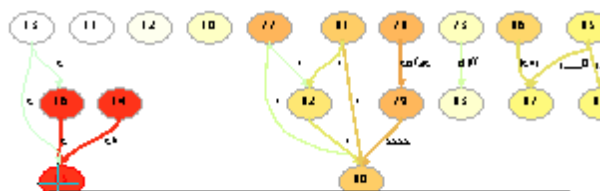
Each ellipse represents a program instruction that was executed. The ellipse colour can vary from white (seldom executed) to intense red (most executed).

Each directed arch that connects two ellipses represents a data dependency between the two instructions. The arch colour can vary from light cyan (for seldom occurring dependencies) to intense red (for most occurring dependencies) and, at the same time, the arch width is modulated by the same factor, the widest for the most occurring.

The visualizer implements a few handy short cuts:

- 'c' -- with the cursor on a node, display the source code of the node with 5 context lines:





```

if(fh1 == NULL) {
    fh1 = fopen("barbara128x128.Y", "r");
    c = 0;
    while ((ch = getc(fh1)) != EOF) {
        compY[c] = ch;
        C++;
    }
}

if(fh2 == NULL) {

```

- ‘C’ -- with the cursor on a node, display the source code of the node with 10 context lines:



```

// open image files only the first time when mainVideoIn is called
// and put them in arrays
if (isFirst == 1) {
    isFirst = 0;

    if(fh1 == NULL) {
        fh1 = fopen("barbara128x128.Y", "r");
        c = 0;
        while ((ch = getc(fh1)) != EOF) {
            compY[c] = ch;
            C++;
        }
    }

    if(fh2 == NULL) {
        fh2 = fopen("barbara128x128.U", "r");
        c = 0;
        while ((ch = getc(fh2)) != EOF) {
            compU[c] = ch;
            C++;
        }
    }
}

```

- ‘e’ -- with the cursor on a node, move the IDE editor cursor on the source line corresponding to the node;
- ‘m’ -- with the cursor on an arch, display the unabridged list of data dependencies represented by the arch:



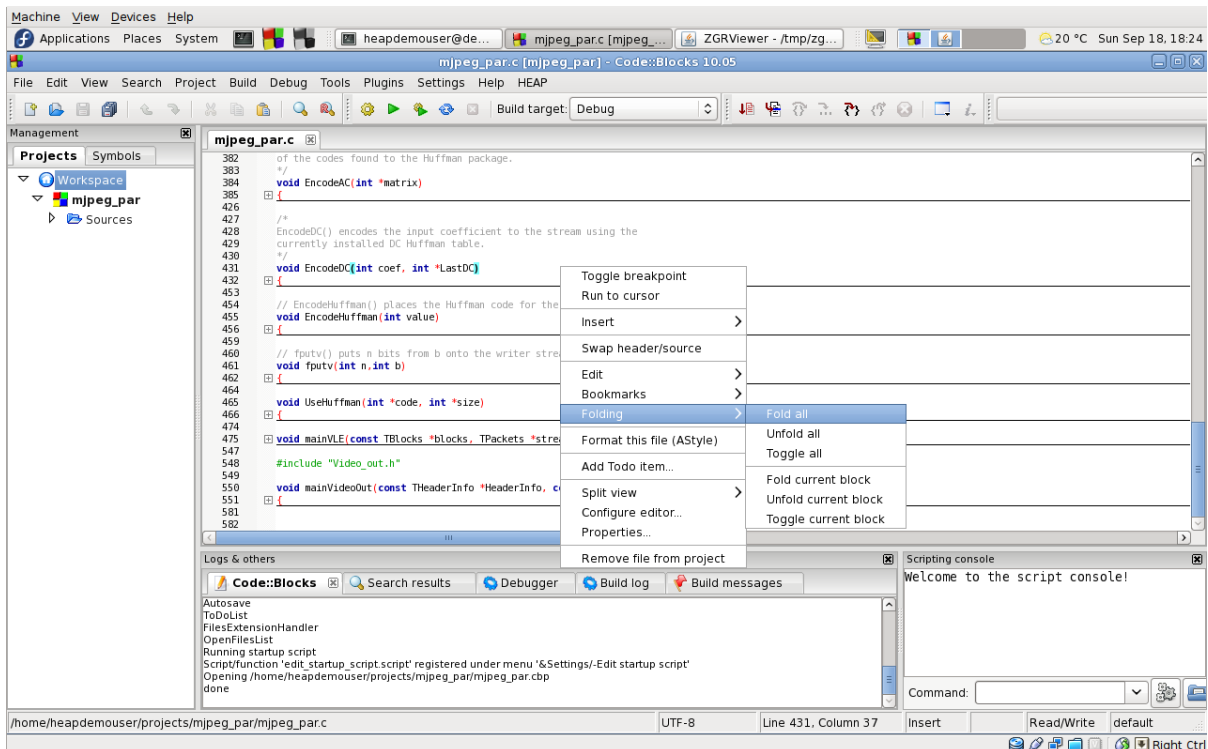
```

sourcematrix, sourcematrix[10], sourcematrix[11], sourcematrix[12],
sourcematrix[13], sourcematrix[14], sourcematrix[15], sourcematrix[16],
sourcematrix[17], sourcematrix[18], sourcematrix[19], sourcematrix[1],
sourcematrix[20], sourcematrix[21], sourcematrix[22], sourcematrix[23],
sourcematrix[24], sourcematrix[25], sourcematrix[26], sourcematrix[27],
sourcematrix[28], sourcematrix[29], sourcematrix[2], sourcematrix[30],
sourcematrix[31], sourcematrix[32], sourcematrix[33], sourcematrix[34],
sourcematrix[35], sourcematrix[36], sourcematrix[37], sourcematrix[38],
sourcematrix[39], sourcematrix[3], sourcematrix[40], sourcematrix[41],
sourcematrix[42], sourcematrix[43], sourcematrix[44], sourcematrix[45],
sourcematrix[46], sourcematrix[47], sourcematrix[48], sourcematrix[49],
sourcematrix[4], sourcematrix[50], sourcematrix[51], sourcematrix[52],
sourcematrix[53], sourcematrix[54], sourcematrix[55], sourcematrix[56],
sourcematrix[57], sourcematrix[58], sourcematrix[59], sourcematrix[5],
sourcematrix[60], sourcematrix[61], sourcematrix[62], sourcematrix[63],
sourcematrix[6], sourcematrix[7], sourcematrix[8], sourcematrix[9]
    
```

- ‘g’ -- best fit of the graph on screen;
- use the **mouse wheel** to zoom in/out.

To further facilitate the exploration of the data dependencies, the graph nodes can be folded by folding in the IDE the lines corresponding to the nodes source lines. For instance, by folding all blocks in the IDE we obtain the dependency view between the functions:

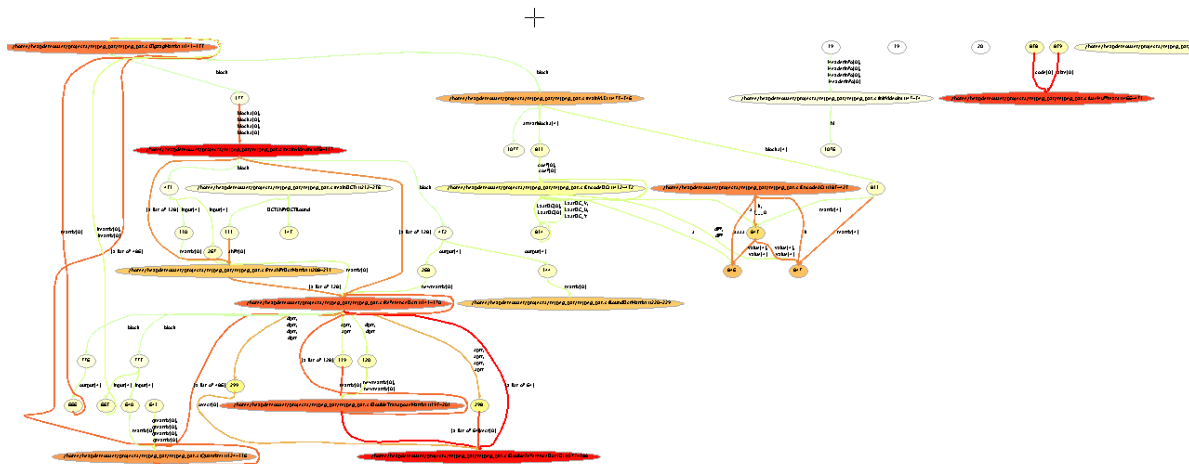
- fold all the blocks in the IDE:



- update the visualizer:



ZGRViewer - /tmp/zgrviewerDotFile3462248229219253419.gv



The nodes still visible are the function arguments.

## 4. New Project

Creating a new project for data dependency analysis requires the following steps:

1. create a new project under the IDE and populate it with the project source files (please refer to the Code::Blocks documentation<sup>10</sup> for detailed explanation and to annex 6).

Unless you are going to use Code::Blocks for development, importing a project for data dependency analysis is usually just a matter of copying the project file tree under the project directory using standard GUI or command line tools

2. make sure the project compiles well with the native compiler (gcc), runs and produces the proper results
3. apply the data dependency analysis as described in section 3.2;
4. visualize the data dependency results as described in section 3.3.

An important aspect to consider when creating a new IDE project is its location. If the project directory is set on the VM virtual disk it can be lost with the next updates of the VM. It is safer to create the project on the host file system that is accessible to the host OS using the shared folders configuration in section 3.

<sup>10</sup> [http://wiki.codeblocks.org/index.php?title=Creating\\_a\\_new\\_project](http://wiki.codeblocks.org/index.php?title=Creating_a_new_project)



## Annexes

### 5. NetBeans Project for HEAP extensions and ZGRViewer

The HEAP-specific classes are expanded in the left panel.

The ZGRViewer project is unexpanded, under the `net.claribole.zgrviewer` and `net.claribole.zgrviewer.dot` packages.

The external ZGRViewer libraries and utilities are located under the “Other Sources” and “Libraries” folders.



File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

assertions

Pro... x Files Services ...va ipc.java x ipcCallbacks.java x XRef.jav

ZGRViewer

- Source Packages
  - IDE.IPC
    - ipc.java
    - ipcInput.java
    - ipcOutput.java
  - IDE.IPC.CB
    - CBContext.java
    - CBDisplay.java
    - CBFold.java
    - CBQuit.java
    - CBSimStats.java
    - ipcCallback.java
    - ipcCallbacks.java
  - IDE.Util
    - RGB.java
  - IDE.graph
    - Display.java
    - DisplayDOT.java
    - Fold.java
    - Folds.java
    - GEdge.java
    - GEdgeStats.java
    - GNode.java
    - GNodeStats.java
    - IR.java
    - NodeXRef.java
    - ToolTip.java
    - XRef.java
- net.claribole.zgrviewer
- net.claribole.zgrviewer.dot
- Other Sources
- Generated Sources (antlr)
- Libraries
- Project Files

```

Integer hits = Integer.parseInt(attr.

// Create and record a new edge
// with the given attributes.
//
GNode from = getNodeForName(sourceNode
assert from != null;

GNode to = getNodeForName(sinkNodeId)
assert to != null;

// Assume all edges are directed for
GEdge edge = new GEdge(from, to, addr

from.addEdge(edge);
to.addEdge(edge);
}

/**
 * Annotate the <transfers> tag contents
 * of an IDE-provided simulation results.
 *
 * @param transfers
 *       <transfers> tag to annotate.
 */
private void annotateProjectTransfersIr(Node project)
{
    assert transfers != null;

    for (Node node = getFirst(transfers);
         node != null; node = node.getNext())
        if (node.getNodeName().equalsIgnoreCase("transfers"))
            annotateProjectTransfersTransfersIr(node);
        // Silently ignore anything else.
}

/**
 * Annotate the <project> tag contents
 * of an IDE-provided simulation results.
 *
 * @param project
 *       <project> tag to annotate.
 */
private void annotateProjectIr(Node project)
{
    assert project != null;

```

Usages Output Tasks Java Call Hierarchy Search Results

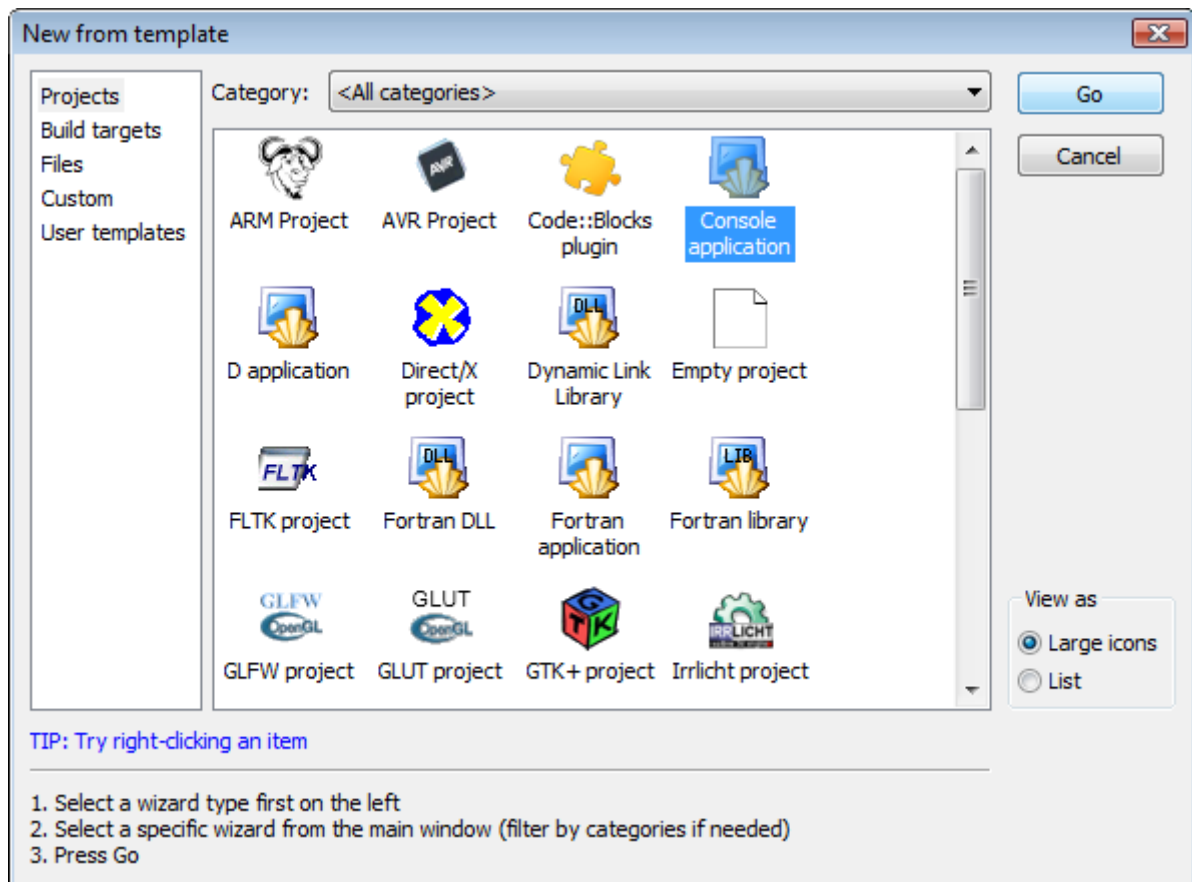


## 6. Excerpts of Code::Blocks Documentation on Creation of a New Project

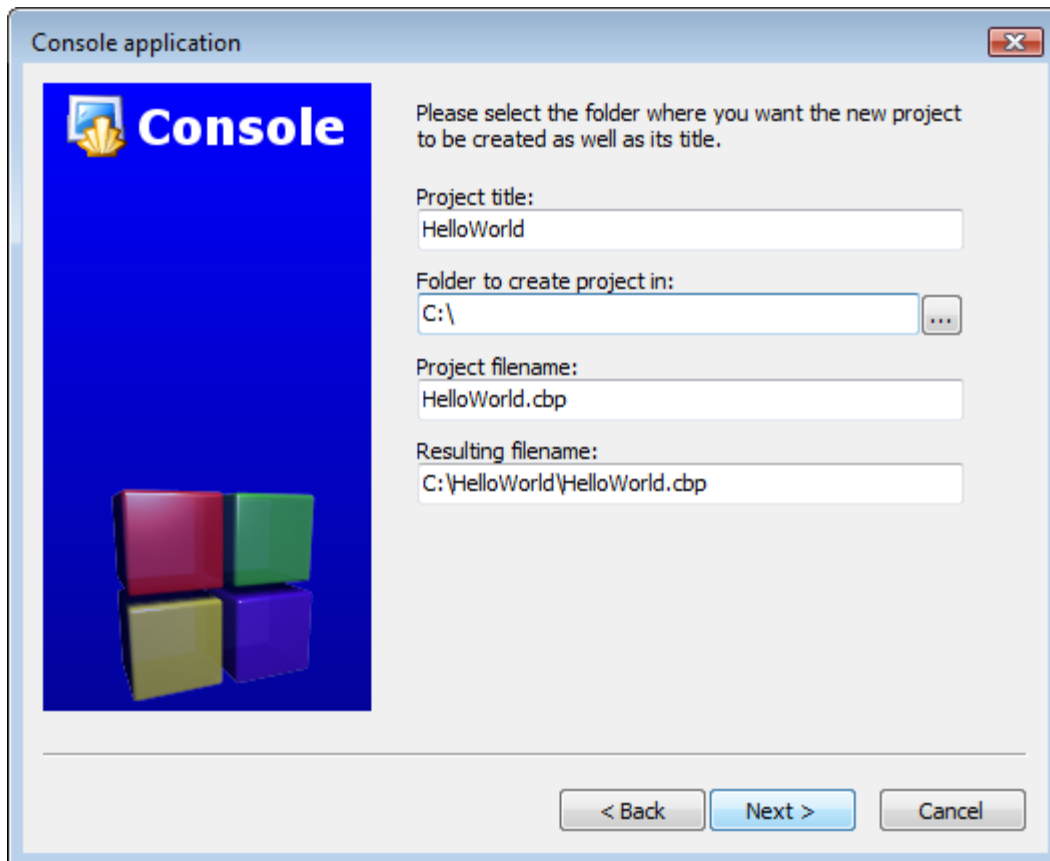
This section is a guide to many of the beginning (and some intermediate) features of the creation and modification of a Code::Blocks project. If this is your first experience with Code::Blocks, here is a good starting point.

### 6.1. The project wizard

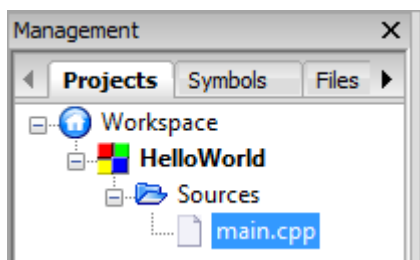
Launch the Project Wizard through `File->New->Project...` to start a new project. Here there are many pre-configured templates for various types of projects, including the option to create custom templates. Select Console application, as this is the most common for general purposes, and click Go.



The console application wizard will appear next. Continue through the menus, selecting C++ when prompted for a language. In the next screen, give the project a name and type or select a destination folder. As seen below, Code::Blocks will generate the remaining entries from these two.



Finally, the wizard will ask if this project should use the default compiler (normally GCC) and the two default builds: Debug and Release. All of these settings are fine. Press finish and the project will be generated. The main window will turn gray, but that is not a problem, the source file needs only to be opened. In the Projects tab of the Management pane on the left expand the folders and double click on the source file main.cpp to open it in the editor.



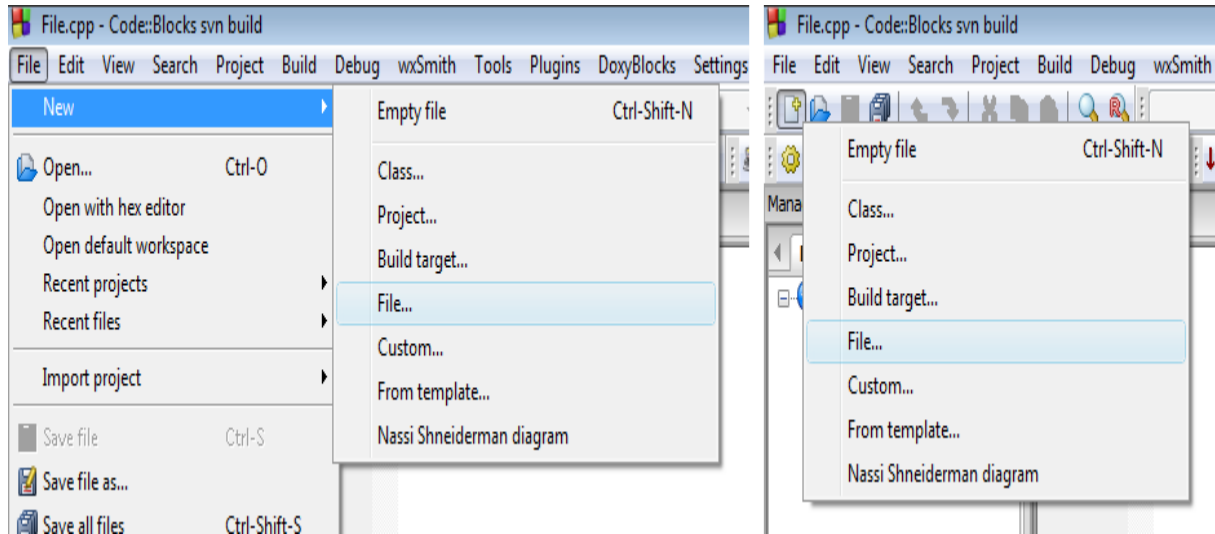
This file contains some default code.

## **6.2. Changing file composition**

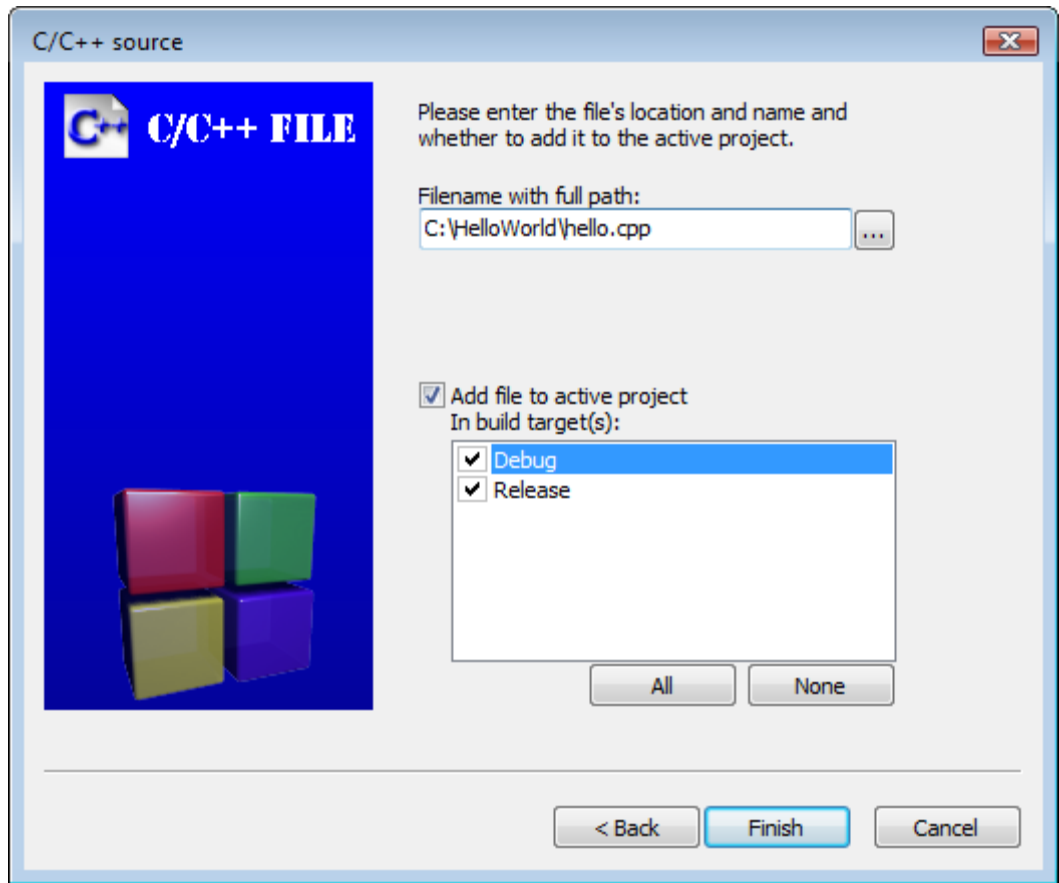
A single source file is of little uses in programs of any useful complexity. In order to handle this, Code::Blocks has several very simple methods of adding additional files to the project.

### **6.2.1. Adding a blank file**

To add the new file to the project, bring up the file template wizard through either `File->New->File...` or `Main Toolbar->New file (button)->File...`



Select C/C++ source and click Go. Continue through the following dialogues very much like the original project creation, selecting C++ when prompted for a language. On the final page, you will be presented with several options. The first box will determine the new file name and location (as noted, the full path is required). You may optionally use the corresponding button to bring up a file browser window to save the file's location. Checking Add file to active project will store the file name in the Sources folder of the Projects tab of the Management panel. Checking any of the build targets will alert Code::Blocks that the file should be compiled and linked into the selected target(s). This can be useful if, for example, the file contains debug specific code, as it will allow the inclusion (or exclusion) from the appropriate build target. In this example, however, the hello function is of key importance, and is required in each target, so select all the boxes and click Finish to generate the file.



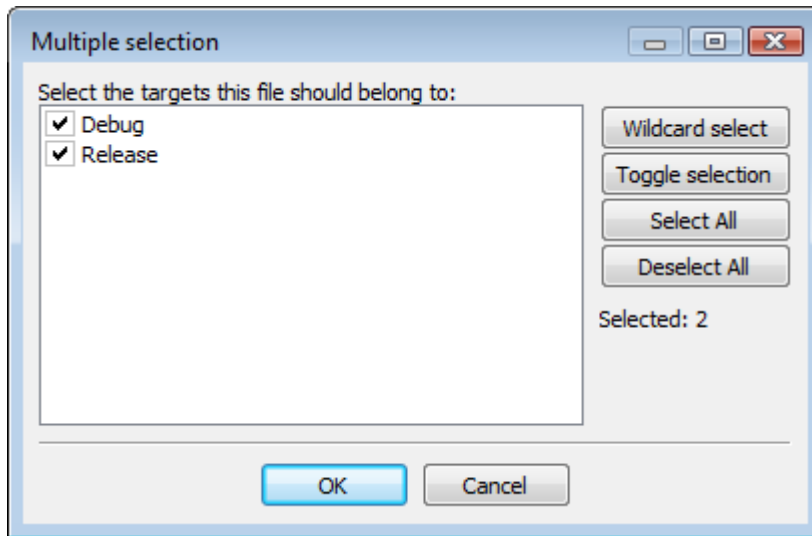




**6.2.2. Adding a pre-existing file**

Click Project->Add files... to open a file browser. Here you may select one or multiple files (using combinations of Ctrl and Shift). (The option Project->Add files recursively... will search through all the subdirectories in the given folder, selecting the relevant files for inclusion.)

Click Open to bring up a dialogue requesting to which build targets the file(s) should belong. For this example, select both targets.



Note: if the current project has only one build target, this dialogue will be skipped.

**6.2.3. Removing a file**

Using the above steps, add a new C++ source file, useless.cpp, to the project. Removing this un-needed file from the project is straightforward. Simply right-click on useless.cpp in the Projects tab of the Management pane and select Remove file from project.

