

Getting Started with the Catalina Geany IDE

Table of Contents

Getting Started with the Catalina Geany IDE.....	1
Introduction.....	1
Using the IDE to build and run a program from scratch.....	2
Using the IDE to build and run an existing simple program.....	12
Using the IDE to build and debug a complex program.....	19
Using the IDE to build and run a program using XMM RAM.....	25
Additions to the Geany IDE for Catalina.....	30
Project Properties.....	30
Catalina Build Commands.....	32

Introduction

Catalina has a new Integrated Development Environment (IDE), supported on both Windows and Linux. It is a customized version of the "Geany" IDE¹.

Geany is a "lightweight" IDE, much simpler than the Code::Blocks IDE. It is easier to use, easier to configure, more user-friendly, and more suitable in general for the types of small C programs typically written for the Propeller.

However, Geany is not as flexible or powerful as Code::Blocks, so Code::Blocks will also remain supported for the foreseeable future, so those who are familiar with it may continue using it.

The Catalina Geany IDE is recommended for those who want to use an IDE but are *not* familiar with Code::Blocks.

This document introduces the new IDE, and gives some examples of how to use it:

- Using the IDE to build and run a program from scratch.
- Using the IDE to build and run an existing simple program.
- Using the IDE to build and debug a complex program.
- Using the IDE to build and run a program using XMM RAM.

Finally, there is a section that discusses the additions made to the standard Geany IDE specifically for Catalina.

This document assumes you are running the Catalina Geany IDE under Windows. There are

¹ *Don't ask me - I guess the spelling "Genie" was already used by another program!*

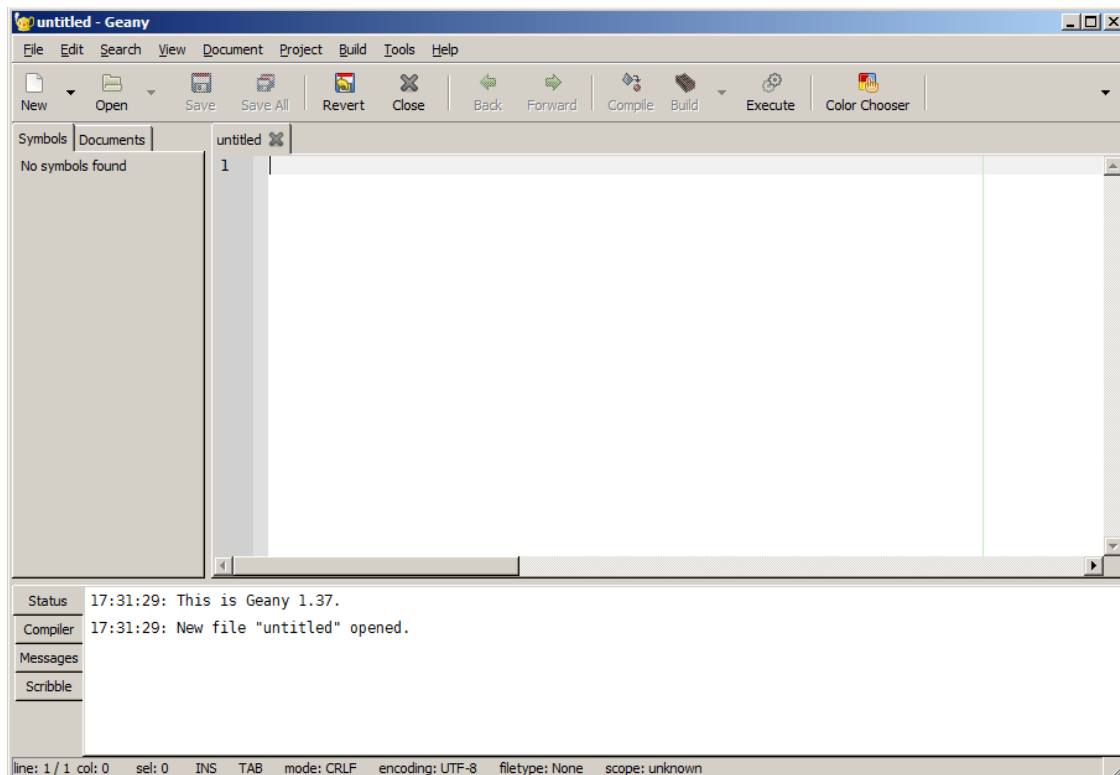
minor cosmetic differences under Linux, but the functionality is identical.

Using the IDE to build and run a program from scratch

On Windows, start the **Catalina Geany IDE** from the Start Menu shortcut of that name.

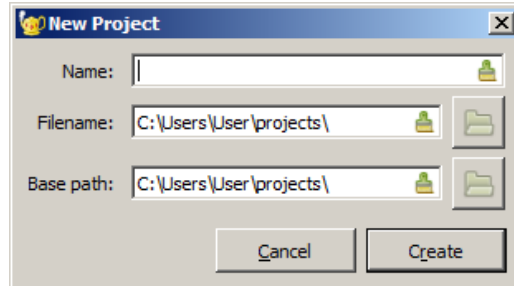
On Linux, open a terminal window, and execute the command **catalina_geany** (this command can be found in the “/opt/catalina/bin” directory²).

You should see a window similar to this:



2 Note that Catalina’s version of Geany expects to be installed in /opt/catalina/catalina_geany, which is where it will end up if you install Catalina to its default location (i.e. /opt/catalina). To install it elsewhere, Geany may need to be recompiled from source.

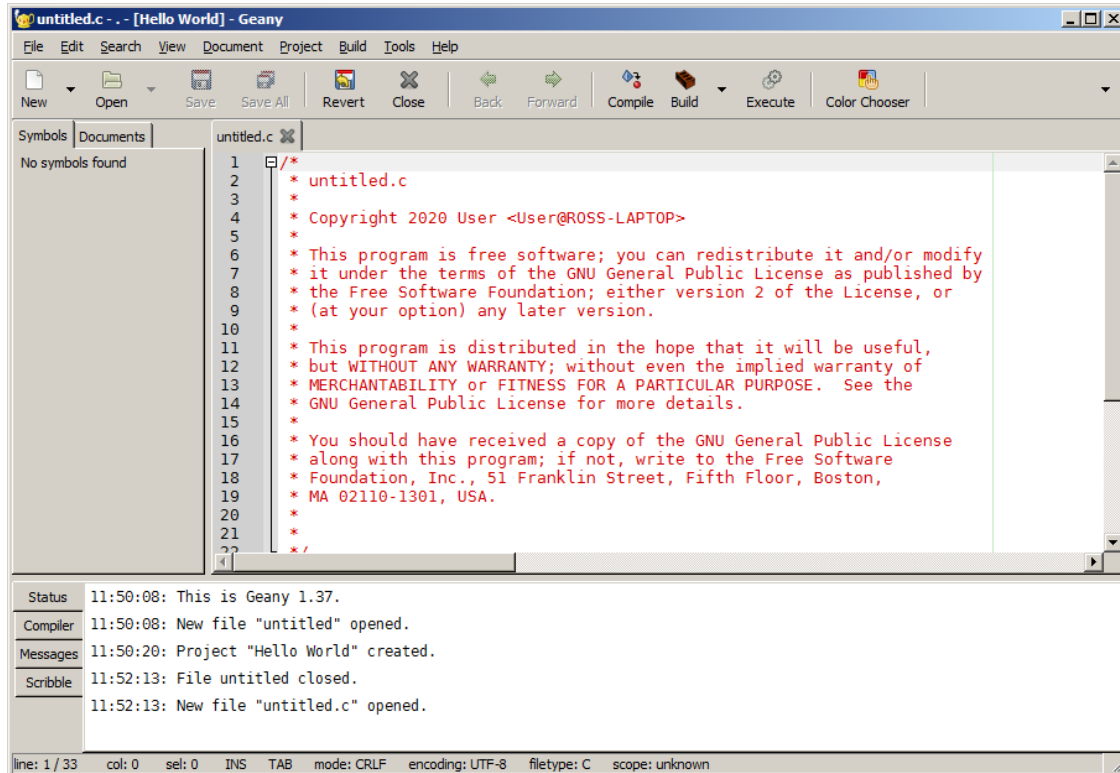
The first thing we typically want to do is create a new project. So select the **Project->New ...** menu entry. You should see a dialog like this:



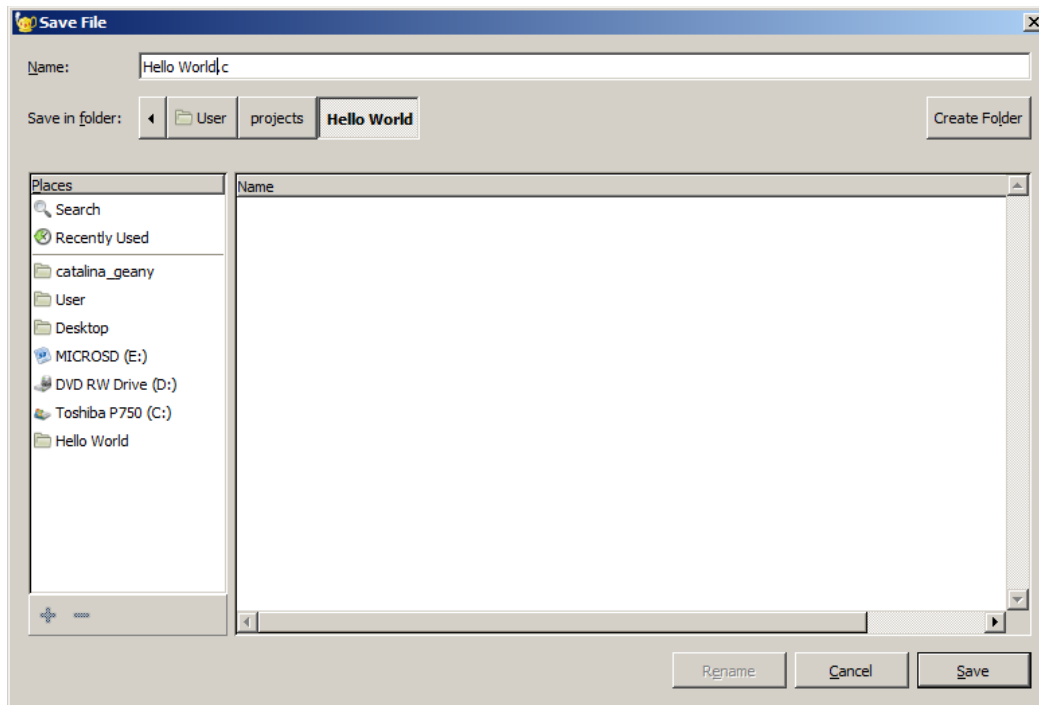
By default in Windows, all projects are created in your Windows User directory, in a subdirectory called "projects" (on Linux, this will be in a subdirectory of your home directory).

All you need to do here is enter the project name - in this case, enter the name **Hello World** and press **Create**. You will be prompted to create the project directory. Just press **OK**.

Next we will create a file, using one of Geany's many built-in file templates. To do this, select **File->New (With Template)->main.c**. You will see the new file in a window called "untitled.c" - something like this:



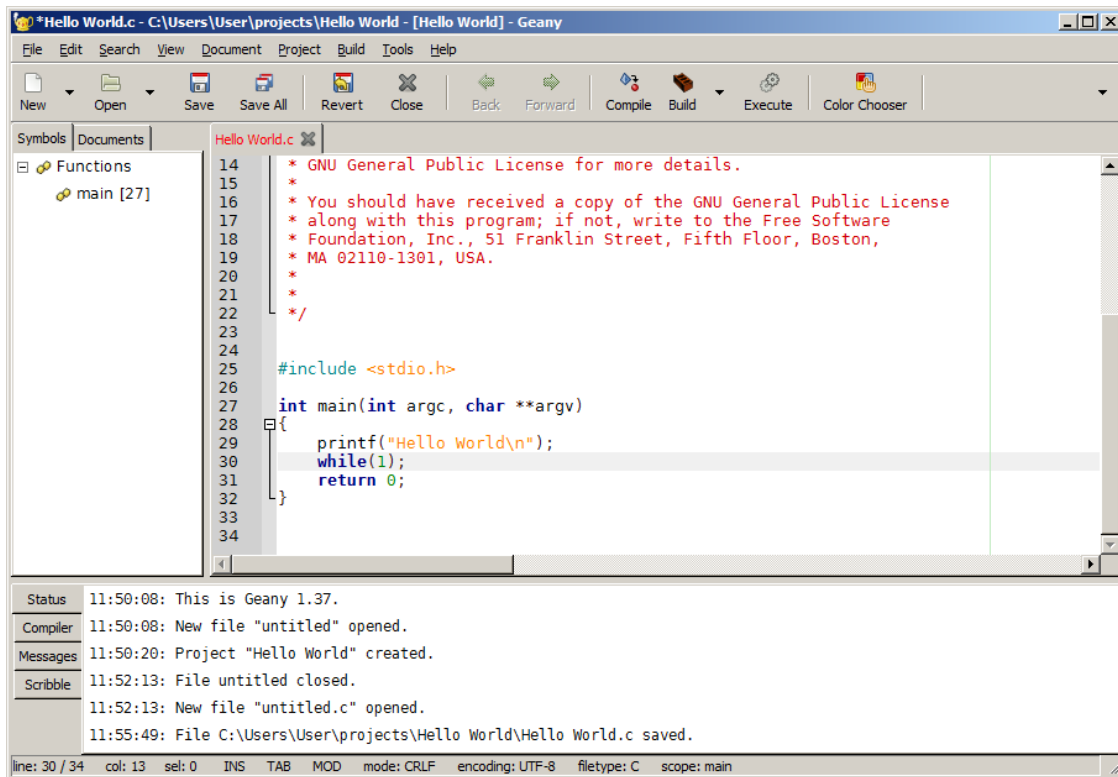
The first thing we want to do is rename and save the file. To do this select **File->Save As ...** - the following dialog box will appear. Enter "Hello World.c" as the file name and press **Save**. This file will be saved in your project's directory:



Next, scroll to the bottom of the file in the right hand pane, and enter the following two lines in the main function, just before the line that says "return 0;". Note that the IDE will assist you as you type.

```
printf("Hello World\n");  
while(1);
```

Your window should now look like this:



The screenshot shows the Geany IDE interface. The title bar reads "Hello World.c - C:\Users\User\projects\Hello World - [Hello World] - Geany". The menu bar includes File, Edit, Search, View, Document, Project, Build, Tools, and Help. The toolbar contains icons for New, Open, Save, Save All, Revert, Close, Back, Forward, Compile, Build, Execute, and Color Chooser. On the left, the Symbols pane shows a tree view with "Functions" and "main [27]". The main editor window displays the following C code:

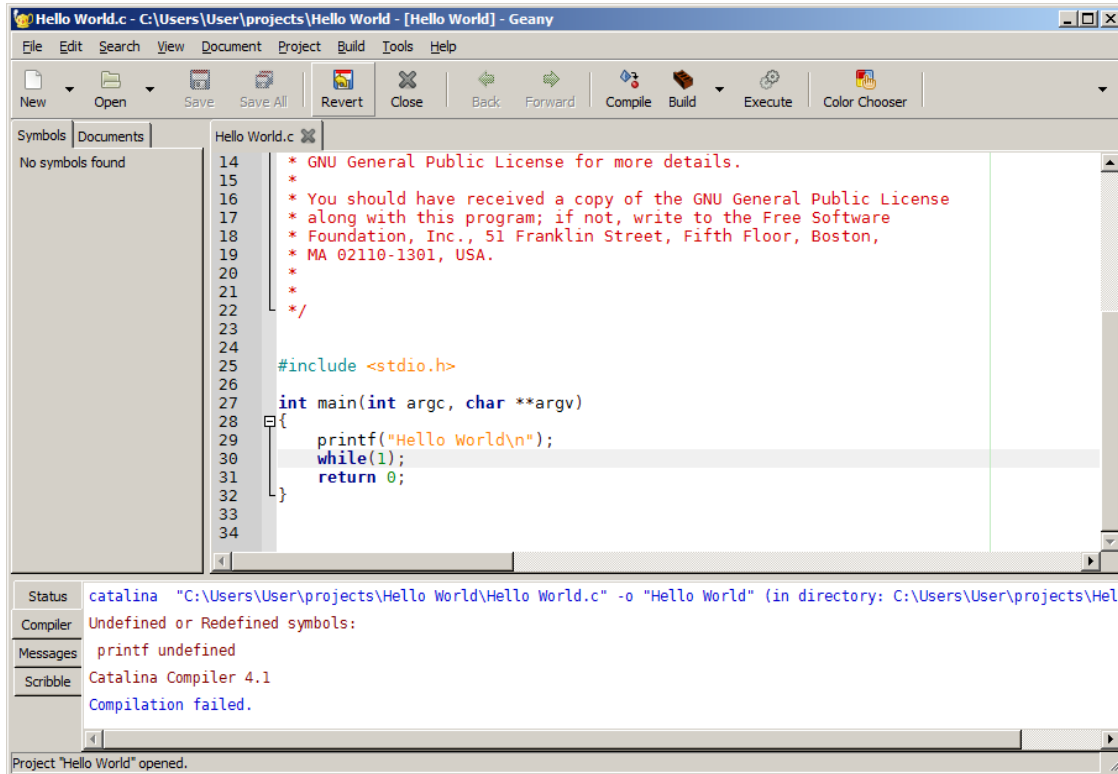
```
14  * GNU General Public License for more details.  
15  *  
16  * You should have received a copy of the GNU General Public License  
17  * along with this program; if not, write to the Free Software  
18  * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,  
19  * MA 02110-1301, USA.  
20  *  
21  *  
22  */  
23  
24  
25  #include <stdio.h>  
26  
27  int main(int argc, char **argv)  
28  {  
29      printf("Hello World\n");  
30      while(1);  
31      return 0;  
32  }  
33  
34
```

At the bottom, the Status pane shows the following messages:

- Status: 11:50:08: This is Geany 1.37.
- Compiler: 11:50:08: New file "untitled" opened.
- Messages: 11:50:20: Project "Hello World" created.
- Scribble: 11:52:13: File untitled closed.
- 11:52:13: New file "untitled.c" opened.
- 11:55:49: File C:\Users\User\projects\Hello World\Hello World.c saved.

The status bar at the bottom indicates: line: 30 / 34 col: 13 sel: 0 INS TAB MOD mode: CRLF encoding: UTF-8 filetype: C scope: main

And now we are ready to try building the project. To do so, select the **Build->Build** menu entry, or press the **Build** button. *Note: we will get undefined symbols from this step - this is expected!* We will explain why, and fix this error in the next step. This is what you should see:

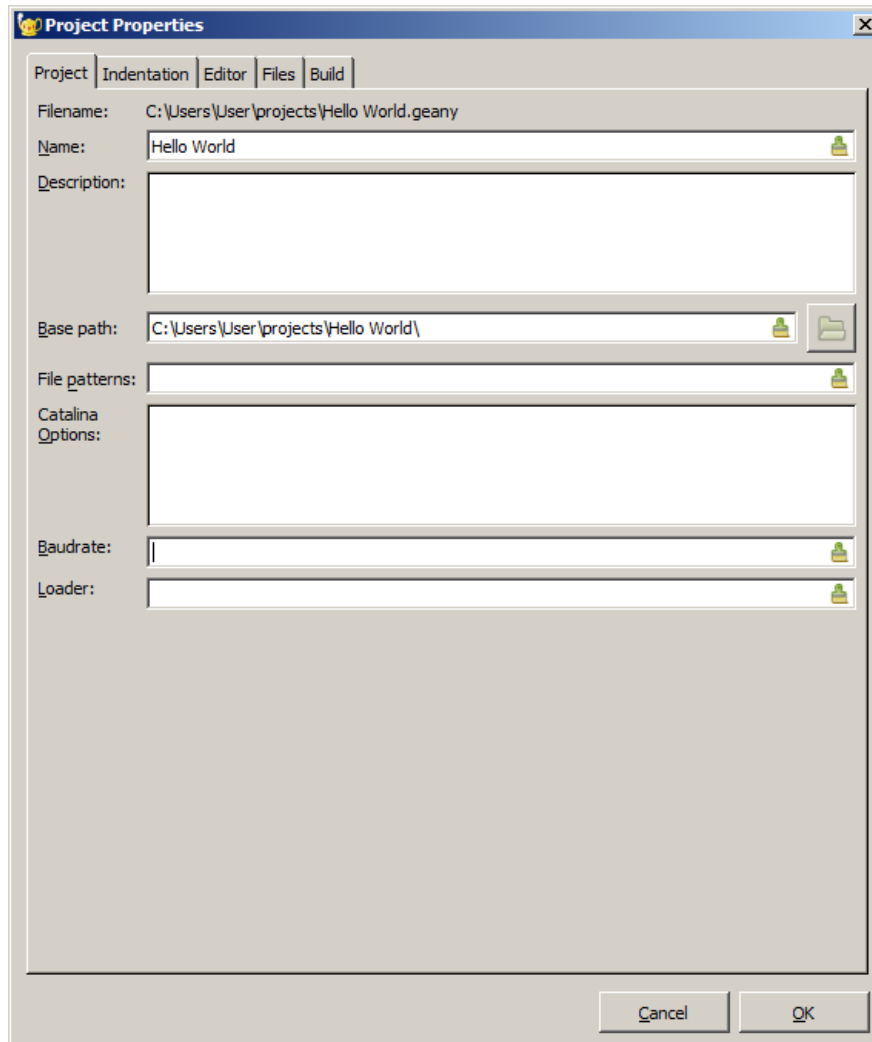


The reason the compilation failed is that "printf" is undefined. This is because we have not told the IDE which version of the C Library to link with. Catalina has several different versions of the C Libraries you can choose from, and it does not have a default setting.

The reason for this is that the standard C libraries are significantly smaller if you exclude specific things you know you won't need in your program. Catalina has four versions of the standard C libraries:

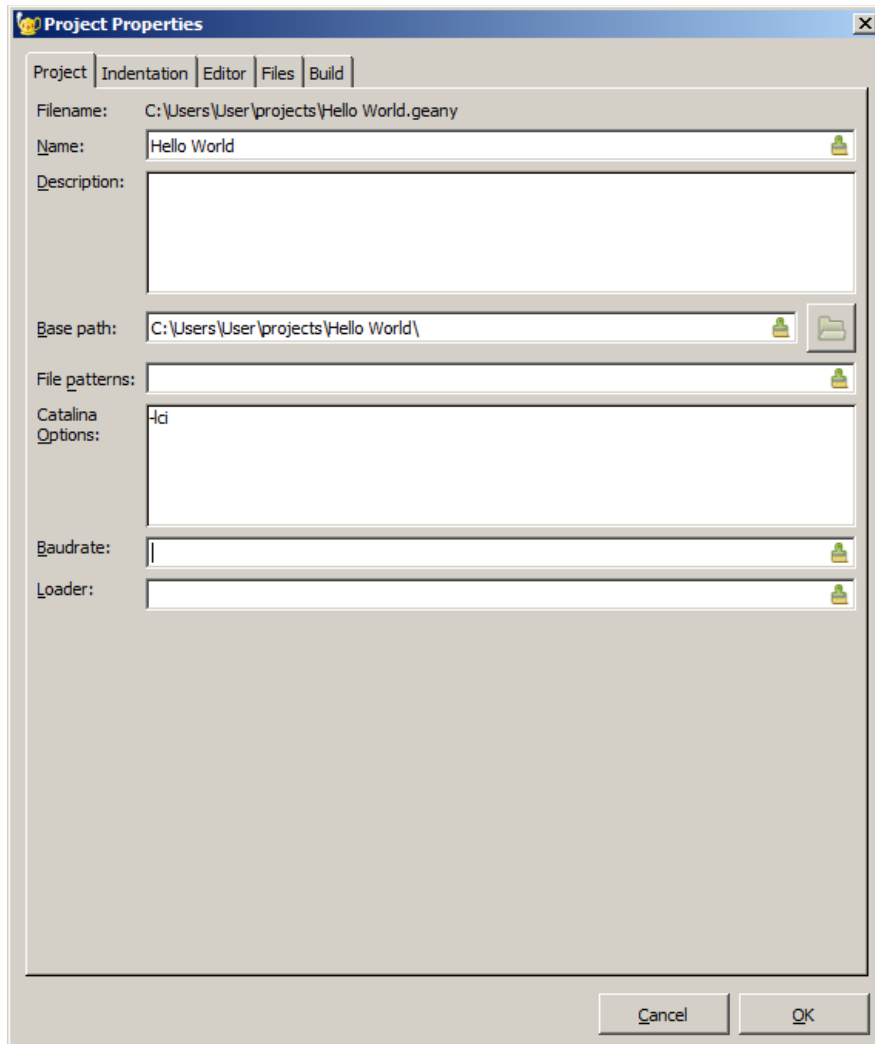
- lci** an *integer* version that does not include floating point I/O in functions such as printf (note that you can still *use* floating point – you just cannot perform I/O on them).
- lc** a version that includes floating point I/O in functions such as printf.
- lcix** an *extended* version that does not support I/O of floating point, but does include file I/O (on platforms that have an SD Card).
- lctx** an *extended* version that includes floating point I/O, plus file I/O (on platforms that have an SD Card).

To set Catalina-specific project options, such as which version of the C library to use, select the **Project->Properties** menu item. You will see a window like this:



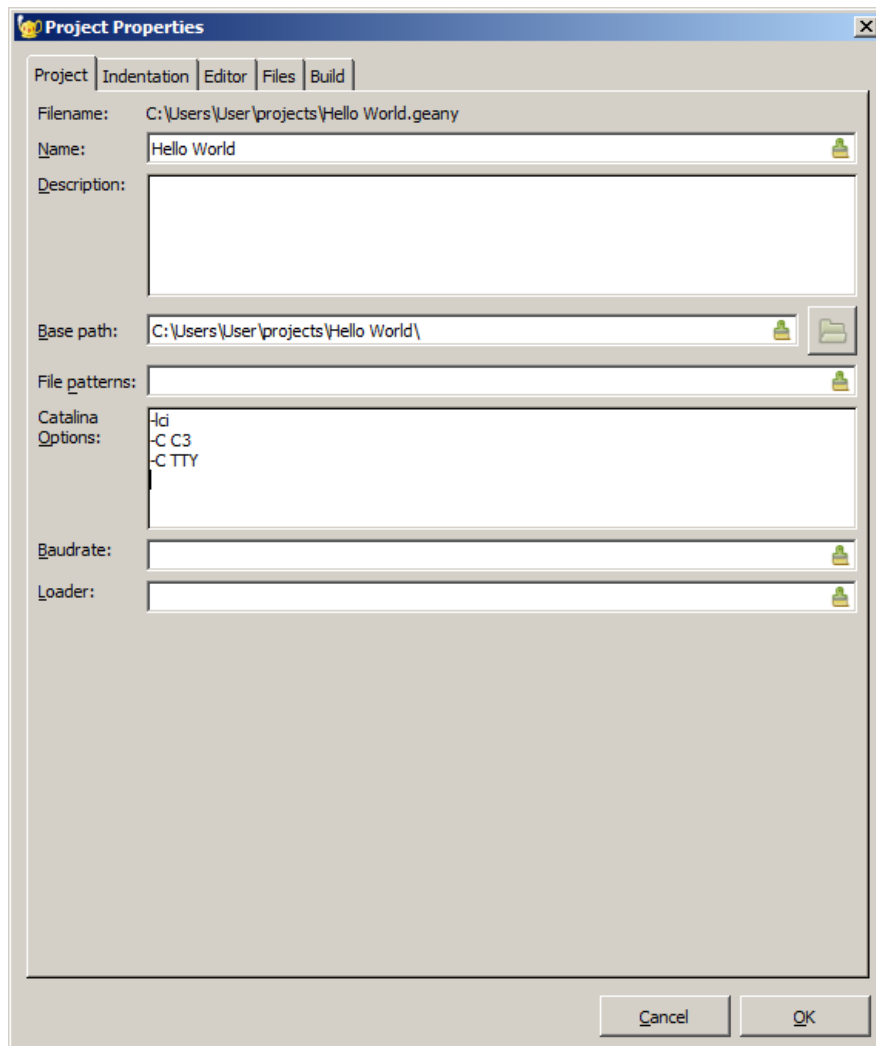
If you are familiar with the Geany IDE, you will notice your first differences from standard Geany in the additions to this dialog box. They are the **Catalina Options**, **Baudrate** and **Loader** fields. For now, all we really need to do is add **-lci** to the **Catalina Options** field - this tells the IDE we want to link our project with the integer version of the standard C library.

The dialog box should then look like this:



However, while we are adding Catalina options, we can take this opportunity to also add any necessary options for our platform, and also tell the project we want to use the TTY plugin for serial I/O (in case this is not the default). This is not required if you are using a standard Propeller platform that is compatible with Catalina's CUSTOM configuration because this is the default, but let's assume you instead have a C3 board. In that case, you would need to also add **-C C3 -C TTY** to the Catalina options.

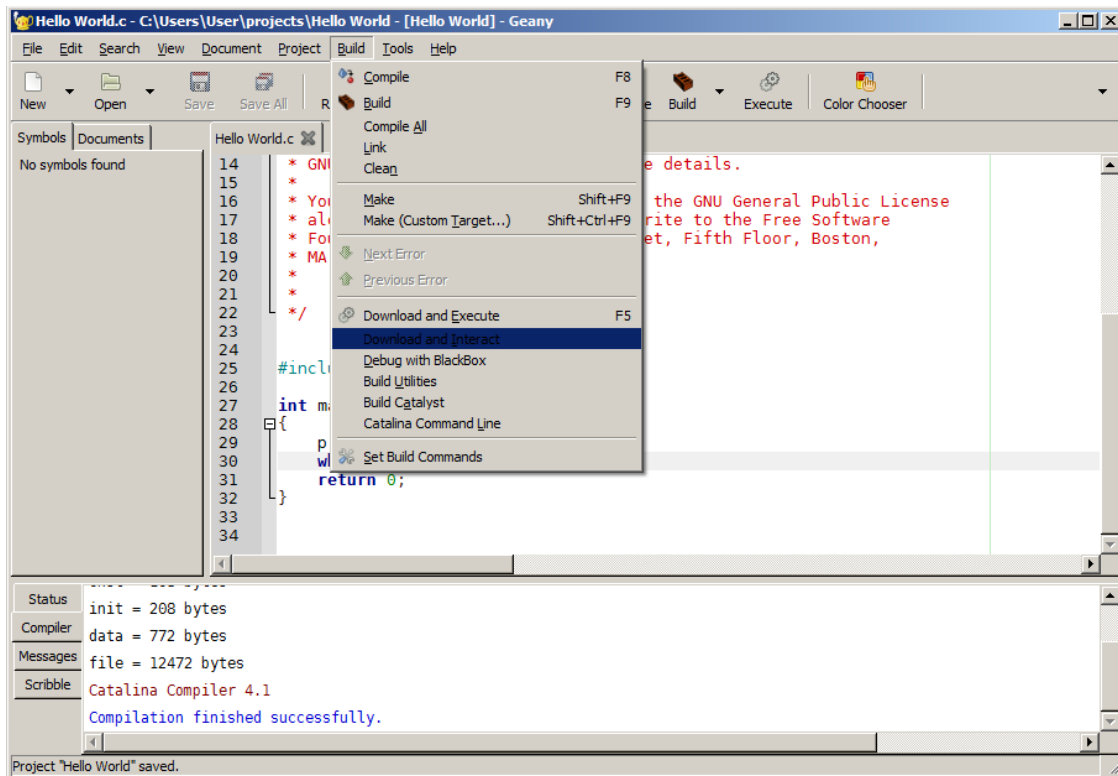
To make the options more readable, you can enter each one on a new line in the **Catalina Options** field. For example:



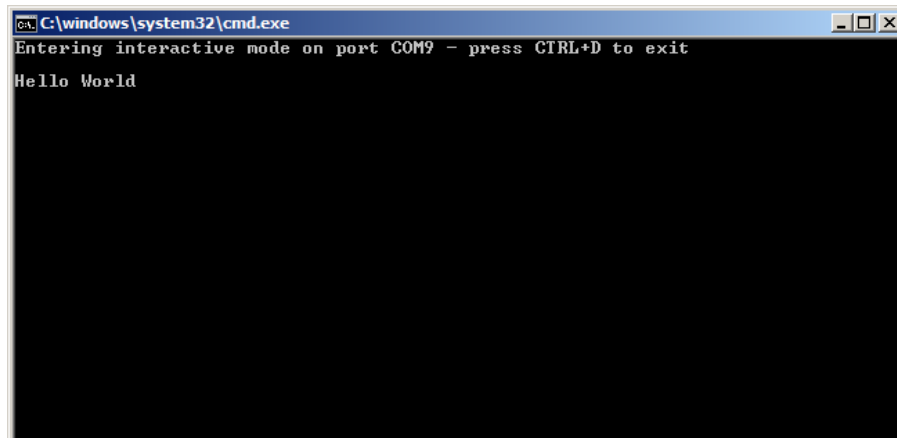
Press **OK** to close the dialog, and then select the **Build->Build** menu item again, or press the **Build** button again.

This time, the compilation should succeed.

Now we can download and execute the program. Because we are using serial I/O as our interface, we want to choose the **Build->Download and Interact** menu item. Otherwise, we would not see any output:



When we do so, we should see an interactive terminal window open up, displaying the output from the program:



For programs that consists of a single C source file, that's all there is to it! You can now close this terminal window.

Just to complete this example, shut down and restart the IDE from the **Catalina Geany IDE** menu item (or the **catalina_geany** command). When you do so, you will notice is that the IDE always restarts where it left off. Normally, this is what you want to do. If it is not, you should close the open project.

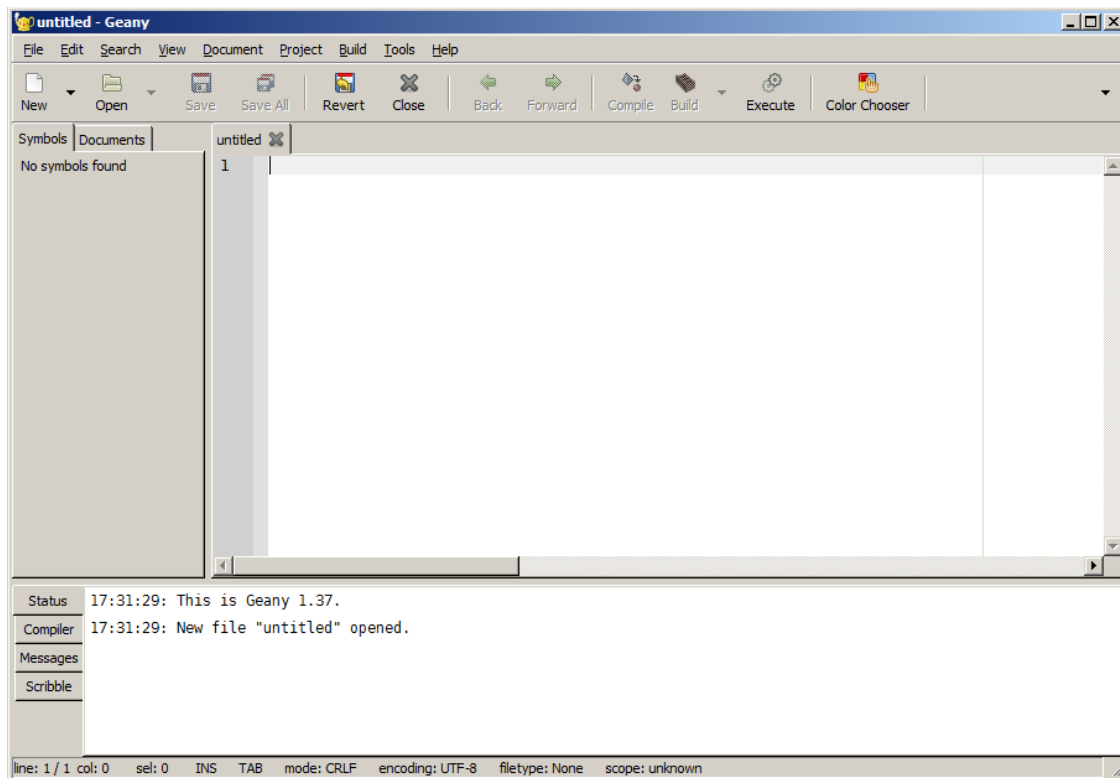
Note that to close the project, you *must* use **Project->Close** menu item. If you just press the **Close** button, it closes the current *file*, but not the *project* - which is not usually what you want to do³

3 *Geany is a little ambiguous about the state of open files - it is not always obvious whether a file has been opened as part of a project or not. You may find you close the project but some files are left open. Or vice-versa. If you accidentally close a file that is part of a project instead of closing the project, don't panic! The file hasn't disappeared - you just need to re-open the file within the project again using the **File->Open** command when the project itself is open.*

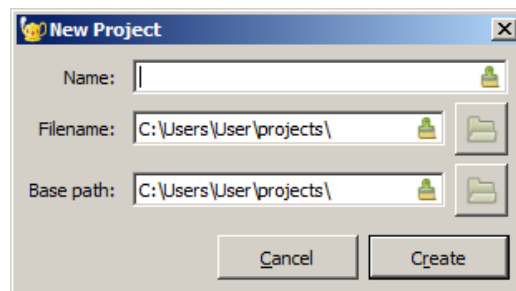
Using the IDE to build and run an existing simple program

If the IDE is not running, start the **Catalina Geany IDE** from the Start Menu shortcut of that name (on Linux, execute the command **catalina-geany**).

If there is a project open, close it by selecting **Project->Close**, and if there are any C files left open, close them by selecting **File->Close**. You should see a window like this (there may or may not be a file called "untitled" still open - that doesn't matter):

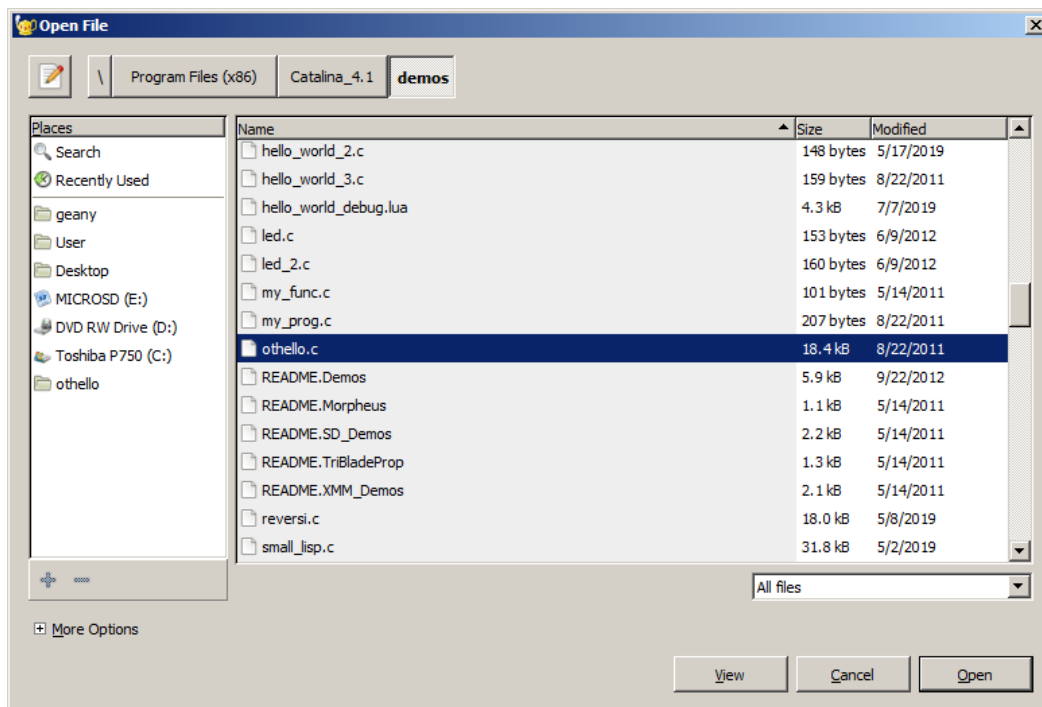


Next, create a new project by selecting the **Project->New ...** menu entry. You should see a dialog like this:

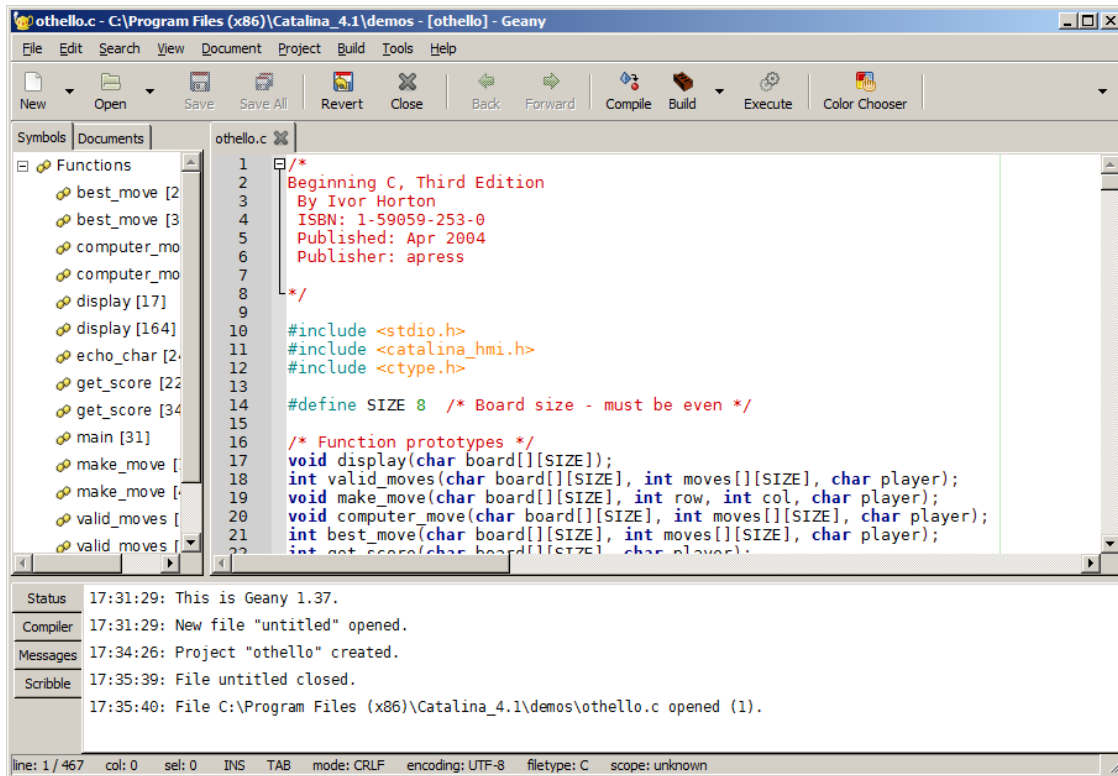


Enter the name **Othello** and press **Create**. You will be prompted to create the project directory. Press **OK**.

The next thing we want to do with our new project is include the "othello.c" program code provided as a Catalina demo program. Select **File->Open ...** or press the **Open** button on the Toolbar. You will be presented with an **Open File** dialog. Navigate to the "demo" directory where Catalina is installed (typically, "C:\Program Files (x86)\Catalina_4.1\demos" on Windows, or "/opt/catalina/demos" on Linux) and select the file "othello.c" from this directory, as shown below. Then press **Open**:



The "othello.c" program will appear in the right hand pane. The symbols defined in the file will appear in the left hand pane:

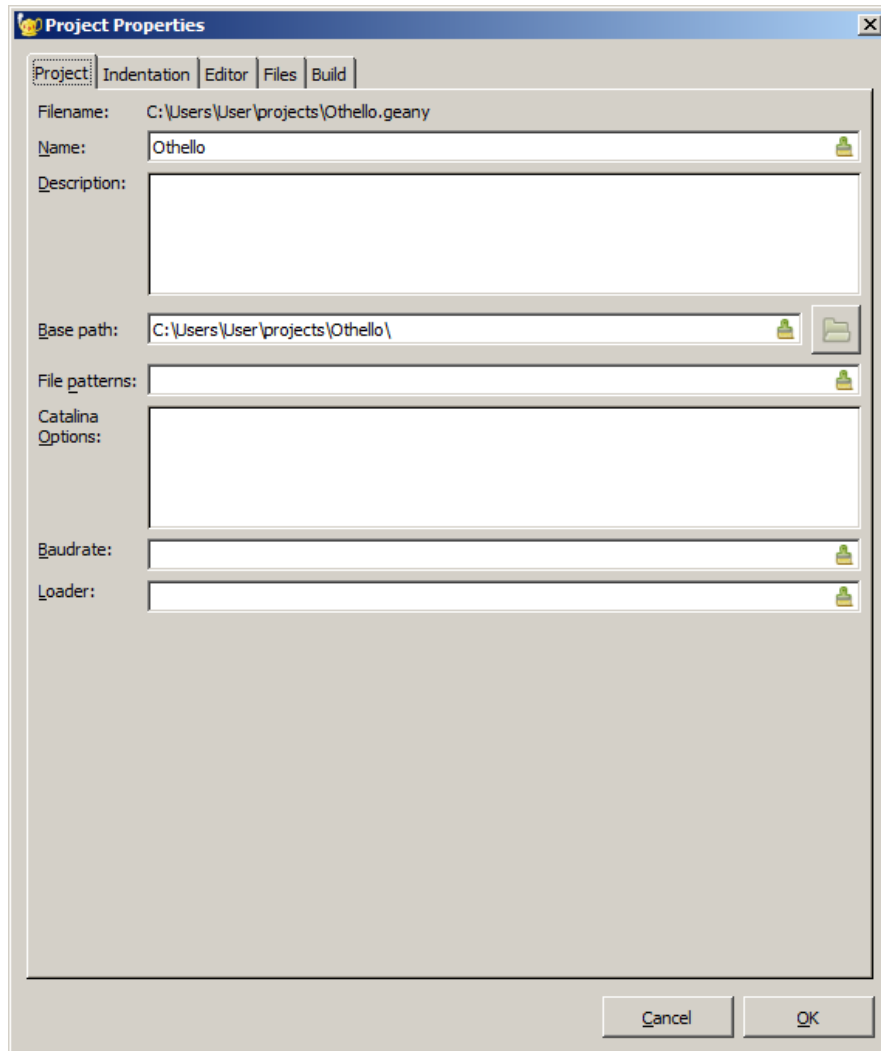


```
1  /*
2  Beginning C, Third Edition
3  By Ivor Horton
4  ISBN: 1-59059-253-0
5  Published: Apr 2004
6  Publisher: apress
7
8  */
9
10 #include <stdio.h>
11 #include <catalina_hmi.h>
12 #include <ctype.h>
13
14 #define SIZE 8 /* Board size - must be even */
15
16 /* Function prototypes */
17 void display(char board[][SIZE]);
18 int valid_moves(char board[][SIZE], int moves[][SIZE], char player);
19 void make_move(char board[][SIZE], int row, int col, char player);
20 void computer_move(char board[][SIZE], int moves[][SIZE], char player);
21 int best_move(char board[][SIZE], int moves[][SIZE], char player);
22 int get_score(char board[][SIZE], char player);
```

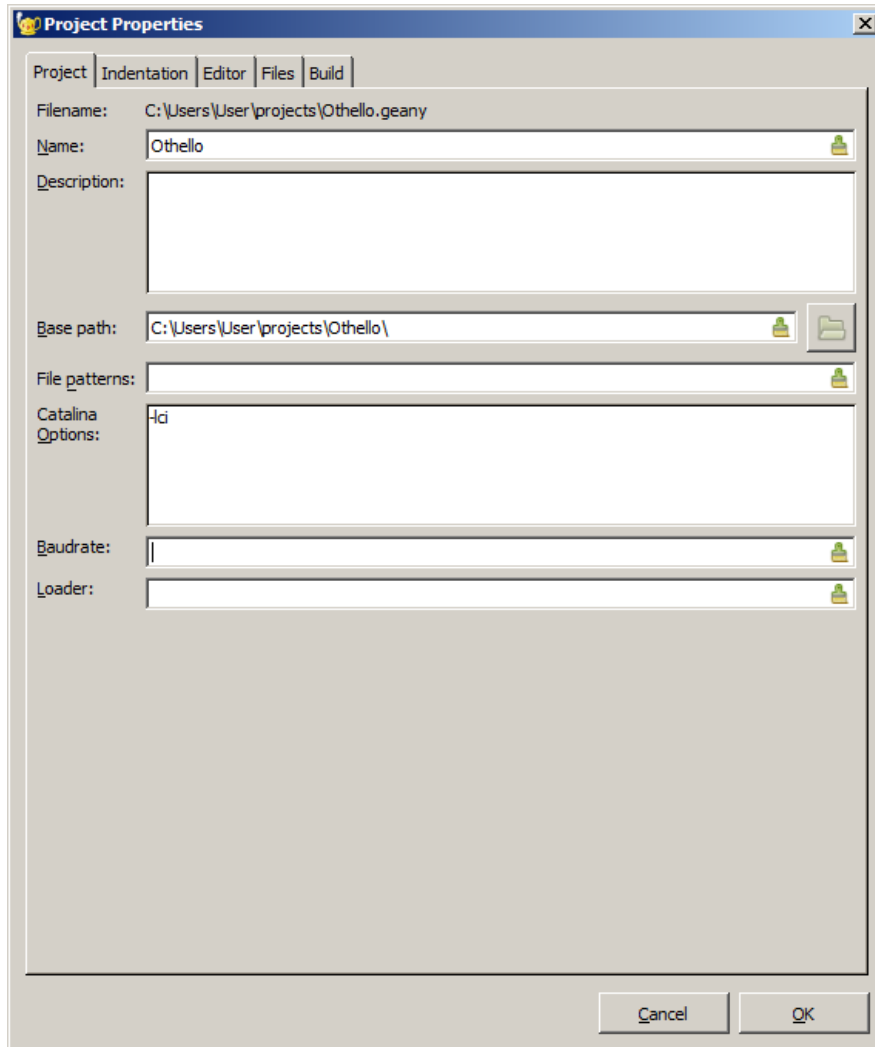
Status 17:31:29: This is Geany 1.37.
Compiler 17:31:29: New file "untitled" opened.
Messages 17:34:26: Project "othello" created.
Scribble 17:35:39: File untitled closed.
17:35:40: File C:\Program Files (x86)\Catalina_4.1\demos\othello.c opened (1).

line: 1 / 467 col: 0 sel: 0 INS TAB mode: CRLF encoding: UTF-8 filetype: C scope: unknown

Before we can build the project, we need to set up some Catalina-specific options, such as which C library to use. This is very easy to do - select the **Project->Properties** menu item to open the properties dialog box:

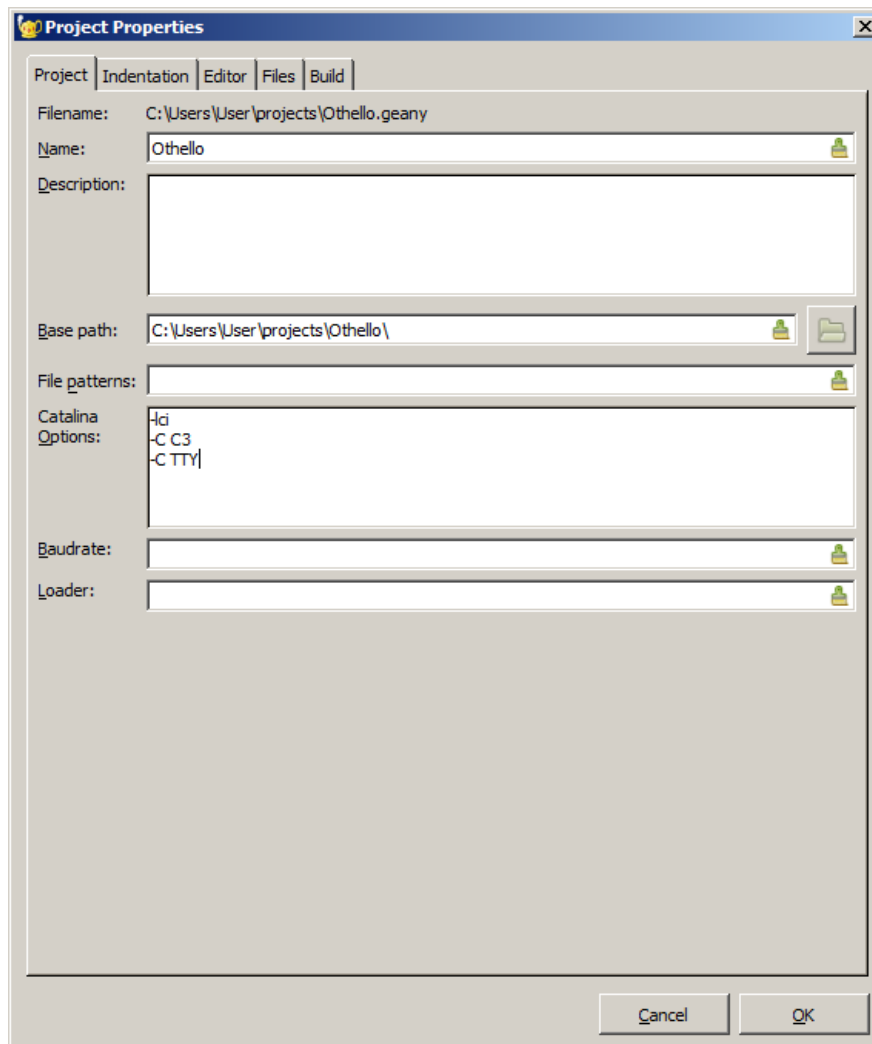


For now, all we really need to do is add **-lci** to the **Catalina Options** field - this tells the IDE we want to link our project with the integer version of the standard C library. The dialog box should then look like this:



However, as in the previous example, while we are adding the library option, we can also add any necessary options for our platform and also tell the project we want to use the TTY plugin for serial I/O. This is not required if you are using a standard Propeller platform that is compatible with Catalina's CUSTOM configuration because it is the default, but let's assume you instead have a C3. In that case, you would need to add **-C C3 -C TTY** to the Catalina options.

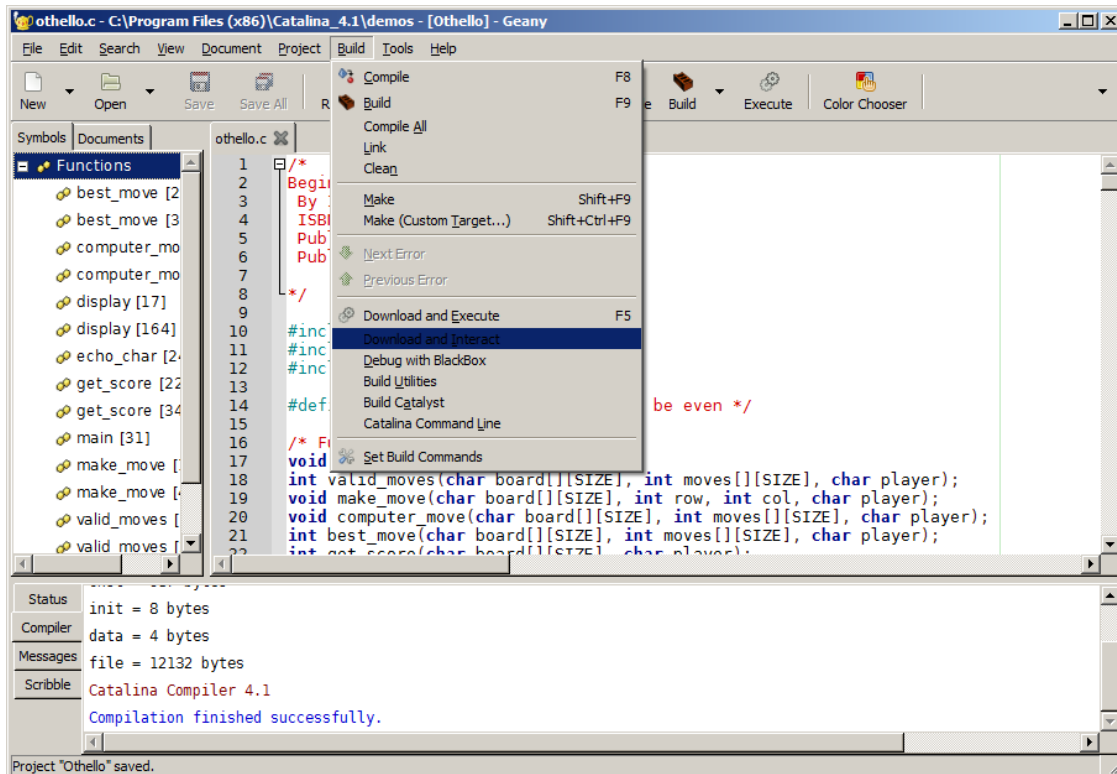
To make the options more readable, you can enter each one on a new line in the **Catalina Options** field:



Press **OK** to close the dialog, and then select the **Build->Build** menu item, or press the **Build** button.

The compilation should succeed without any errors.

Now we can run the program. Because we are using serial I/O as our interface, we want to choose the **Build->Download and Interact** menu item. Otherwise, we would not see any output:



When we do so, we should see an interactive terminal window open up, displaying the output from the program:

```

C:\windows\system32\cmd.exe
Entering interactive mode on port COM9 - press CTRL+D to exit

REUERSI

You can go first on the first game, then we will take turns.
You will be white - (O)
I will be black - (X).
Select a square for your move by typing a digit for the row
and a letter for the column with no spaces between.

Good luck! Press Enter to start.
-

```

That's it! A similar procedure can be used to build most of the programs in the "demo" folder.

In the next example, we will demonstrate compiling a program that consists of multiple files, and also using the debugger from within the IDE.

Using the IDE to build and debug a complex program

If the IDE is not running, start the **Catalina Geany IDE** from the Start Menu shortcut of that name (on Linux, execute the command **catalina-geany**).

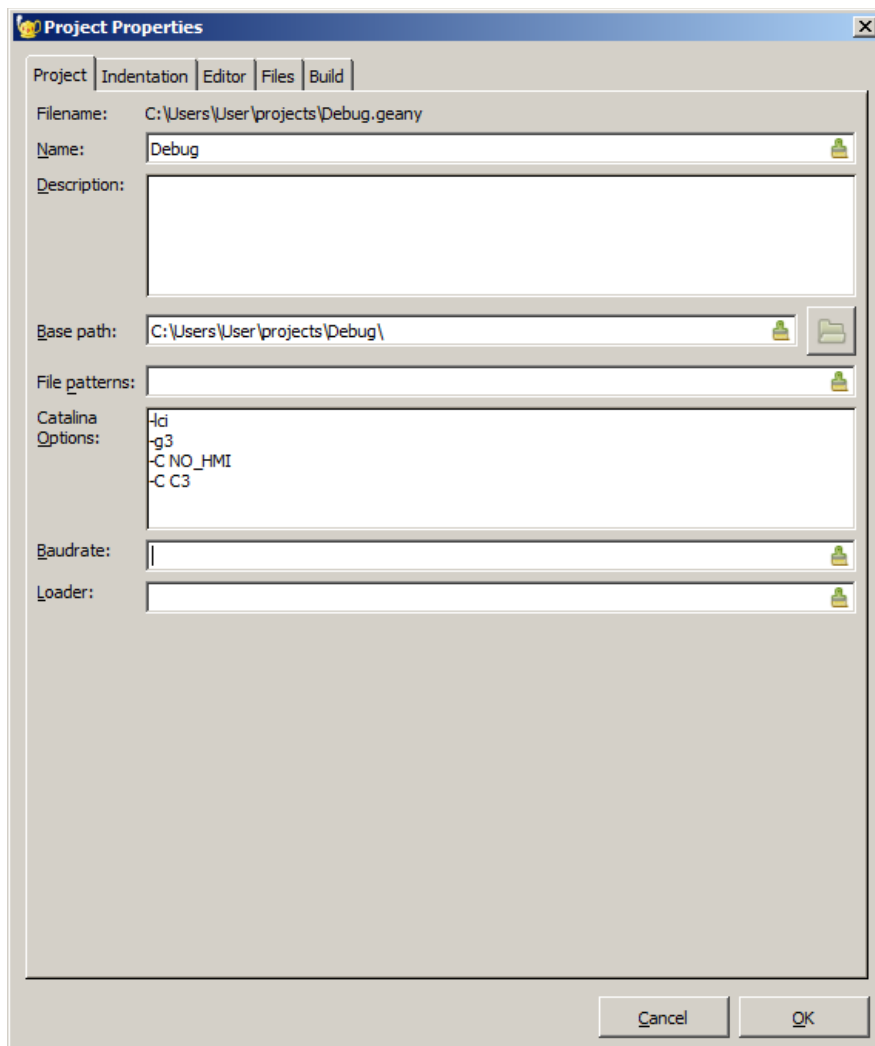
If there is a project open, close it by selecting **Project->Close**, and if there are any C files left open, close them by selecting **File->Close**. You should see a window like this (there may or may not be a file called "untitled" still open - that doesn't matter):

Now we can create a new project using the **Project->New** menu item. This time, we will name our project **Debug**.

The first thing we want to do with this project is tell the IDE we want a debug compilation. To do that, select the **Project->Properties** menu item and enter the Catalina Options **-lci -g3 -C NO_HMI**, plus any other platform options (such as **-C C3**) in the **Catalina Options** field.

The **-lci** identifies the C library we want to use, **-g3** enables debugging, and **-C NO_HMI** disables the HMI, which might otherwise use the serial port (which in this case we want to use for debugging). We don't actually care about the output, as this is simply a demo of the compilation and debugging process.

Your project properties should now look something like this:



In the last example, we compiled a program that existed in the Catalina demo folder, without copying it into our project. In this example, we are actually going to copy the relevant *.c and *.h files to our project folder. This is what you would normally want to do to modify the files, and in this case it is required to allow us to use the **Compile All** and **Link** commands (which we must use instead of **Build** to compile this project).

The easiest way to copy the files is in a **Catalina Command Line** window. Open a Catalina Command Line using the Start Menu shortcut of that name, then execute the following commands:

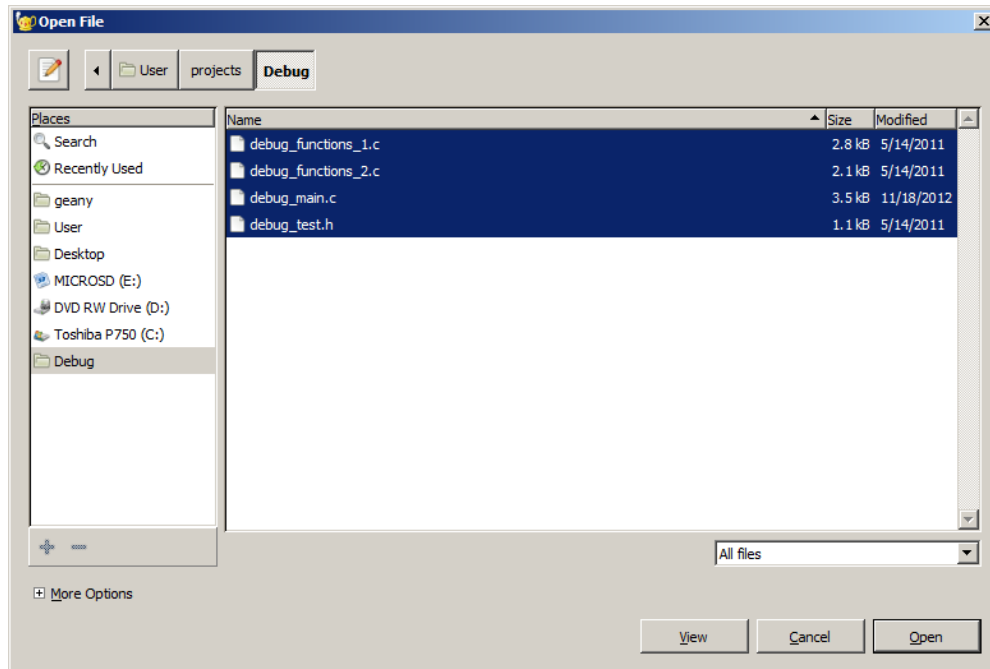
On Windows:

```
copy demos\debug\*.c %HOMEPATH%\projects\Debug
copy demos\debug\*.h %HOMEPATH%\projects\Debug
```

On Lunix:

```
cp /opt/catalina/demos/debug/*.c $HOME/projects/Debug
cp /opt/catalina/demos/debug/*.h $HOME/projects/Debug
```

Now in the IDE you can select **File->Open**, or press the **Open** button to open the files we just copied to the project folder. Note that you can select multiple files to open at once:



Unlike Code::Blocks, the Catalina Geany IDE has no knowledge about project dependencies, and no built-in knowledge of how to make complex programs that consist of multiple source files. However, we can simply compile all the C files that are in the project folder by selecting the **Build->Compile All** menu item. In this case, this is exactly what we want to do. However, to use the **Compile All** command, the program files to be compiled must exist in the project folder - they cannot just be references to files stored elsewhere, as we did in the previous example. This is why we had to copy them into the project folder.

If simply compiling *all* the C source files is *not* all you need to do to build your program, or they

have to be compiled or otherwise processed in a particular order, then you will have to use the **Compile** command on each of the files individually.

When we compile the source files, we may get some warning messages, but the compilation should complete successfully:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include "debug_test.h"
5
6  typedef unsigned long long my_int;
7
8  typedef enum aa { cat, dog };
9
10
11 typedef struct bb {
12     int a;
13     char * b;
14     float c[10];
15 };
16
17 typedef struct cc {
18     int **a;
19     float *b[10];
20     char **c;
21 };
22
23 typedef union dd {

```

Status: debug_functions_2.c:
Compiler: debug_main.c:
Messages: Catalina Compiler 4.1
Scribble: option -g will NOT override current target (blackcat)
Compilation finished successfully.

line: 170 / 186 col: 10 sel: 0 INS TAB mode: CRLF encoding: UTF-8 filetype: C scope: main

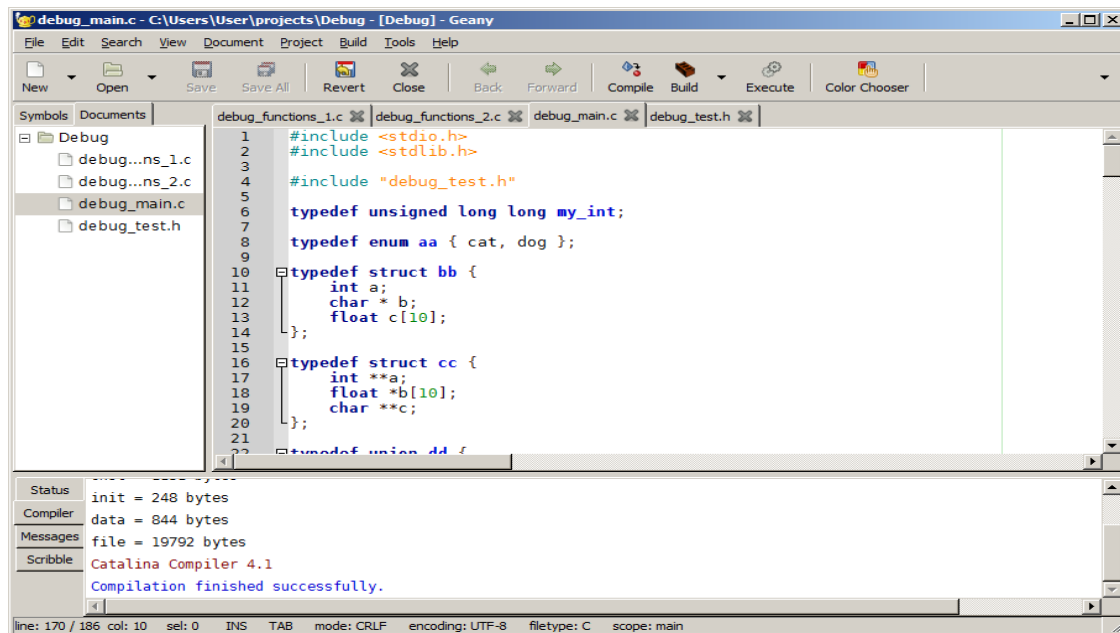
Now, we have compiled all the C files in the project (into ".obj" files under Windows, or ".o" files under Linux), but we have not yet linked them together in an executable binary.

Before we do so, it is important to select the "main" C file, because the IDE uses the name of that file for the output binary. So, in this case, select the file "debug_main.c" in the left hand pane, or using the tabs at the top of the right hand pane, and then select **Build->Link**. The project should link all the compiled objects into an executable binary (in this case, it will be called "debug_main.binary").

The use of the **Compile** (or **Compile All**) command and then the **Link** command is an alternative to the single-step **Build** command, and is generally required for complex programs that consist of multiple source files.

When you use the **Compile** or **Compile All** commands, you may sometimes get warning messages – this is because some of the Catalina options (such as libraries) are not actually required during the compilation phase, only during the linking phase - and so you may see messages saying that some options are being ignored. This is normal.

After executing the **Link**, this is what you should see:



The screenshot shows the Geany IDE interface. The main editor window displays a C program with the following code:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include "debug_test.h"
5
6 typedef unsigned long long my_int;
7
8 typedef enum aa { cat, dog };
9
10 typedef struct bb {
11     int a;
12     char * b;
13     float c[10];
14 };
15
16 typedef struct cc {
17     int **a;
18     float *b[10];
19     char **c;
20 };
21
22 typedef union dd {
```

The left-hand pane shows a project tree with the following files:

- Debug
- debug...ns_1.c
- debug...ns_2.c
- debug_main.c
- debug_test.h

The bottom status bar shows the following information:

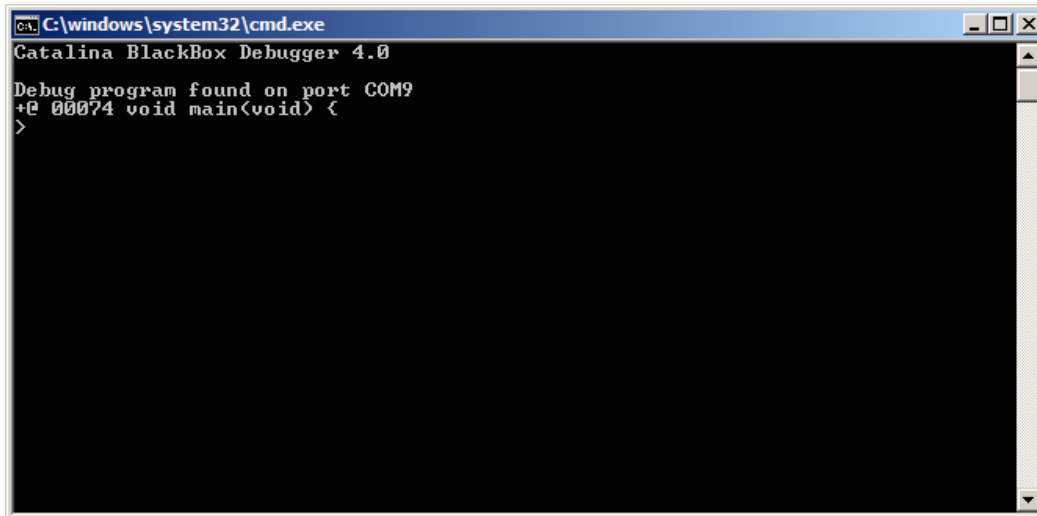
Status	init = 248 bytes
Compiler	data = 844 bytes
Messages	file = 19792 bytes
Scribble	Catalina Compiler 4.1

The bottom status bar also displays: `line: 170 / 186 col: 10 sel: 0 INS TAB mode: CRLF encoding: UTF-8 filetype: C scope: main`

The next step is to download the program for debugging. Since we don't want a terminal to see any I/O, we do that by selecting the **Build->Download and Execute** menu item, or just pressing the **Execute** button. Again, it is important to ensure the main program file is selected in the left hand pane, as this will be used to determine the name of the binary file to be downloaded.

When the download is complete, close the window (you should see the message (**program exited with code: 0**) - this message refers to the *download*, not the program itself).

Finally, select **Build->Debug with BlackBox**. You should see a window like this:



```
C:\windows\system32\cmd.exe
Catalina BlackBox Debugger 4.0
Debug program found on port COM9
+@ 00074 void main(void) {
>
```

We can now interact with the BlackBox debugger. For instance, enter **n <CR>** to step through the program line by line. To step into a function on the line about to be executed, instead enter the command **s i <CR>**. Enter **h <CR>** for help, or **q <CR>** to quit.

There is a separate document ("Getting Started with BlackBox") that gives more information about using the BlackBox debugger, and it uses this demo program as an example.

Using the IDE to build and run a program using XMM RAM

NOTE 1: To execute this example, you must have a Propeller 1 platform supported by Catalina that has XMM RAM. This example assumes you are using a Credit Card Computer (C3).

NOTE 2: This example assumes you have worked through the previous examples, so it does not give screenshots of all the screens you will see along the way.

If the IDE is not running, start the **Catalina Geany IDE** from the Start Menu shortcut of that name (on Linux, execute the command **catalina-geany**).

If there is a project open, close it by selecting **Project->Close**, and if there are any C files left open, close them by selecting **File->Close**.

Create a new project by selecting **Project->New** and enter the name **Startrek**.

In this example, we can compile the file from its original location, so select **File->Open**, or press the **Open** button. Navigate to the "demo" directory where Catalina is installed (typically, "C:\Program Files (x86)\Catalina_4.1\demos" on Windows, or "/opt/catalina/demos" on Linux) and select the file "startrek.c" from this directory, as shown below. Then press **Open**. You should see this:

```

startrek.c - C:\Program Files (x86)\Catalina_4.1\demos - [Startrek] - Geany
File Edit Search View Document Project Build Tools Help
New Open Save Save All Revert Close Back Forward Compile Build Execute Color Chooser
Symbols Documents startrek.c x
Functions
  cint [134]
  cint [2120]
  complete_ma
  complete_ma
  compute_vect
  compute_vect
  course_contr
  course_contr
  damage_cont
  damage_cont
  dirdist_calc [1
  dirdist_calc [1
  end_of_game
  end_of_game
1  /*
2  * startrek.c
3  *
4  * Super Star Trek Classic (v1.1)
5  * Retro Star Trek Game
6  * C Port Copyright (C) 1996 <Chris Nystrom>
7  *
8  * This program is free software; you can redistribute it and/or modify
9  * in any way that you wish. _Star Trek_ is a trademark of Paramount
10 * I think.
11 *
12 * This program is distributed in the hope that it will be useful,
13 * but WITHOUT ANY WARRANTY; without even the implied warranty of
14 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
15 *
16 * This is a C port of an old BASIC program: the classic Super Star Trek
17 * game. It comes from the book _BASIC Computer Games_ edited by David Ahl
18 * of Creative Computing fame. It was published in 1978 by Workman Publishing,
19 * 1 West 39 Street, New York, New York, and the ISBN is: 0-89489-052-3.
20 *
21 * See http://www.cactus.org/~nystrom/startrek.html for more info.
22 *
Status 13:21:42: File C:\Program Files (x86)\Catalina_4.1\demos\othello.c opened (1).
Compiler 13:21:44: File C:\Program Files (x86)\Catalina_4.1\demos\othello.c closed.
Messages 13:21:52: Project "Startrek" created.
Scribble 13:21:53: New file "untitled" opened.
13:22:20: File untitled closed.
13:22:21: File C:\Program Files (x86)\Catalina_4.1\demos\startrek.c opened (1).
line: 1 / 2211 col: 0 sel: 0 INS TAB mode: CRLF encoding: UTF-8 filetype: C scope: unknown

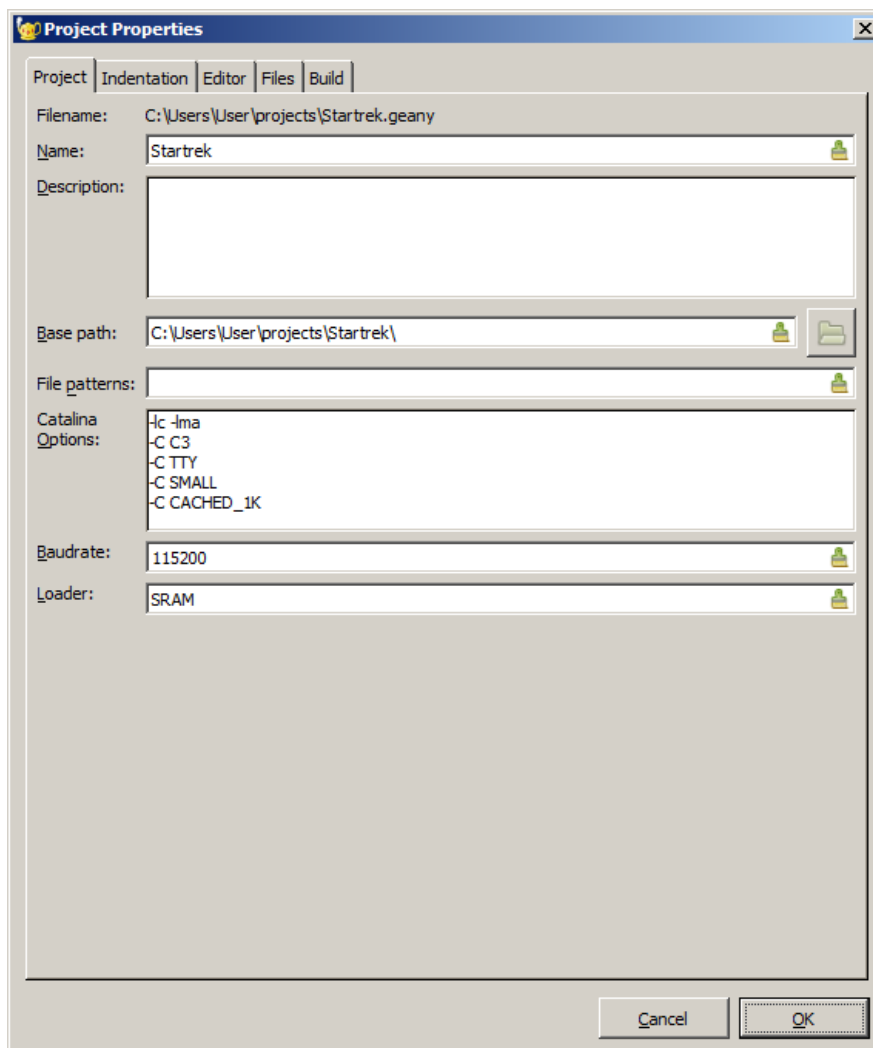
```

Now, we must set the project's Catalina Options. Select **Project->Properties**.

For this program, we must use the standard C library (**-lc**) and also a maths library (**-lma**). We must also specify the platform to use (**-C C3**) and that we want to use serial I/O (**-C TTY**). Finally, we must also specify the memory model (**-C SMALL**) and the caching option to be used (**-C CACHED_1K**).

For the first time, we also need to specify something in the **Loader** field. We will specify **SRAM** here, because we are compiling the program to execute from SRAM (this platform also has FLASH that we could use).

So your project properties will end up looking something like this:



Now we are ready to build the project. Select **Build->Build** or press the **Build** button.

The program should compile successfully.

However, before we can execute the program, we must build the XMM utilities (if we have not already done so). We can do that from within the IDE. Select **Build->Build Utilities**. A terminal window should appear:

```
C:\windows\system32\cmd.exe
=====  
Building Catalina Utilities  
=====  
This command will build various Catalina utilities. It is important that  
the utilities be built with options compatible with those you use when  
you build your Catalina programs.  
For example, if you intend to compile programs to execute from FLASH RAM,  
then to load the program you must build the FLASH load utility.  
Similarly, if you if you intend to compile programs to use a specific  
cache size then you must build the load utilities with the same cache  
size.  
You can re-run this program any time if you need to change options.  
Press ENTER to begin ...
```

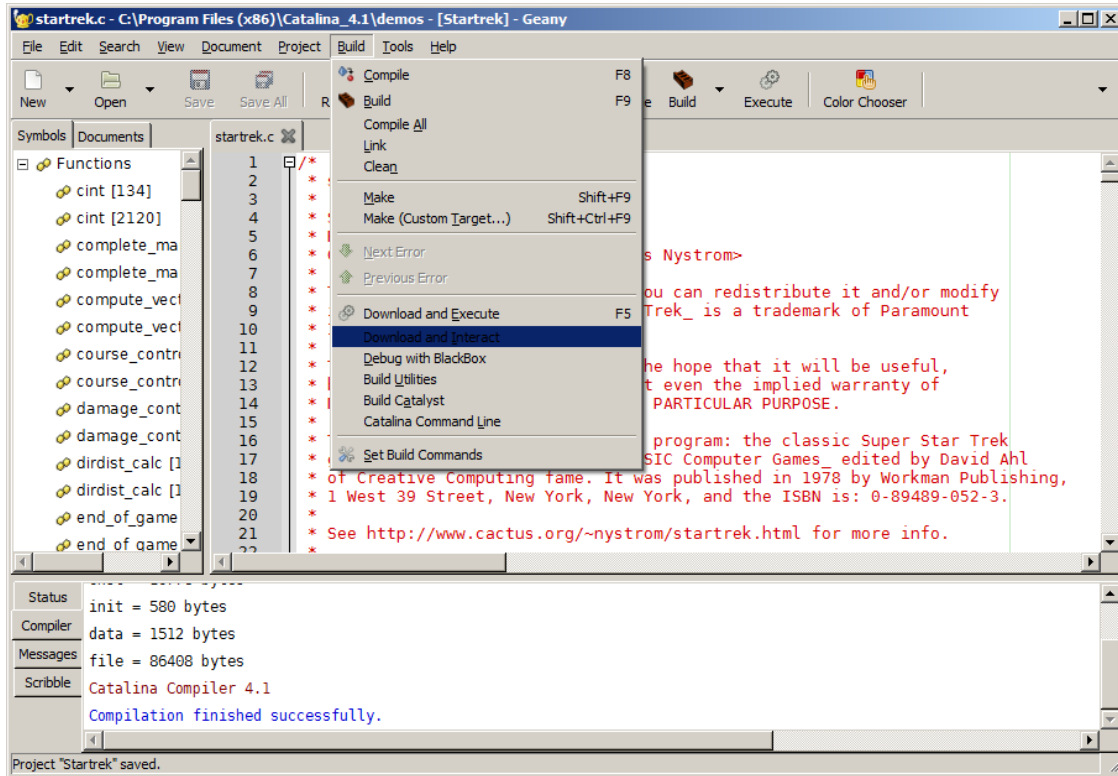
You must step through the build_utilities program, specifying the following options:

- Your platform (**C3**)
- Your XMM board (in the case of the C3 the XMM RAM is built-in, so you do not need to specify anything here)
- Your FLASH cache size (**1**)
- Your FLASH Boot options (nothing is required here)
- Your SRAM cache size (**1**)
- Whether you want to use FLASH or SRAM (**S**).

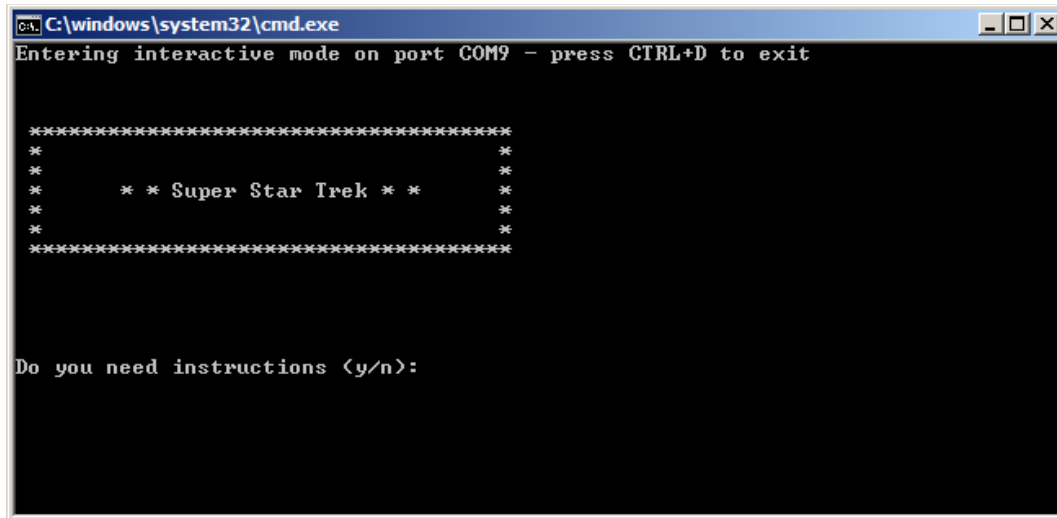
The build_utilities window will close automatically when it is complete.

It is important that the options entered in build_utilities match the options in the Project Properties – in particular, the platform and the cache size.

Now we are ready to download the program – since we want to see the serial I/O, we do so by selecting **Build->Download and Interact**:



You should see the program download using the first and second loaders (the SRAM loader will be used automatically here), and then an interactive terminal window will appear:



```
C:\windows\system32\cmd.exe
Entering interactive mode on port COM9 - press CTRL+D to exit

*****
*                                           *
*           * * Super Star Trek * *       *
*                                           *
*****

Do you need instructions <y/n>:
```

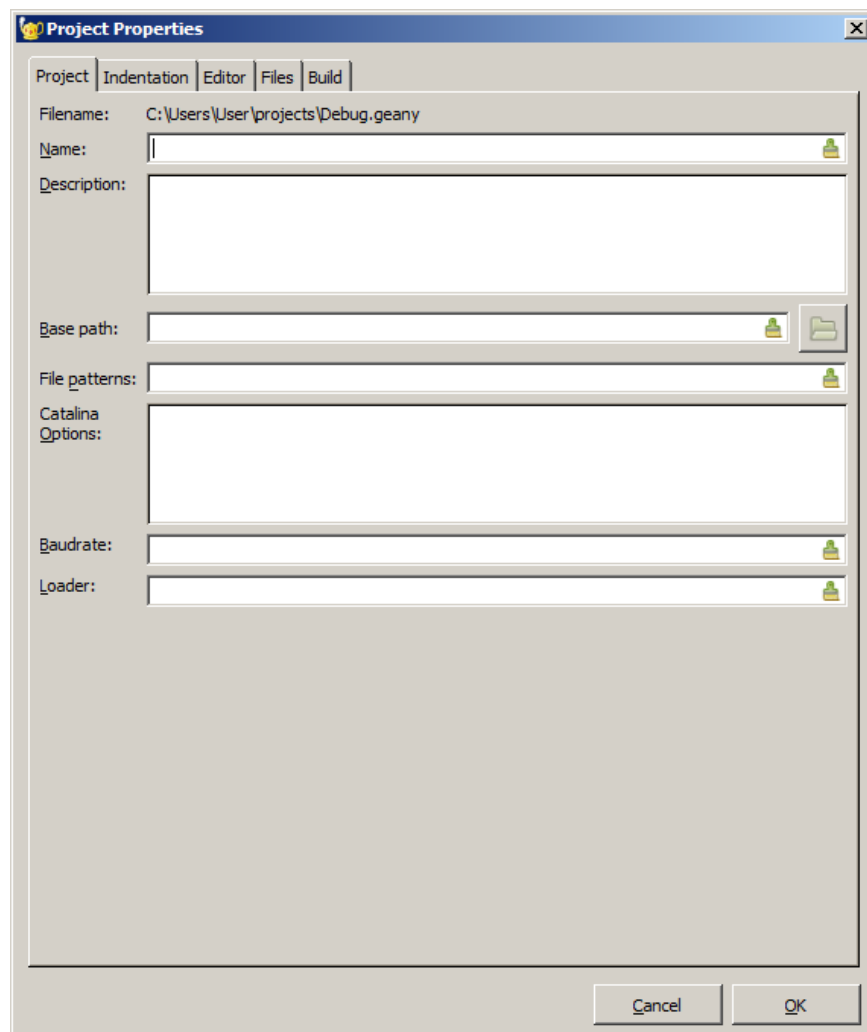
We're done! You can build other Catalina demo programs that require XMM RAM using similar steps.

Additions to the Geany IDE for Catalina

Geany is a very flexible IDE. However, some minor changes have been made to better support the Catalina C compiler. These are in the form of Catalina-specific additions to the standard Geany project handling.

Project Properties

We have already seen the main changes to Geany for Catalina - they are the new fields on the **Project** tab of the **Project Properties** dialog:



The three additional fields on this dialog, and their use, are as follows:

Catalina Options: This is a multi-line text box that is used to specify all the project options. They should be separated by spaces or new line characters (i.e. not commas). For example:

platform options: **-C C3, -C CUSTOM, -C P2_EVAL** etc

library options: **-lc, -lci, -lcx, -lcix, -lm, -lma, -lmb, -lmc, -lthreads, -linterrupts** etc

memory model options: **-C TINY, -C LARGE, -C SMALL, -C COMPACT, -C NATIVE** etc

miscellaneous options: **-p2, -O5, -g3, -e** etc

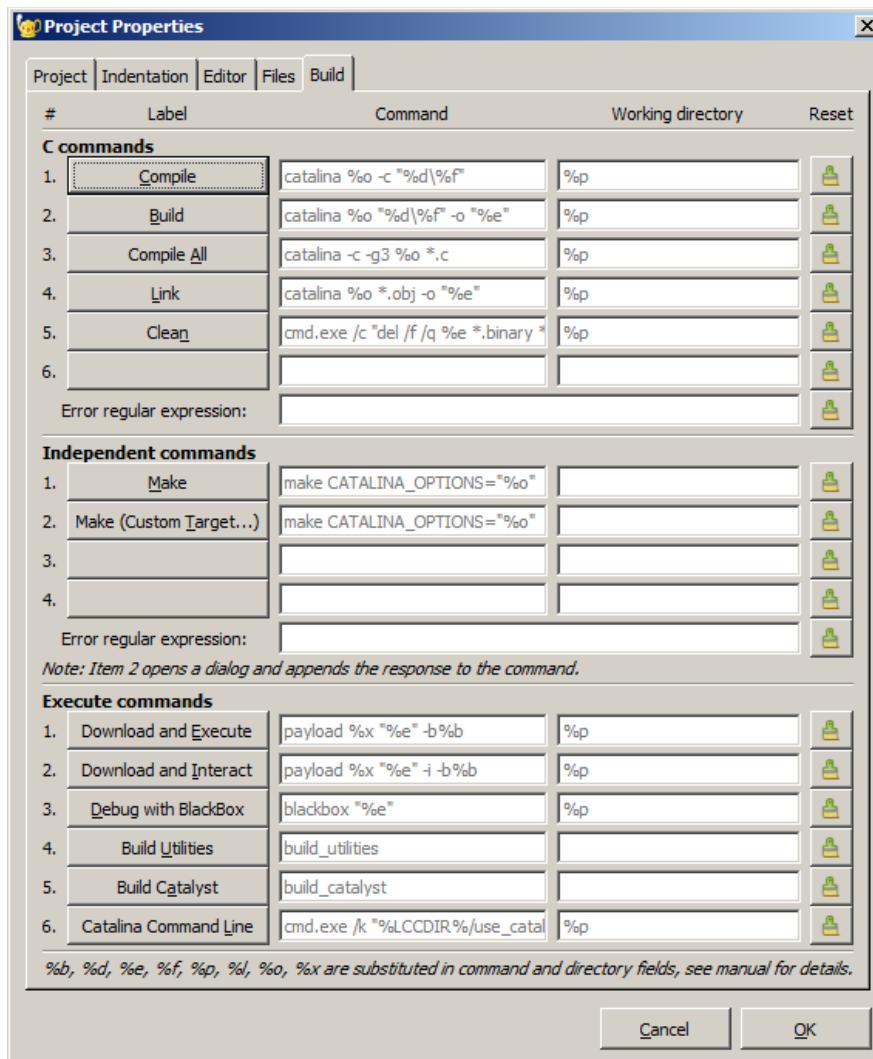
See the Catalina documentation for a full set of such options. The value in this field can be used in build commands using **%o** (see below for Catalina Build Commands).

Baudrate: This is the baudrate for payload to use. On the Propeller 1, this would usually be **115200** (which is the default), but on the Propeller 2 it may need to be manually specified (e.g. as **230400**). The value in this field can be used in build commands using **%b** (see below for Catalina Build Commands).

Loader: This is for Propeller 1 XMM programs that require a special loader (which must be build using the **build_utilities** batch script). Possible values include **XMM, SRAM, FLASH, or EEPROM**. This field should be left blank for normal Propeller 1 and Propeller 2 Hub RAM loads. The value in this field can be used in build commands using **%x** (see below for Catalina Build Commands).

Catalina Build Commands

The other change to the standard Geany IDE is in the form of customized build commands. Here is the **Build** tab of Catalina's version of the **Project Properties** dialog for C programs (this version is for Windows – see later for the Linux version, which has very slight command differences):

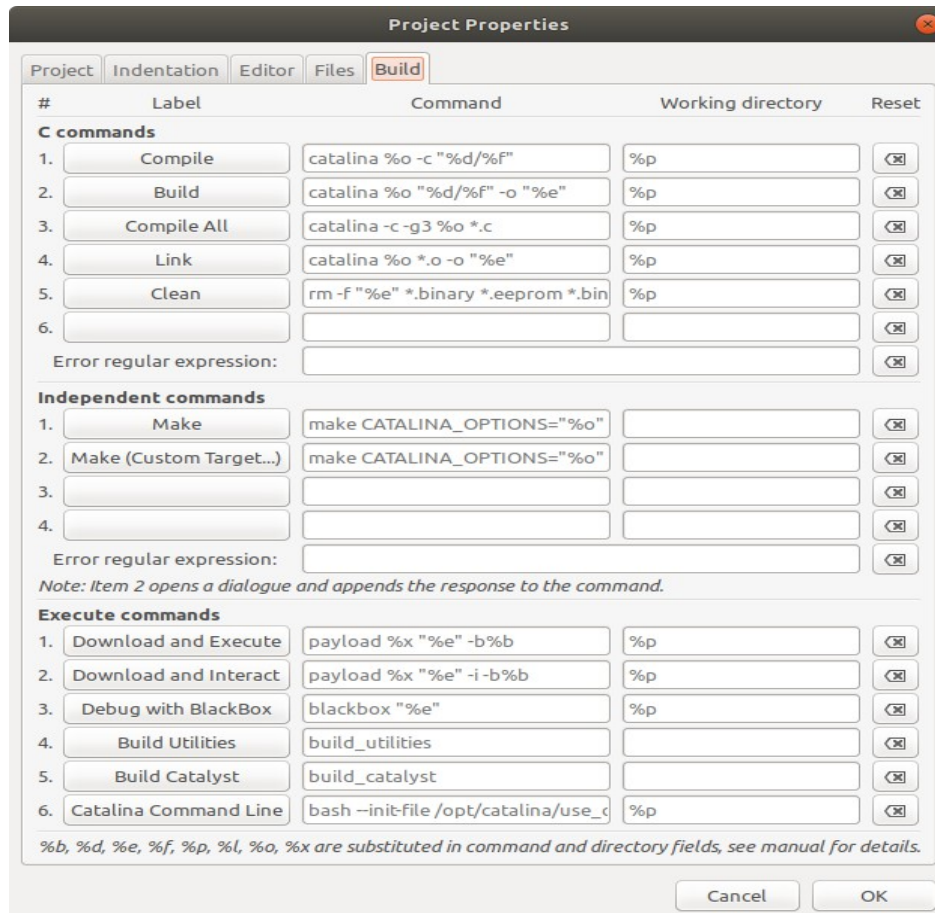


The first thing to note is that Catalina has many *more* customizable build commands than the normal Geany IDE. This reflects the many more things that must typically be done to support a separate external processor.

The default values for the customized build commands for C files are as shown. (in grey text – if they are modified from the default commands, they are shown in black text).

Note the use of the standard Geany IDE placeholders: **%d**, **%e**, **%f** and **%p**, plus the Catalina-specific additions: **%b**, **%o** and **%x**.

Here is the Linux version of the Build commands:



The build commands can be customized per project if required.

For example, if you decide you would rather use the BlackCat debugger rather than BlackBox for debugging a particular project, simply replace the command **blackbox "%e"** in Execute command 3 with a command like **blackcat -P COM9 -D "%e"** (note that BlackCat does not detect the port to use automatically - it must be specified). Note that the name of the command can be modified as well - just press the command button.

Note that Catalina retains the Geany IDE **make** commands, which can be used to make complex projects - but you must install **make** separately. Neither Catalina nor the Geany IDE include a version of **make**. The easiest way to install **make** in Windows is by installing **MinGW**

(see www.mingw.org). When using **make**, note that the Catalina options are passed in to **make** using the **CATALINA_OPTIONS** variable.

Note that Geany is an open source IDE. The source code for the changes to the standard Geany IDE are included in each Catalina release. See the file “catalina_geany_source.zip” in the catalina_geany subdirectory.