A Tutorial on using *Code::Blocks* with Catalina 3.0.3

Basic Concepts

Like nearly all compilers, Catalina is essentially a command-line tool.

Fortunately, for Catalina users who don’t like using command-line tools, or would just *prefer* to use a Integrated Development Environment (IDE), there is **Code::Blocks** - a professional graphical user interface that can be used as a front end for almost *any* C compiler.

Code::Blocks provides a complete environment for developing C programs. It provides support for complex project structures, language sensitive editing, auto-completion, symbol management, intelligent search functions, automatic build capabilities and much more. Code::Blocks includes support for many different C compilers. This support is implemented via *plugins*.

Catalina provides an enhanced Code::Blocks Compiler plugin which adds support for the Catalina compiler. A project wizard for creating Catalina projects is also provided, as are a set of 'tool' wrappers that allow the use of various Catalina utilities (such as **payload** and **blackbox**) from within Code::Blocks.

Version 3.0.3 (and later) of Catalina incorporates several enhancements specifically intended to simplify the use of Catalina from within Code::Blocks (e.g. enhancements to **blackbox** to allow it to automatically find the port to be used when debugging).

This document provides details on installing and using Code::Blocks with Catalina. It is not a tutorial on either **C**, **Catalina** or **Code::Blocks**. It just provides enough information to get Code::Blocks up and running, and use it to compile Catalina programs.

Prerequisites

This document assumes you are installing Code::Blocks for use with Catalina release **3.0.3** or later, and that you already have this version of Catalina installed in its default location. If you have not already installed Catalina, you should do that first - see the Catalina documentation for details.

If you have Catalina installed in a non-default location, the installation will still work correctly provided you set the environment variable **LCCDIR** to point to the main Catalina directory before starting the installation of Code::Blocks (otherwise you will have to manually configure Code::Blocks to tell it where Catalina is installed).

Catalina **3.0.3** supports only Code::Blocks version **10.05**. Previous versions of Catalina supported Code::Blocks version 8.02, but if you have that version installed you will need to upgrade it to version 10.05 before you can install the Catalina support described in this document.

Like Catalina itself, Code::Blocks can be installed and used under either Linux or Windows. This document applies to both platforms, with any differences (mainly during the installation process) noted.

Installing and Configuring Code::Blocks

Step 1 – Extract the Components

In the **codeblocks** directory of the Catalina distribution are two compressed files – one specifically for *Windows* and another one for *Linux*. Each one contains:

- a file called **compiler-0.99.cbplugin** (or similar), which is a replacement for the standard Code::Blocks Compiler plugin, and includes Catalina support;
- a sub-directory called **templates**, which contains the files necessary to add the Catalina Project Wizard to Code::Blocks; and
- a file called **catalina_tools.conf**, which contains the definition of menu entries for invoking various Catalina utilities from within Code::Blocks.

Extract the Catalina files from the appropriate archive for your platform – i.e. either **codeblocks_win32.zip** (which when unzipped will create a *Windows* folder) or **codeblocks_Linux.tgz** (which when uncompressed will create a *Linux* folder).

Step 2 – install Code::Blocks

Catalina does not include a copy of Code::Blocks itself. You must download version **10.05** of Code::Blocks from http://www.codeblocks.org. The Code::Blocks support included with this release of Catalina will *not work* with any previous version, and if you have such a version of Code::Blocks installed you should remove it before installing version 10.05.

Note that you only need a *binary* distribution of Code::Blocks – you do not need to build either Code::Blocks (or Catalina) from source to use Catalina with Code::Blocks. Binary distributions for Code::Blocks are available from the Code::Blocks web site for many Windows and Linux platforms.

Note that during the installation of Code::Blocks you can select various options, including which plugins to install and which C compiler to set as default. However, the standard distribution of Code::Blocks doesn't know about Catalina yet, so if you are prompted to select a C compiler during the installation process, you can select any of the compilers listed (such as the GNU GCC compiler).

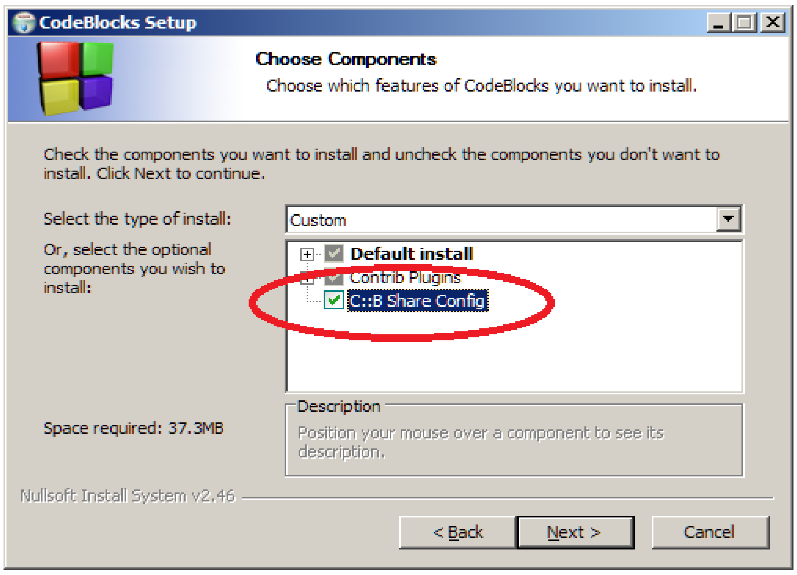
Separate instructions for Windows and Linux installation are given separately below.

Windows Installation

Under Windows, you simply run the Code::Blocks setup program and follow the instructions.

There are only two special things to note – the first is that you must install the **C:B Share Config** component, since we will need to use it later.

This component can be selected in the **Choose Components** dialog:



Choose which features of CodeBlocks you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select the type of install: Custom

Or, select the optional components you wish to install:

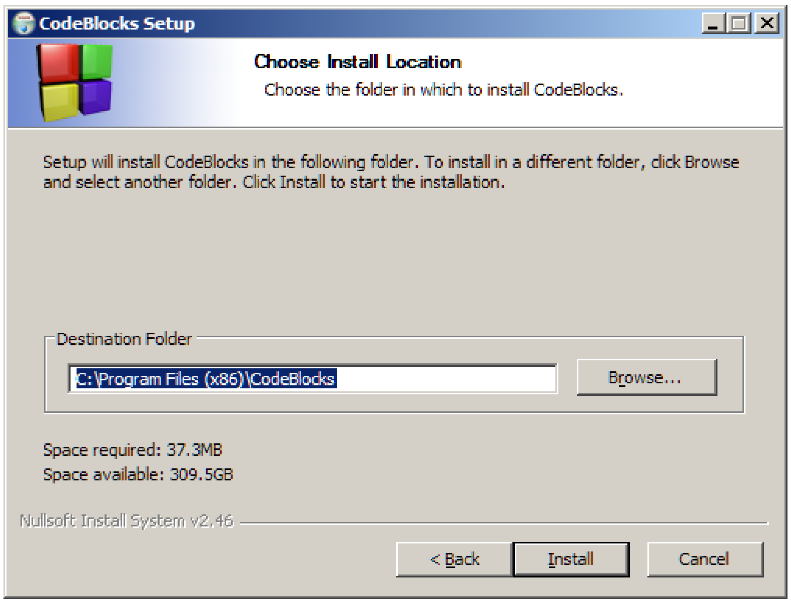
- Default install
- Contrib Plugins
- C::B Share Config ✓

Space required: 37.3MB

Description

Position your mouse over a component to see its description.

The second is that you should make a note of where Code::Blocks is installed, since you will need this information later. This information is displayed during installation in the **Choose Install Location** dialog:



Choose the folder in which to install CodeBlocks.

Setup will install CodeBlocks in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder

C:\Program Files (x86)\CodeBlocks

Space required: 37.3MB

Space available: 309.5GB

Linux Installation

Under Linux, the easiest way to install Code::Blocks is by using **yum** (or the equivalent package installer for your Linux distribution). You will probably want to install both **codeblocks** and **codeblocks-contrib** (which contains additional useful Code::Blocks plugins). For example:

```
    yum install codeblocks-10.05 codeblocks-contrib-10.05
```

The Linux version of the **C::B Share Config** utility (called **cb_share_config**) is installed automatically.

Step 3 – Update the Code::Blocks permissions

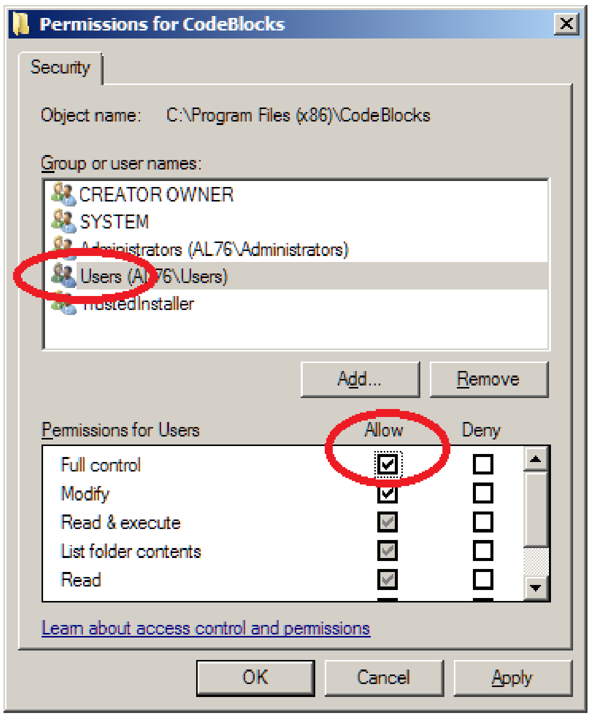
As initially installed, Code::Blocks does not know about the Catalina compiler. We therefore need to subsequently install an updated version of the Compiler plugin which adds Catalina support. This updated version also supports all the existing Code::Blocks compilers, so it is safe to install it even if you already have Code::Blocks installed for use with another compiler.

However, before we can update any Code::Blocks plugins, we need to grant the current user the required access to the Code::Blocks installation directory.

The method to do this is different for Windows and Linux installations. Each is described separately, below.

Windows Installation

The simplest way to set the required permissions is to open Windows Explorer to the location you just installed Code::Blocks, right click on the main CodeBlocks folder (where you just installed it) and select **Properties**. In the Properties dialog box, select the **Security** tab, then press the **Edit** button. Select the **Users** node and then select **Full Control** in the **Allow** column. Apply the changes and close the dialog. The method of doing this may vary slightly under different versions of Windows:


Permissions for CodeBlocks
Security
Add...
Remove
Learn about access control and permissions
OK
Cancel
Apply

Object name: C:\Program Files (x86)\CodeBlocks

Group or user names:
- CREATOR OWNER
- SYSTEM
- Administrators (AL76\Administrators)
- Users (AL76\Users)
- TrustedInstaller

Permissions for Users	Allow	Deny
Full control	✓	☐
Modify	✓	☐
Read & execute	✓	☐
List folder contents	✓	☐
Read	✓	☐

Linux Installation

The precise command to execute will depend on where Code::Blocks is installed in your Linux distribution, and also whether you have a 32 or 64 bit version of Linux and CodeBlocks.

In any case, the simplest way to update the permissions is to locate the codeblocks **lib** directory, and then execute a command similar to the following (this must be executed as **root)**:

```
chmod -R a+w /usr/lib/codeblocks
```

or:

```
chmod -R a+w /usr/lib64/codeblocks
```

or:

```
chmod -R a+w /usr/local/lib/codeblocks
```

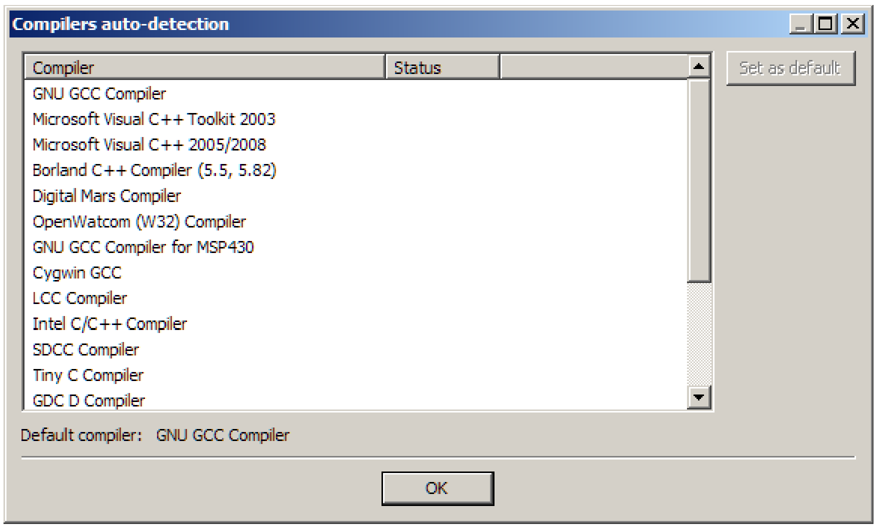
or:

```
chmod -R a+w /usr/local/lib64/codeblocks
```

If you cannot find the appropriate directory, you can still update the Compiler plugin – but you will have to do so while running Code::Blocks as **root**.

Step 4 – Update the Code::Blocks Compiler plugin

Now start Code::Blocks. If this is the first time you have run Code::Blocks you may see a **Compilers auto-detection** dialog. This dialog will not show Catalina yet, as the standard Code::Blocks Compiler plugin does not know how to detect it:



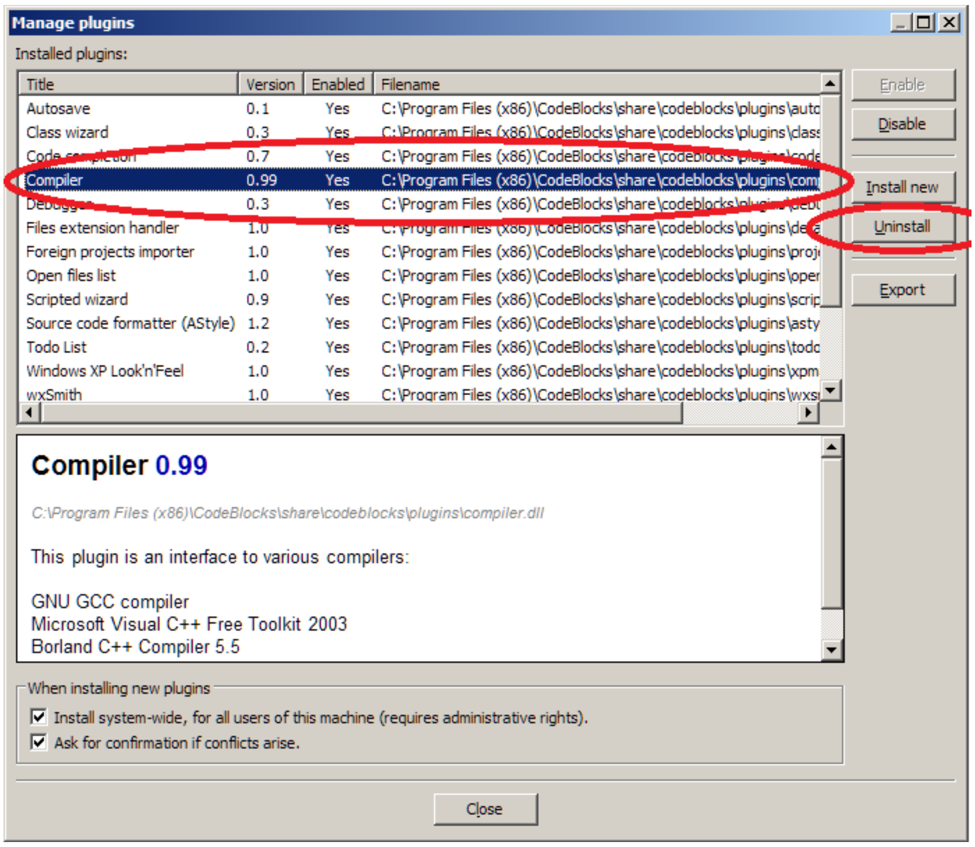
Compiler	Status
GNU GCC Compiler	
Microsoft Visual C++ Toolkit 2003	
Microsoft Visual C++ 2005/2008	
Borland C++ Compiler (5.5, 5.82)	
Digital Mars Compiler	
OpenWatcom (W32) Compiler	
GNU GCC Compiler for MSP430	
Cygwin GCC	
LCC Compiler	
Intel C/C++ Compiler	
SDCC Compiler	
Tiny C Compiler	
GDC D Compiler	

Default compiler: GNU GCC Compiler

You don't need to select a compiler at this point (although it does not matter if you do) – just press **OK**.

Also, it doesn't matter if this dialog does not appear (e.g. if you chose not to include the standard Code::Blocks Compiler plugin during the installation process).

Next, from the main Code::Blocks menu, select **Plugins -> Manage Plugins …** A dialog box similar to the following will appear:



Installed plugins:

Title	Version	Enabled	Filename
Autosave	0.1	Yes	C:\Program Files (x86)\CodeBlocks\share\codeblocks\plugins\auto
Class wizard	0.3	Yes	C:\Program Files (x86)\CodeBlocks\share\codeblocks\plugins\class
Code completion	0.7	Yes	C:\Program Files (x86)\CodeBlocks\share\codeblocks\plugins\code
Compiler	0.99	Yes	C:\Program Files (x86)\CodeBlocks\share\codeblocks\plugins\com
Debugger	0.3	Yes	C:\Program Files (x86)\CodeBlocks\share\codeblocks\plugins\debu
Files extension handler	1.0	Yes	C:\Program Files (x86)\CodeBlocks\share\codeblocks\plugins\de
Foreign projects importer	1.0	Yes	C:\Program Files (x86)\CodeBlocks\share\codeblocks\plugins\proj
Open files list	1.0	Yes	C:\Program Files (x86)\CodeBlocks\share\codeblocks\plugins\oper
Scripted wizard	0.9	Yes	C:\Program Files (x86)\CodeBlocks\share\codeblocks\plugins\scrip
Source code formatter (AStyle)	1.2	Yes	C:\Program Files (x86)\CodeBlocks\share\codeblocks\plugins\asty
Todo List	0.2	Yes	C:\Program Files (x86)\CodeBlocks\share\codeblocks\plugins\todo
Windows XP Look'n'Feel	1.0	Yes	C:\Program Files (x86)\CodeBlocks\share\codeblocks\plugins\xpm
wxSmith	1.0	Yes	C:\Program Files (x86)\CodeBlocks\share\codeblocks\plugins\wxsı

Compiler 0.99

C:\Program Files (x86)\CodeBlocks\share\codeblocks\plugins\compiler.dll

This plugin is an interface to various compilers:

GNU GCC compiler
Microsoft Visual C++ Free Toolkit 2003
Borland C++ Compiler 5.5

When installing new plugins

☑ Install system-wide, for all users of this machine (requires administrative rights).
☑ Ask for confirmation if conflicts arise.

If this dialog shows a version of the **Compiler** plugin already installed, select it and press **Uninstall**. Then press **Install new** to install the new version of the plugin. The new version of the plugin is located in the Catalina codeblocks folder for your operating system. For example (for both 32 bit and 64 bit Windows):

```
C:\Program Files\Catalina\codeblocks\Windows\compiler-0.99.cbplugin
```

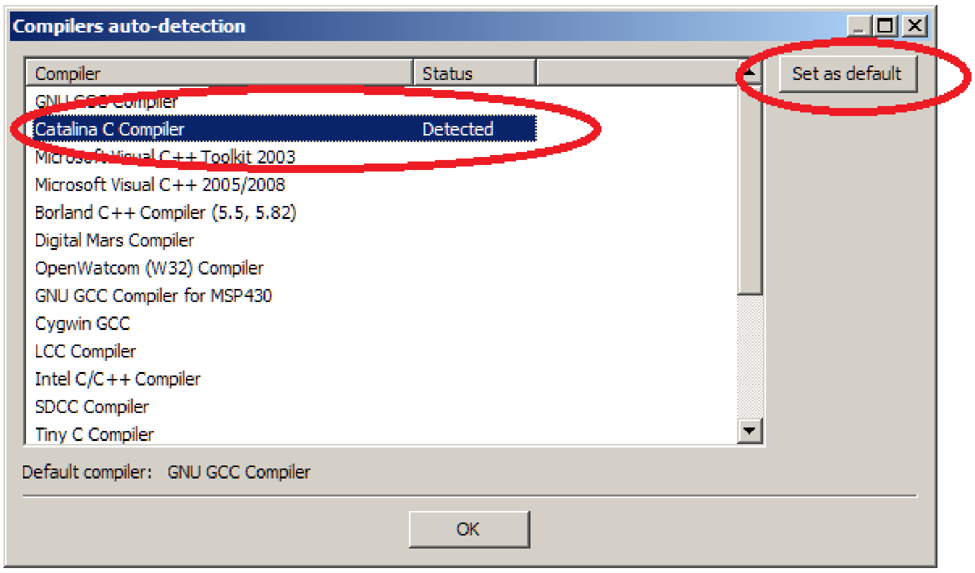
or (for 32 bit Linux)

```
/usr/local/lib/catalina/codeblocks/Linux/32_bit/libcompiler-0.99.cbplugin
```

or (for 64 bit Linux)

```
/usr/local/lib/catalina/codeblocks/Linux/64_bit/libcompiler-0.99.cbplugin
```

Once you have successfully installed the plugin, the **Compilers auto-detection** may appear again, and if Catalina has been installed correctly, it should now show the Catalina compiler as **Detected**:



Compilers auto-detection

Compiler	Status
GNU GCC Compiler	
Catalina C Compiler	Detected
Microsoft Visual C++ Toolkit 2003	
Microsoft Visual C++ 2005/2008	
Borland C++ Compiler (5.5, 5.82)	
Digital Mars Compiler	
OpenWatcom (W32) Compiler	
GNU GCC Compiler for MSP430	
Cygwin GCC	
LCC Compiler	
Intel C/C++ Compiler	
SDCC Compiler	
Tiny C Compiler	

Default compiler: GNU GCC Compiler

In this dialog, you can select the **Catalina C Compiler** and press **Set as Default**.

If the dialog box does not appear, or it does not show Catalina as **Detected**, don't worry – you can configure the default compiler later from within Code::Blocks itself.

Step 5. Install the Catalina Wizard files

For this step, we must first close Code::Blocks, locate the Code::Blocks **templates** directory, and then copy the Catalina Project Wizard files from the Catalina codeblocks directory.

Windows Installation

Under Windows you should copy the contents of the **templates** folder from your Catalina distribution, such as:

```
C:\Program Files\Catalina\codeblocks\Windows\templates
```

to the CodeBlocks templates folder, which will be in a location off the main CodeBlocks installation folder such as:

```
C:\Program Files (X86)\CodeBlocks\share\CodeBlocks\templates
```

You will need Administrator privileges to do this copy. Make sure to copy all the files in the source folder (i.e. copy recursively).

Linux Installation

Under Linux you should copy the contents of the **templates** folder from your Catalina distribution, such as:

```
/usr/local/lib/catalina/codeblocks/Linux/templates
```

to the CodeBlocks templates folder, which will be in a location off the main CodeBlocks installation folder such as:

```
/usr/share/codeblocks/templates
```

or:

```
/usr/local/share/codeblocks/templates
```

You will need to be **root** to do this copy. Make sure to copy all the files in the source folder (i.e. copy recursively).

Step 6 – Install the Catalina tools

Code::Blocks allows additional tools to be configured as menu items, and it is convenient to add Catalina tools such as **payload** and **blackbox** to the menu so that you can use them from within Code::Blocks. To install them, we will use the Code::Blocks **C:B Share Config** utility. This utility is separate to Code::Blocks itself, which should remain closed for this step.

Windows Installation

You access the **C:B Share Config** utility via the Windows Start menu.

In Windows XP:

Start->Programs->CodeBlocks->CB Share Config

In Windows Vista or Windows 7:

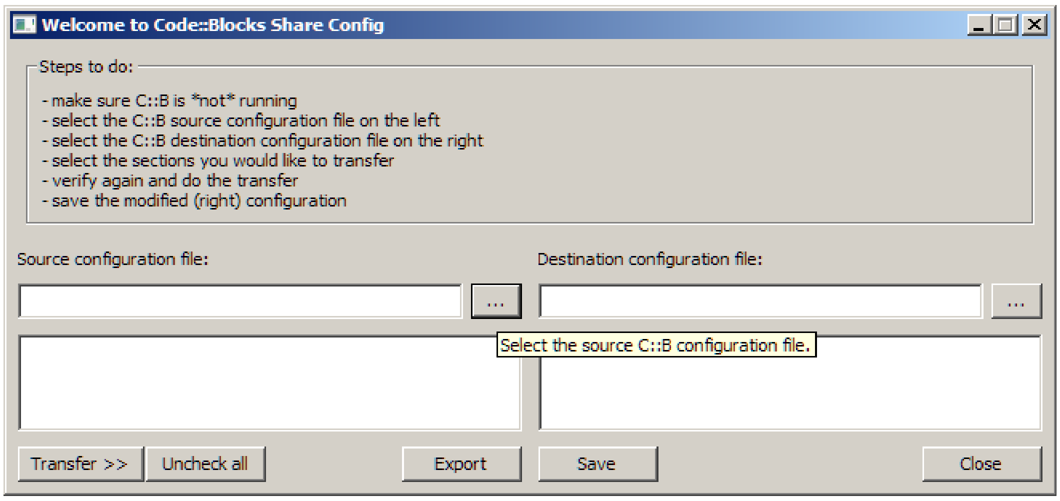
Start->All Programs->CodeBlocks->CB Share Config

Linux Installation

You must access the **C:B Share Config** utility via the command-line:

```
cb_share_config
```

In both Windows and Linux, a dialog box similar to the following will appear:


Welcome to Code::Blocks Share Config

Steps to do:

- make sure C::B is *not* running
- select the C::B source configuration file on the left
- select the C::B destination configuration file on the right
- select the sections you would like to transfer
- verify again and do the transfer
- save the modified (right) configuration

Source configuration file:

Destination configuration file:

Select the source C::B configuration file.

In the left panel select the file **catalina_tools.conf** from the Catalina codeblocks directory appropriate to your operating system – i.e. (in Windows):

```
C:\Program Files\Catalina\codeblocks\Windows\catalina_tools.conf
```

or (in Linux):

```
/usr/local/lib/catalina/codeblocks/Linux/catalina_tools.conf
```

In the right panel you need to locate and select your Code::Blocks configuration file. This will be located in your configuration data folder.

For example (in Windows XP):

```
C:\Documents and Settings\<your_name>\Application Data\
   codeblocks\default.conf
```

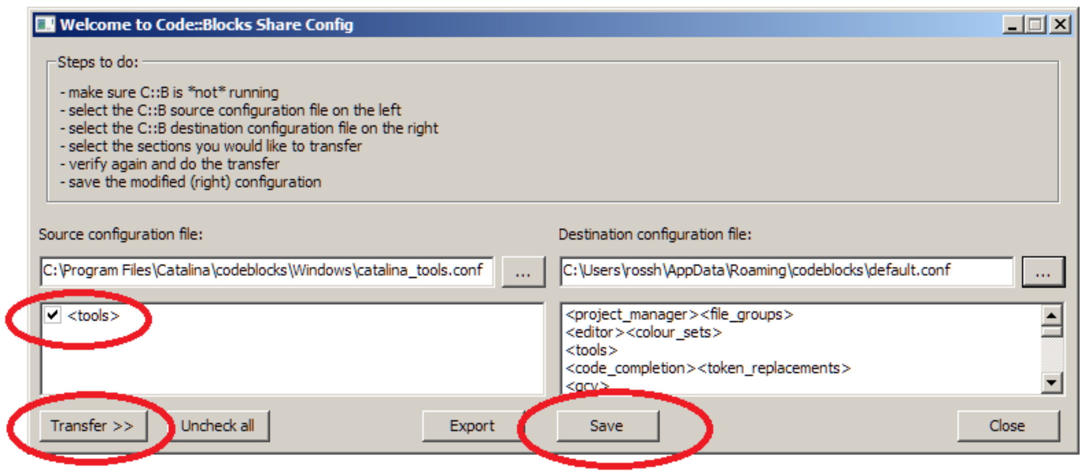
or (in Windows Vista or Windows 7):

```
C:\Users\<your_name>\AppData\Roaming\codeblocks\default.conf
```

or (in Linux)[1]:

```
/home/<your_name>/.codeblocks/default.conf
```

In the left pane, check the **<tools>** node (it may be the only entry) and press **Transfer>>** to copy the tools entries to your own **default.conf** file:



Welcome to Code::Blocks Share Config

Steps to do:

- make sure C::B is *not* running
- select the C::B source configuration file on the left
- select the C::B destination configuration file on the right
- select the sections you would like to transfer
- verify again and do the transfer
- save the modified (right) configuration

Source configuration file: C:\Program Files\Catalina\codeblocks\Windows\catalina_tools.conf

Destination configuration file: C:\Users\rossh\AppData\Roaming\codeblocks\default.conf

[✓] <tools>

```
<project_manager><file_groups>
<editor><colour_sets>
<tools>
<code_completion><token_replacements>
<gcv>
```

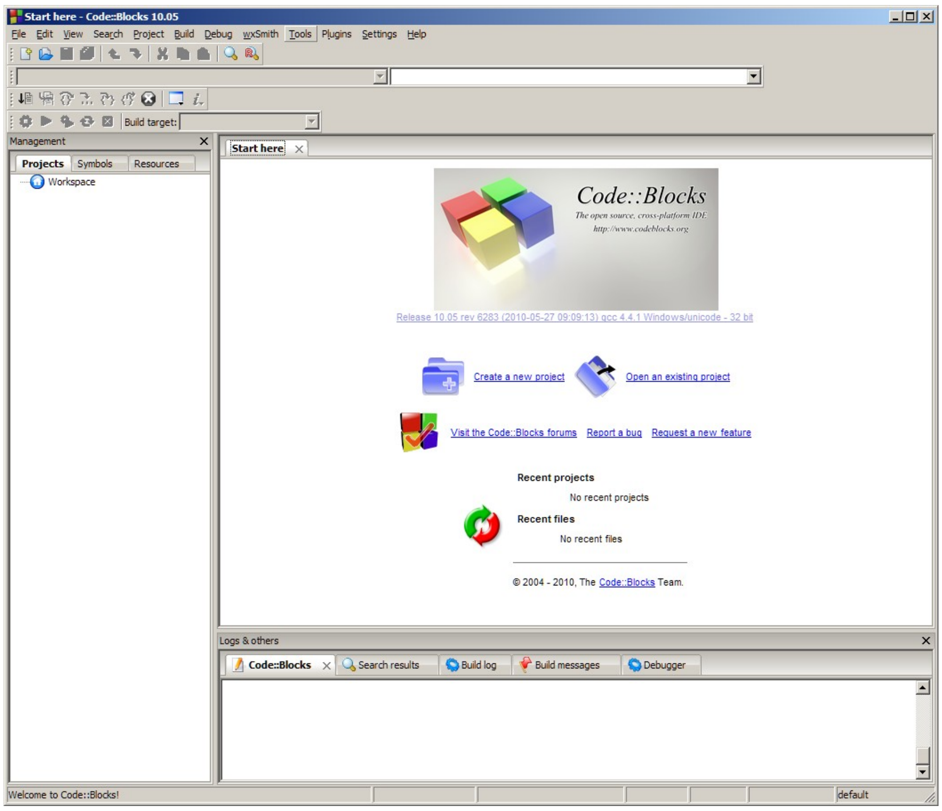
Confirm the transfer, then press **Save** to save the destination configuration file. Then close the **CB Share Config** utility.

The configuration of Code::Blocks for Catalina is now complete - let's start using it!

[1] Note that on some versions of Linux, you will need to explicitly select **Show Hidden Files** in the file selection dialog box to see folders such as **.codeblocks**

Using Code::Blocks

When you first start Code::Blocks, the main window will look something like the following:



Release 10.05 rev 6283 (2010-05-27 09:09:13) gcc 4.4.1 Windows/unicode - 32 bit

No recent projects

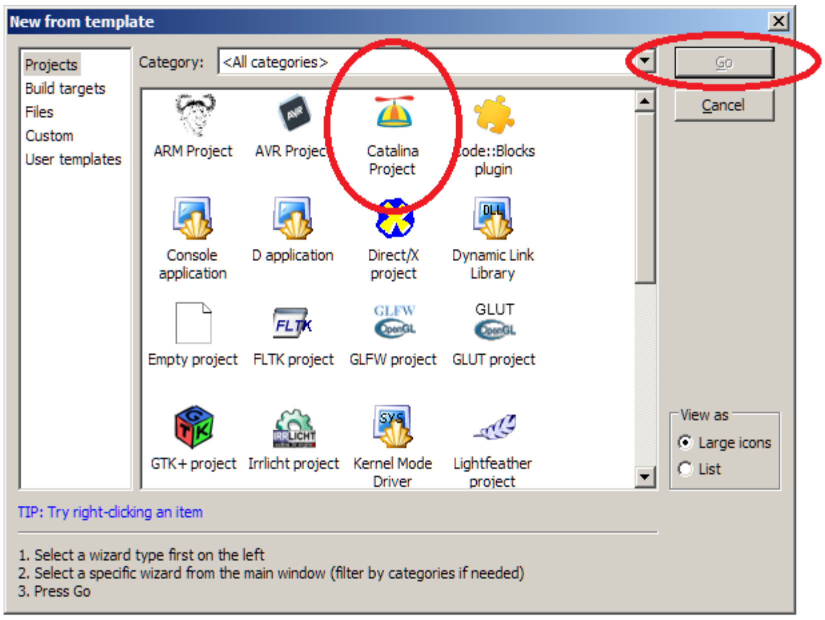
No recent files

© 2004 - 2010, The Code::Blocks Team.

Welcome to Code::Blocks!

Building your first Catalina Project

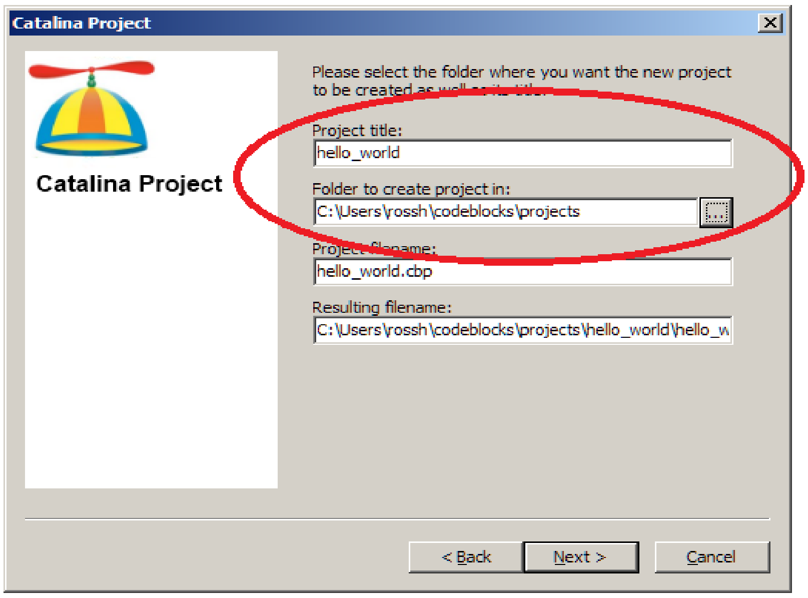
There are various ways to create a new project from within Code::Blocks, but the simplest way to get started is use the **Catalina Project Wizard**. To do this, select the entry **File->New->Project...** from the main menu – a dialog box similar to the one shown below will appear. Select **Catalina Project** and press **Go**:



TIP: Try right-clicking an item

1. Select a wizard type first on the left
2. Select a specific wizard from the main window (filter by categories if needed)
3. Press Go

After displaying a welcome page (just press **Next >**) Code::Blocks will display a page asking you for a title (name) and a directory for your project. Choose a suitable name for your project (don't use any embedded spaces) and where you want the directory for this project to be created (this can be anywhere, but you may choose to use the same directory for all your Catalina projects):



Catalina Project

Please select the folder where you want the new project to be created as well as its title.

Project title: hello_world

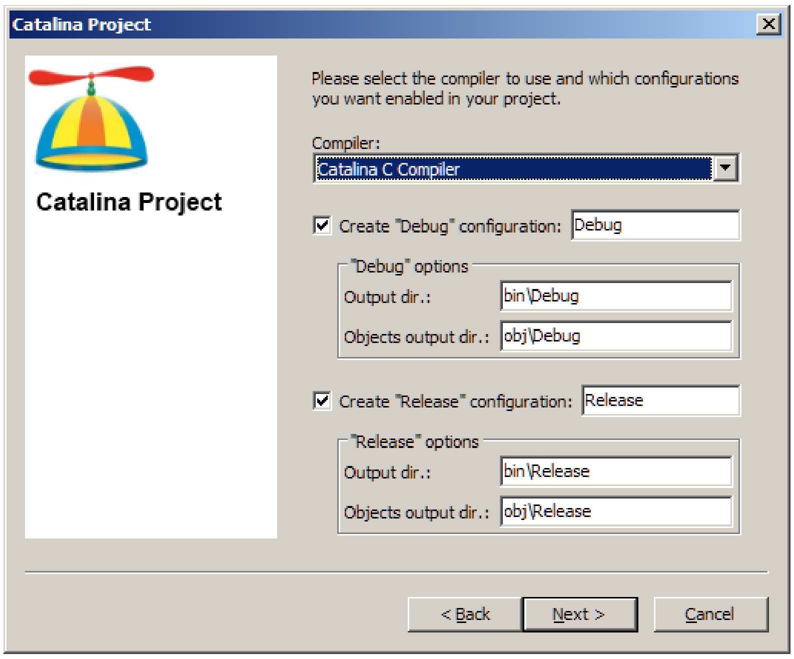
Folder to create project in: C:\Users\rossh\codeblocks\projects

Project filename: hello_world.cbp

Resulting filename: C:\Users\rossh\codeblocks\projects\hello_world\hello_w

Then press **Next >**

On the next page, Code::Blocks will display some information regarding various project options, such as what compiler to use (Catalina!) and whether to create both a release and a debug version of your program:



Catalina Project

Please select the compiler to use and which configurations you want enabled in your project.

Compiler: Catalina C Compiler

☑ Create "Debug" configuration: Debug

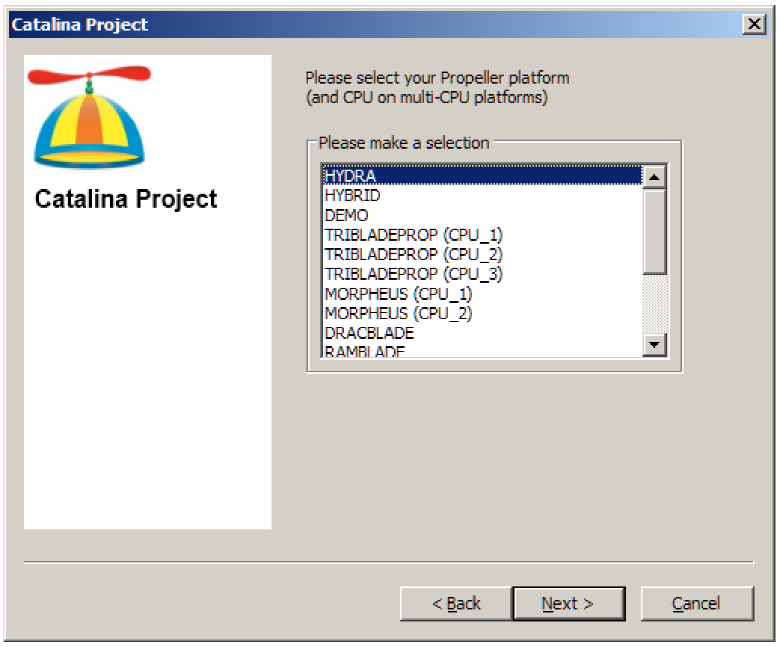
"Debug" options
Output dir.: bin\Debug
Objects output dir.: obj\Debug

☑ Create "Release" configuration: Release

"Release" options
Output dir.: bin\Release
Objects output dir.: obj\Release

At this stage, there is no reason to change anything here, so just press **Next >** again.

On the next page, things start to get interesting. Here you can choose the Propeller platform you have. Select the most appropriate entry from the list (some platforms have multiple entries, if they can be configured various ways) and press **Next >**

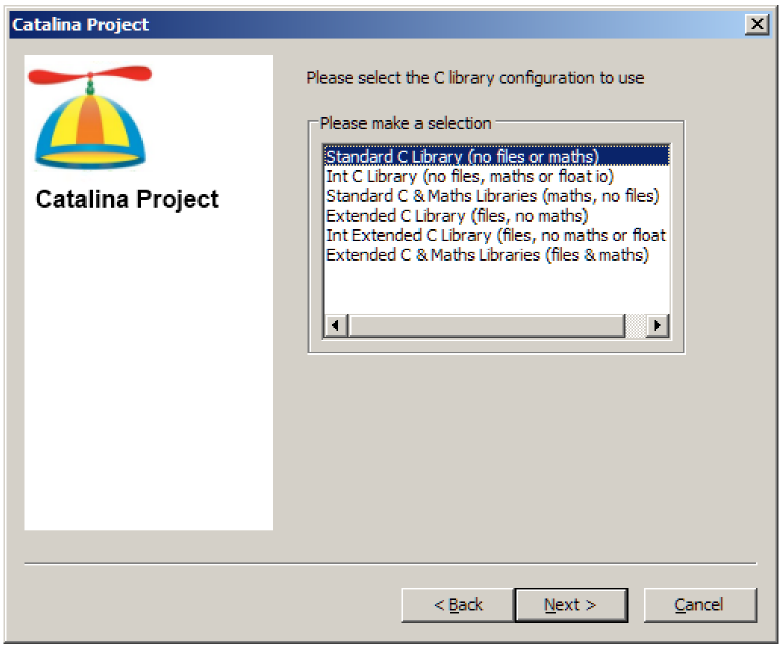


Please select your Propeller platform
(and CPU on multi-CPU platforms)

Please make a selection

HYDRA
HYBRID
DEMO
TRIBLADEPROP (CPU_1)
TRIBLADEPROP (CPU_2)
TRIBLADEPROP (CPU_3)
MORPHEUS (CPU_1)
MORPHEUS (CPU_2)
DRACBLADE
RAMBLADE

On the next page, you can select what basic library configuration you want:



Please select the C library configuration to use

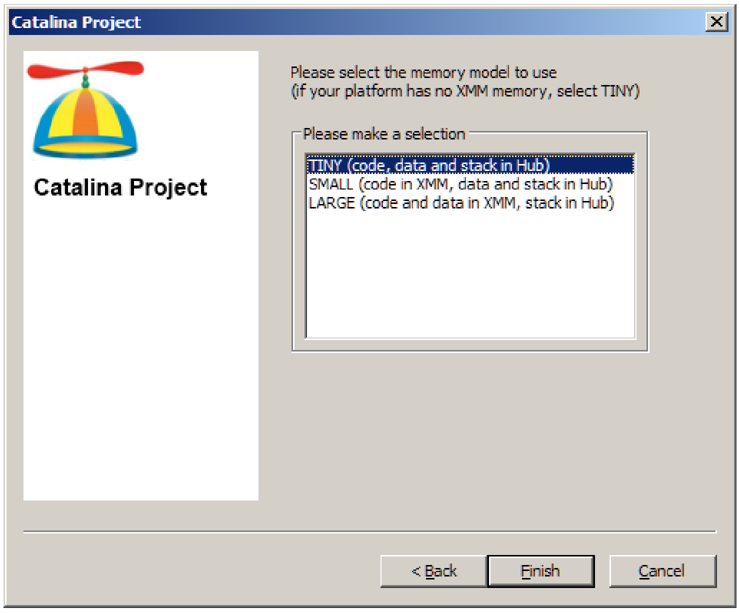
Please make a selection

Standard C Library (no files or maths)
Int C Library (no files, maths or float io)
Standard C & Maths Libraries (maths, no files)
Extended C Library (files, no maths)
Int Extended C Library (files, no maths or float
Extended C & Maths Libraries (files & maths)

For the purposes of this example, select the **Standard C Library** and then press **Next >**

On the next page, you will be able to select what memory model you want to use for this project.

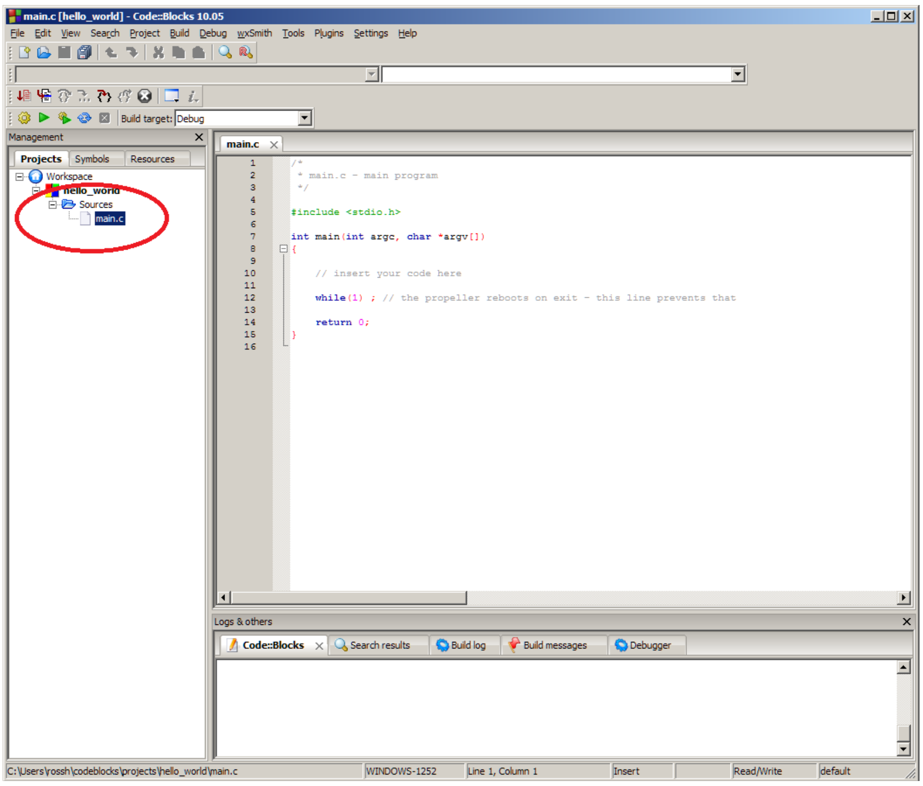
If you have a platform with XMM RAM, you can select any of the Catalina memory models here – but for the purposes of this example, just select the **TINY** memory model (which applies to all Propeller platforms) then press **Finish**



Please select the memory model to use
(if your platform has no XMM memory, select TINY)

Please make a selection
TINY (code, data and stack in Hub)
SMALL (code in XMM, data and stack in Hub)
LARGE (code and data in XMM, stack in Hub)

Now Code::Blocks will create a project for you with the selected options. In the **Projects** tab of the Navigation pane, you will see your project:


main.c [hello_world] - Code::Blocks 10.05
File Edit View Search Project Build Debug wxSmith Tools Plugins Settings Help
Build target: Debug
Management
Projects Symbols Resources
Workspace
hello_world
Sources
main.c
Logs & others
Code::Blocks Search results Build log Build messages Debugger

```
/*
 * main.c - main program
 */

#include <stdio.h>

int main(int argc, char *argv[])
{

    // insert your code here

    while(1) ; // the propeller reboots on exit - this line prevents that

    return 0;
}
```

C:\Users\rossh\codeblocks\projects\hello_world\main.c WINDOWS-1252 Line 1, Column 1 Insert Read/Write default

Open the project node and you will see a node called **Sources**. Open this node and you will see that a file has been automatically created for your project called **main.c** – double click on this file to display it in the source code editor window.

For the purposes of this example, lets simply add a single line to this file in the text editor. Where the file contains the line:

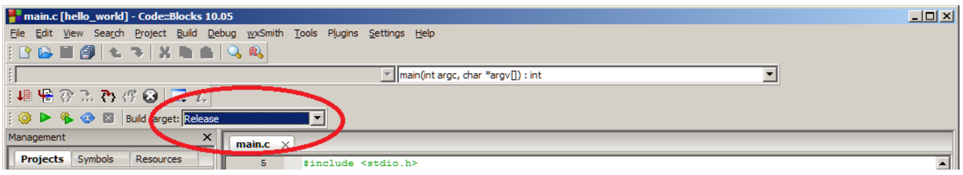
```
// insert your code here
```

we will add the line:

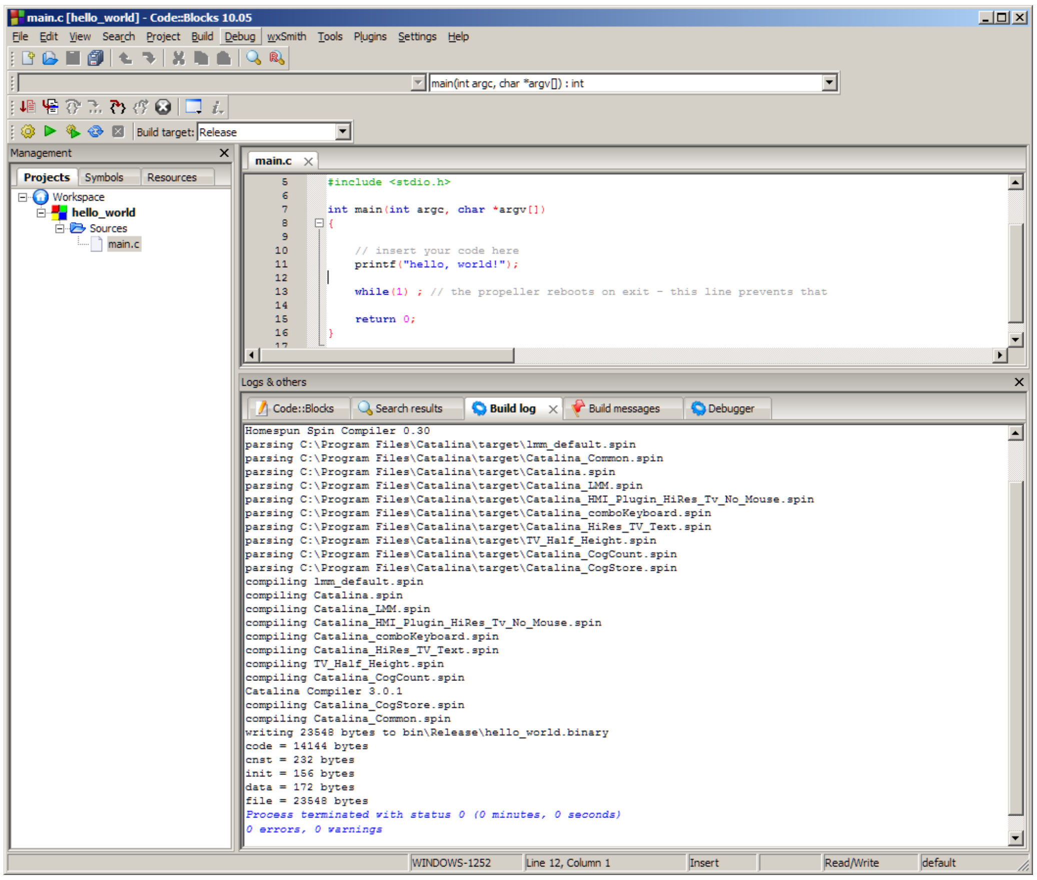
```
printf("hello, world!");
```

As you type this line, you will probably notice that Code::Blocks is anticipating what you intended – for instance, it knows about the **printf()** function, and will prompt you as soon as you have typed a few characters. Just press **TAB** when the prompt is indicating the correct function.

Next, make sure we are working with the Release version by choosing Release in the **Build Target** drop down list, or by selecting **Build->Select Target->Release** from the menu (we will work with Debug versions later):


main.c [hello_world] - Code::Blocks 10.05
File Edit View Search Project Build Debug wxSmith Tools Plugins Settings Help
main(int argc, char *argv[]) : int
Build target: Release
Management
Projects Symbols Resources
main.c
#include <stdio.h>

Then we can compile this project. We can do this by pressing **Ctrl-F9**, or by selecting **Build->Build** from the menus, or by right clicking on the project in the navigation pane and selecting **Build**. The output of the build will be shown in the build log pane. It will look something like this:

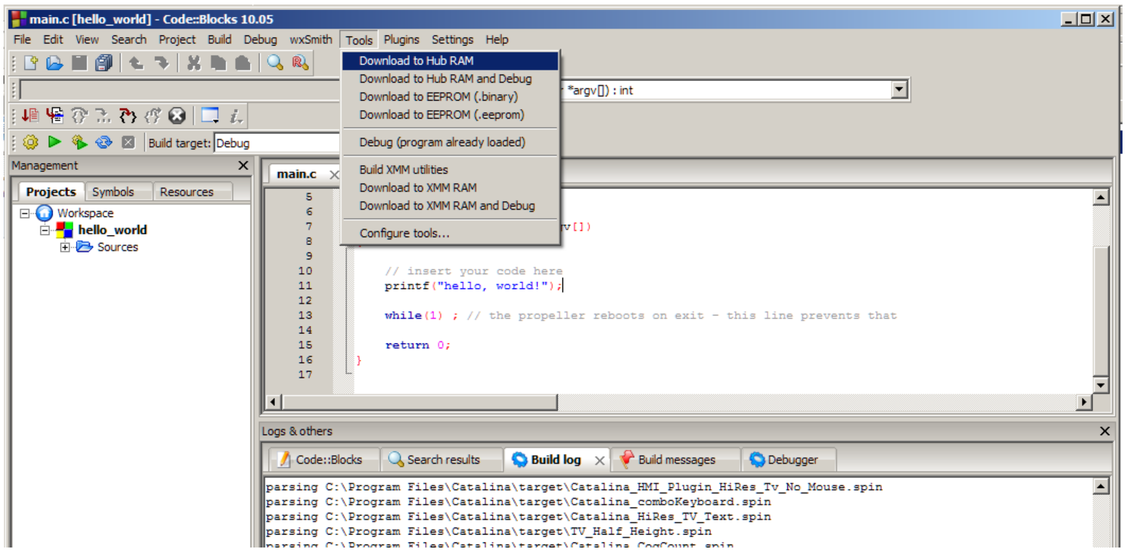


```
#include <stdio.h>

int main(int argc, char *argv[])
{

    // insert your code here
    printf("hello, world!");

    while(1) ; // the propeller reboots on exit - this line prevents that

    return 0;
}
```

```
Homespun Spin Compiler 0.30
parsing C:\Program Files\Catalina\target\lmm_default.spin
parsing C:\Program Files\Catalina\target\Catalina_Common.spin
parsing C:\Program Files\Catalina\target\Catalina.spin
parsing C:\Program Files\Catalina\target\Catalina_LMM.spin
parsing C:\Program Files\Catalina\target\Catalina_HMI_Plugin_HiRes_Tv_No_Mouse.spin
parsing C:\Program Files\Catalina\target\Catalina_comboKeyboard.spin
parsing C:\Program Files\Catalina\target\Catalina_HiRes_TV_Text.spin
parsing C:\Program Files\Catalina\target\TV_Half_Height.spin
parsing C:\Program Files\Catalina\target\Catalina_CogCount.spin
parsing C:\Program Files\Catalina\target\Catalina_CogStore.spin
compiling lmm_default.spin
compiling Catalina.spin
compiling Catalina_LMM.spin
compiling Catalina_HMI_Plugin_HiRes_Tv_No_Mouse.spin
compiling Catalina_comboKeyboard.spin
compiling Catalina_HiRes_TV_Text.spin
compiling TV_Half_Height.spin
compiling Catalina_CogCount.spin
Catalina Compiler 3.0.1
compiling Catalina_CogStore.spin
compiling Catalina_Common.spin
writing 23548 bytes to bin\Release\hello_world.binary
code = 14144 bytes
cnst = 232 bytes
init = 156 bytes
data = 172 bytes
file = 23548 bytes
Process terminated with status 0 (0 minutes, 0 seconds)
0 errors, 0 warnings
```

Assuming the project built correctly (which it should do if you typed the line in correctly!) you can now do things with the project executable that has been generated by selecting various Catalina utilities from the **Tools** menu.

For instance, if you have a Propeller connected to your computer and it is powered on, just select **Tools->Download to Hub RAM**, as shown below:



```
                                v[])

    // insert your code here
    printf("hello, world!");

    while(1) ; // the propeller reboots on exit - this line prevents that

    return 0;
}
```

```
parsing C:\Program Files\Catalina\target\Catalina_HMI_Plugin_HiRes_Tv_No_Mouse.spin
parsing C:\Program Files\Catalina\target\Catalina_comboKeyboard.spin
parsing C:\Program Files\Catalina\target\Catalina_HiRes_TV_Text.spin
parsing C:\Program Files\Catalina\target\TV_Half_Height.spin
parsing C:\Program Files\Catalina\target\Catalina_CogCount.spin
```

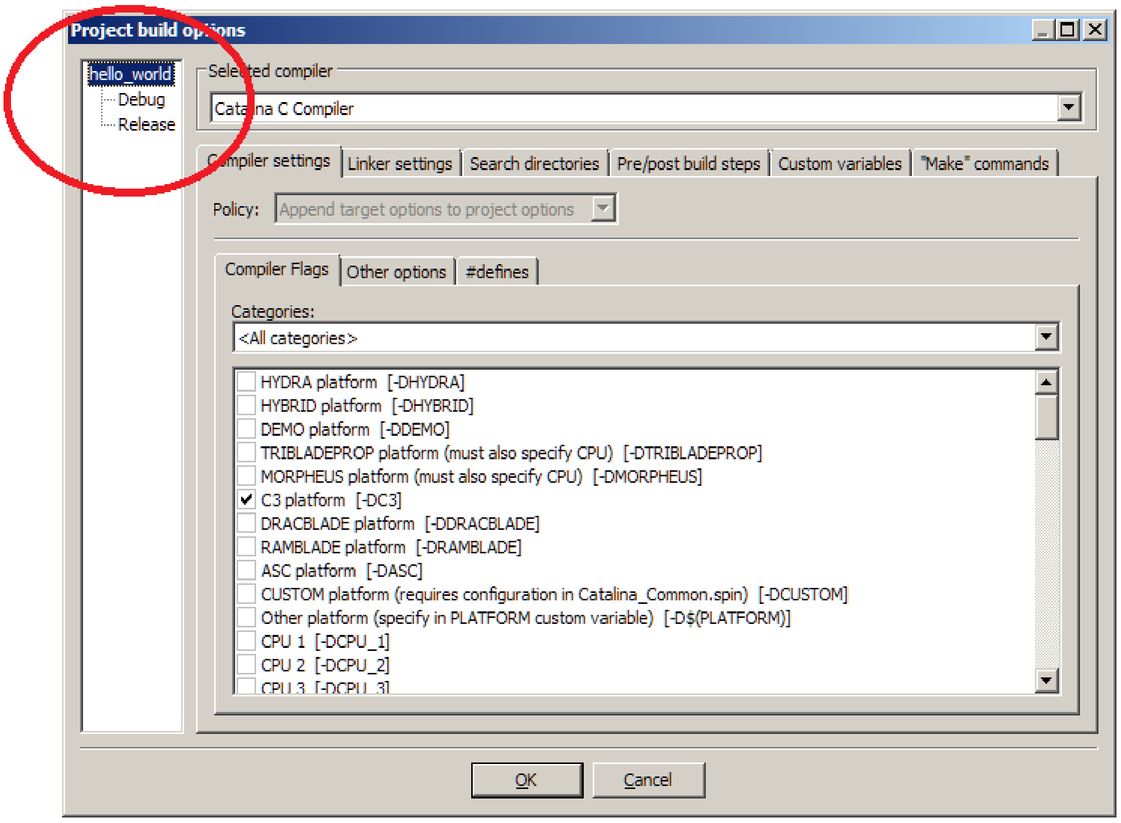
If everything works correctly, after a few seconds you should see your Propeller displaying a friendly greeting!

Manually Specifying Catalina Options

The **Catalina Project Wizard** sets up various basic compiler options for you, but there are many more options that you can set manually in your project. To see them all, select **Project->Build Options …** from the main menu.

A dialog box similar to the one shown below will appear. The most important (and initially confusing) thing about setting options in Code::Blocks is that you can set them at the *Project* level, or at the level of each *Build Target* (e.g. *Release* or *Debug*).

It is generally a good idea to only set all options at the *Project* level unless there is a good reason to do otherwise - so whenever you open this dialog box, it is worthwhile getting into the habit of checking which node is selected, and selecting the *Project* node if it is not already selected:



Selected compiler: Catalina C Compiler

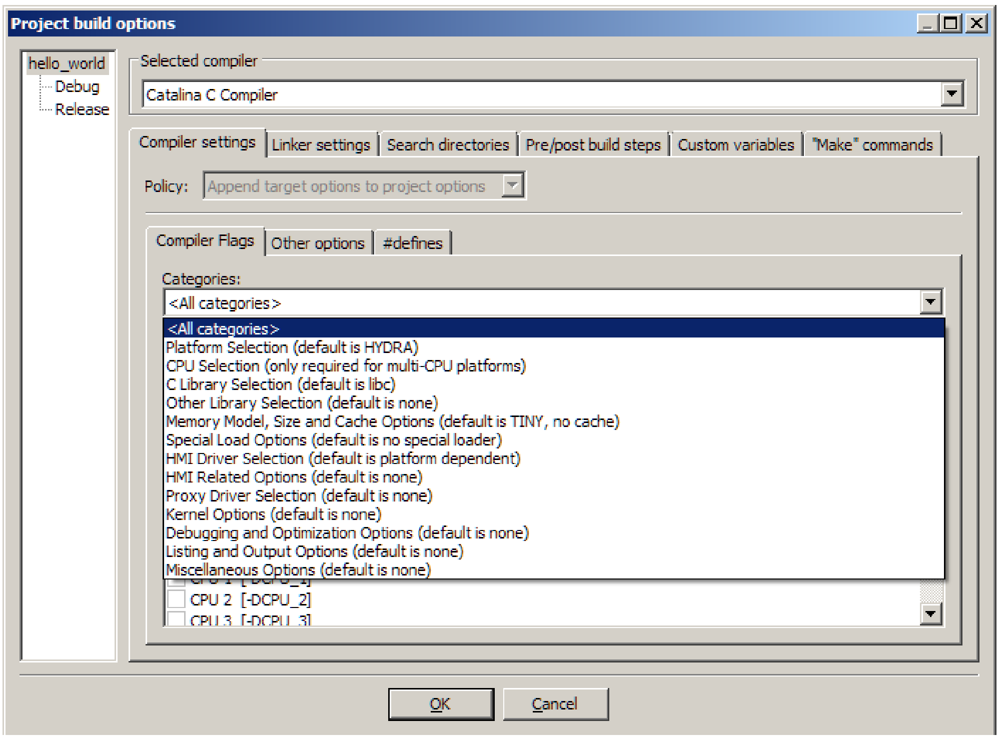
Policy: Append target options to project options

Categories: <All categories>

- [ ] HYDRA platform [-DHYDRA]
- [ ] HYBRID platform [-DHYBRID]
- [ ] DEMO platform [-DDEMO]
- [ ] TRIBLADEPROP platform (must also specify CPU) [-DTRIBLADEPROP]
- [ ] MORPHEUS platform (must also specify CPU) [-DMORPHEUS]
- [✓] C3 platform [-DC3]
- [ ] DRACBLADE platform [-DDRACBLADE]
- [ ] RAMBLADE platform [-DRAMBLADE]
- [ ] ASC platform [-DASC]
- [ ] CUSTOM platform (requires configuration in Catalina_Common.spin) [-DCUSTOM]
- [ ] Other platform (specify in PLATFORM custom variable) [-D$(PLATFORM)]
- [ ] CPU 1 [-DCPU_1]
- [ ] CPU 2 [-DCPU_2]
- [ ] CPU 3 [-DCPU_3]

The selected compiler for all projects created using the **Catalina Project Wizard** will be the **Catalina C Compiler**, and the **Compiler Flags** tab will show a complete list of Catalina compiler options. Some options will already have been selected by the Wizard.

The complete list of compiler options is far too cumbersome to be much use, so instead, choose a particular *category* of options from the **Categories** drop down list:



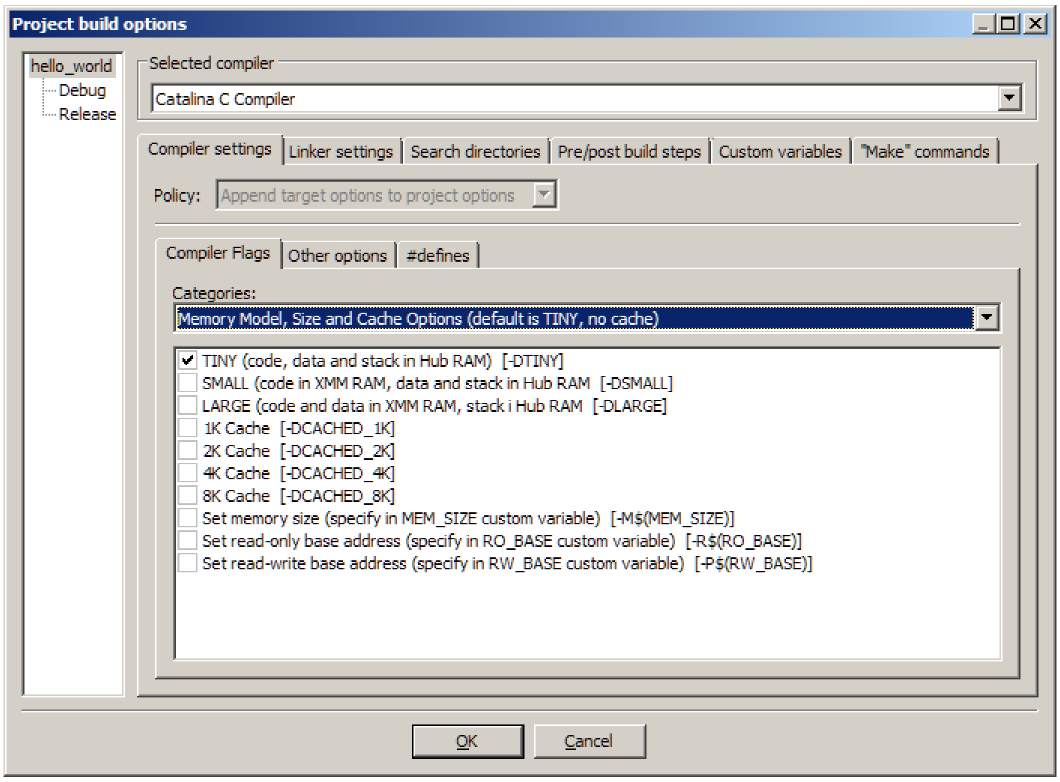
<All categories>
Platform Selection (default is HYDRA)
CPU Selection (only required for multi-CPU platforms)
C Library Selection (default is libc)
Other Library Selection (default is none)
Memory Model, Size and Cache Options (default is TINY, no cache)
Special Load Options (default is no special loader)
HMI Driver Selection (default is platform dependent)
HMI Related Options (default is none)
Proxy Driver Selection (default is none)
Kernel Options (default is none)
Debugging and Optimization Options (default is none)
Listing and Output Options (default is none)
Miscellaneous Options (default is none)

The categories are intended to group the available options by function. The categories are:

- Platform Selection
- CPU Selection
- C Library Selection
- Other Library Selection
- Memory Model, Size and Cache Options
- Special Load Options
- HMI Driver Selection
- HMI Related Options
- Proxy Driver Selection
- Kernel Options
- Debugging and Optimization Options
- Listing and Output Options
- Miscellaneous Options

Some of these options you will have already seen in the Wizard. Others are for advanced use only. The options are all the same as those used when invoking Catalina from a command line – but with Code::Blocks you don't have to remember them all, or what each one is for!.

For example, here are the options listed under the Memory Model, Size and Cache options:



Categories: Memory Model, Size and Cache Options (default is TINY, no cache)

- [✓] TINY (code, data and stack in Hub RAM) [-DTINY]
- [ ] SMALL (code in XMM RAM, data and stack in Hub RAM [-DSMALL]
- [ ] LARGE (code and data in XMM RAM, stack i Hub RAM [-DLARGE]
- [ ] 1K Cache [-DCACHED_1K]
- [ ] 2K Cache [-DCACHED_2K]
- [ ] 4K Cache [-DCACHED_4K]
- [ ] 8K Cache [-DCACHED_8K]
- [ ] Set memory size (specify in MEM_SIZE custom variable) [-M$(MEM_SIZE)]
- [ ] Set read-only base address (specify in RO_BASE custom variable) [-R$(RO_BASE)]
- [ ] Set read-write base address (specify in RW_BASE custom variable) [-P$(RW_BASE)]

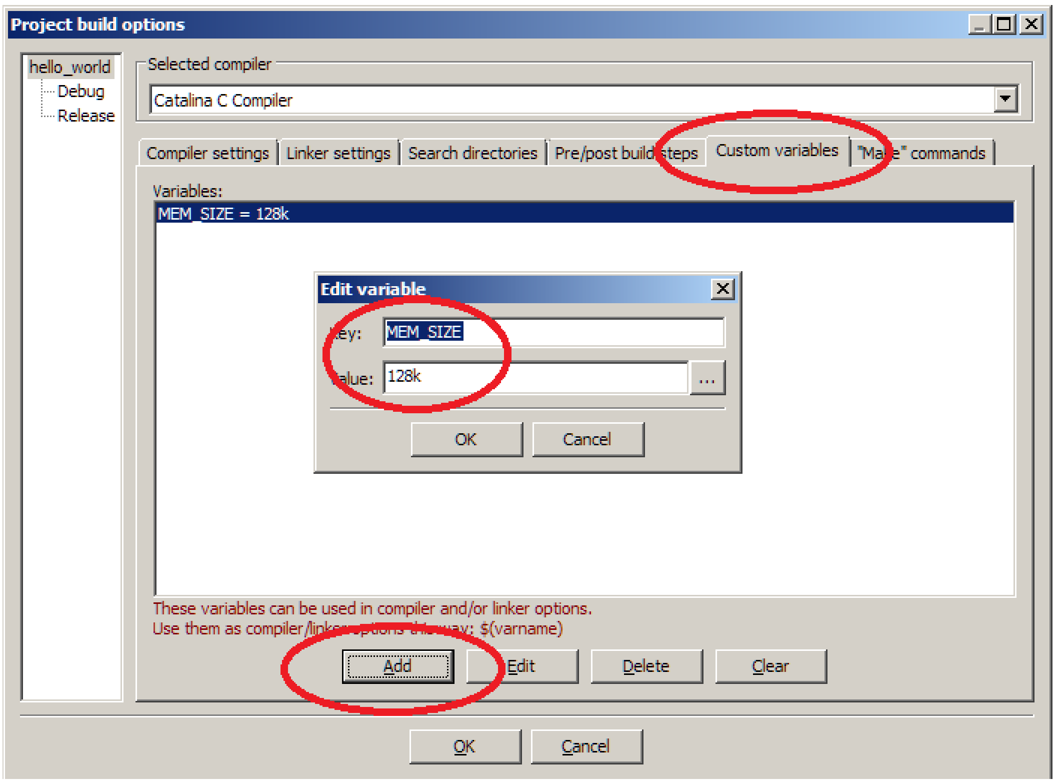
The **TINY** option is shown as selected because this is what we chose in the Wizard. If your Propeller platform has no XMM, this is the only option you will ever need to select from this category – the others are of interest for platforms with XMM RAM.

The Code::Blocks IDE allows you to specify multiple options even if they are incompatible – e.g. here you could select both **TINY** and **LARGE** as your memory model. However, one reason for arranging the options in categories is that it makes it much easier to see if you have selected incompatible combinations. However, it is still possible to select incompatible options at the *Project* and *Build Target* levels, so if the build is not behaving as you expected, it is worth checking that options specified at different levels are not interfering with each other.

For more detail about each option, refer to the **Catalina Reference Manual** (the actual command-line option that corresponds to each selection is shown in square brackets).

In some cases, the option indicates that it requires a named variable which must be specified on the **Custom Variables** tab. An example in the category above would be the **Set Memory Size** option, which expects the actual memory size to be specified in a variable called **MEM_SIZE**. To use this option, you not only need to select it in this dialog box, you then need to go to the **Custom Variables** tab and enter both the appropriate variable and an appropriate value for it.

For example, to use the value **128k** as the **Memory Size**, you would define the **MEM_SIZE** variable on the **Custom Variable** tab as shown below:



Note that if there is no predefined option to do what you need, you can also specify compiler options by specifying them on the **Other Options** tab, or define arbitrary symbols by specifying them on the **#defines** tab (note that these will be normal C symbols, not Catalina symbols).

After changing any Compiler options, it is a good idea to rebuild the project and verify that the options are correctly specified in the command line shown in the build log. To do this, it may be useful to modify the default logging to show the actual command-line used. This is discussed in the next section.

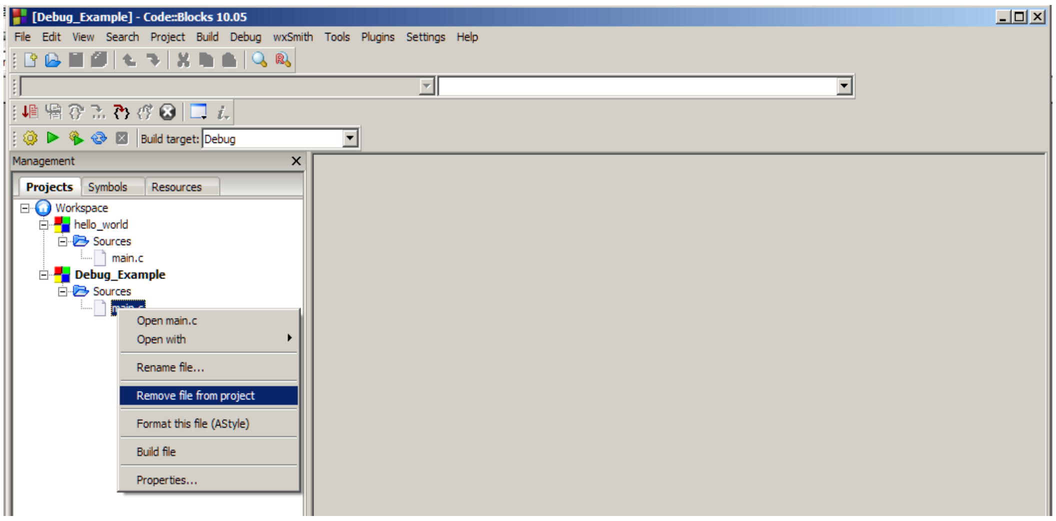
Building a more Complex Catalina Project

Now we are ready to build a more complex project. We will build a project using files that already exist in the Catalina **demos** directory, and also use this project to demonstrate the use of the **blackbox** debugger from within Code::Blocks.

You don't need to close the current project. Simply invoke the **Catalina Project Wizard** again (**File->New->Project...**) and specify **Debug_Example** as the name of the project (don't use spaces in the project name, as this name is also used as the name of the final executable).

Also in the wizard, select your Propeller platform (e.g. **C3**), but this time select the **Integer-Only C Library** (the **Standard C Library** will result in too large an executable on some platforms), and the **TINY** memory model.

When the new project has been built, it will again contain a default *main.c* source file, but we don't want that file – so right click on the file and select **Remove File From Project**:


[Debug_Example] - Code::Blocks 10.05
File Edit View Search Project Build Debug wxSmith Tools Plugins Settings Help
Build target: Debug
Management
Projects Symbols Resources
Workspace
hello_world
Sources
main.c
Debug_Example
Sources
Open main.c
Open with
Rename file...
Remove file from project
Format this file (AStyle)
Build file
Properties...

Then, right click on the project node and select **Add Files ...**

In the file file selection dialog box that appears, navigate to the Catalina debug example in the demo folder – e.g. (in Windows):

```
C:\Program Files\Catalina\demos\debug
```

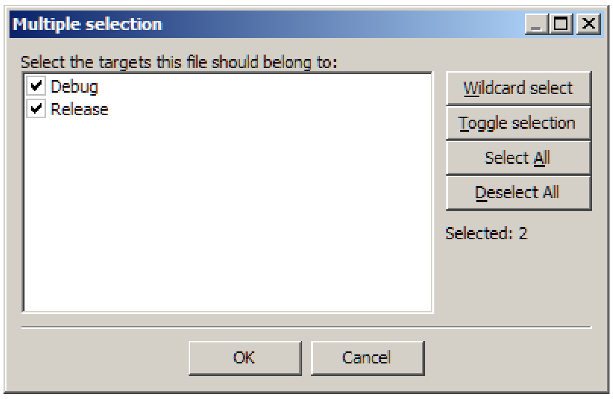
Or (in Linux):

```
/usr/local/lib/catalina/demos/debug
```

Then select the following four files (by holding down SHIFT while selecting each one):

```
debug_test.h
debug_functions_1.c
debug_functions_2.c
debug_main.c
```

When you press **Open**, you will see a dialog box similar to the following appear:



Multiple selection

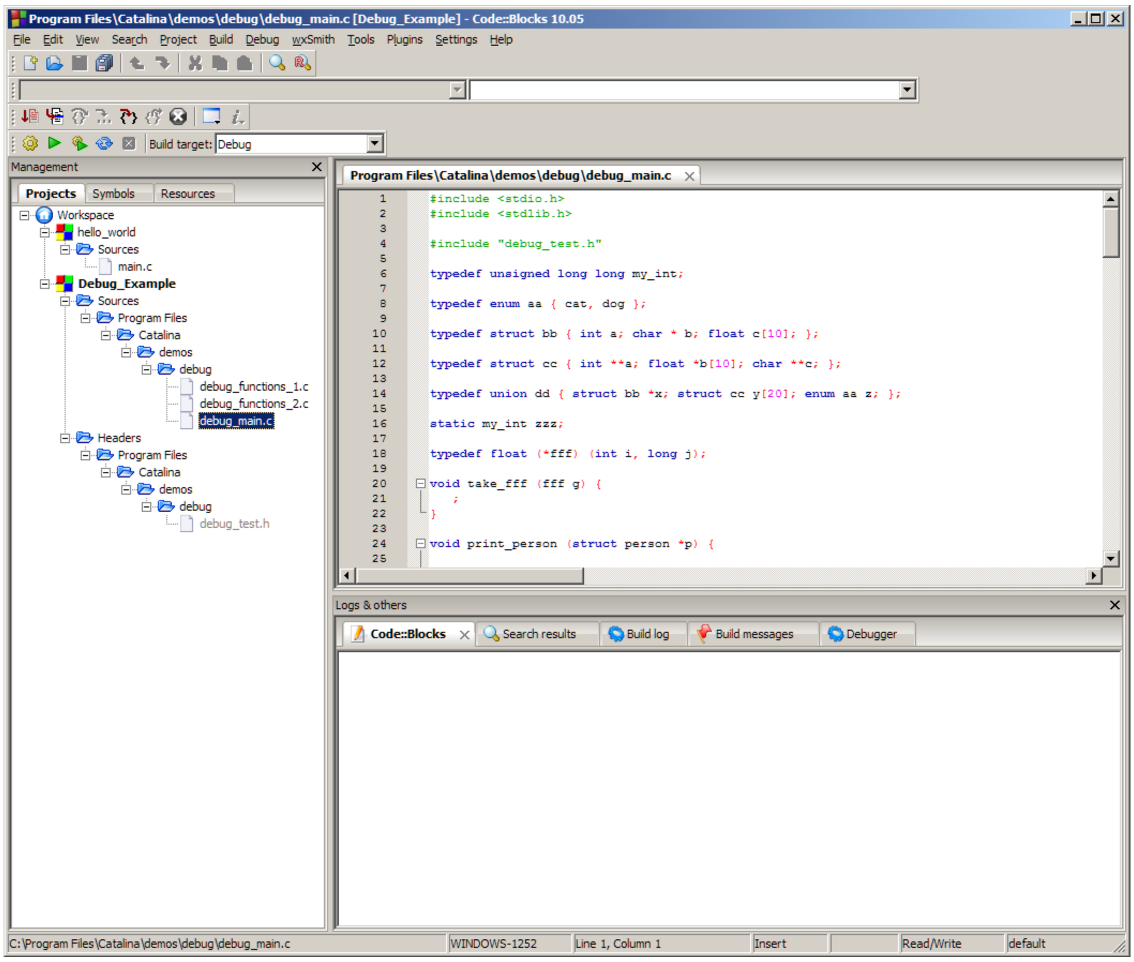
Select the targets this file should belong to:

- ✓ Debug
- ✓ Release

Selected: 2

Since we *do* want to add all these files to both the Release and Debug targets, just press **OK**.

Now when you open the project node in the navigation pane, and keep opening each sub-node, you will end up with something like the following:



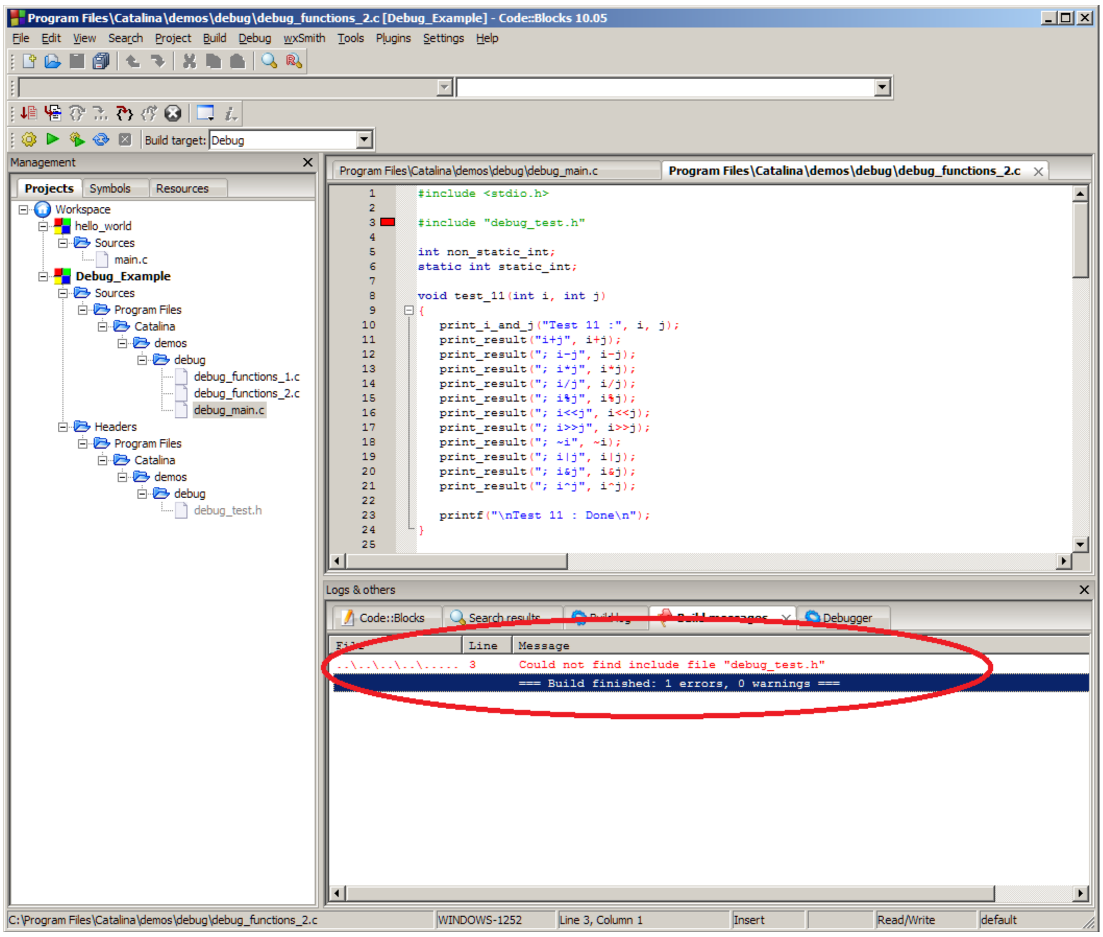
Program Files\Catalina\demos\debug\debug_main.c [Debug_Example] - Code::Blocks 10.05

```
#include <stdio.h>
#include <stdlib.h>

#include "debug_test.h"

typedef unsigned long long my_int;

typedef enum aa { cat, dog };

typedef struct bb { int a; char * b; float c[10]; };

typedef struct cc { int **a; float *b[10]; char **c; };

typedef union dd { struct bb *x; struct cc y[20]; enum aa z; };

static my_int zzz;

typedef float (*fff) (int i, long j);

void take_fff (fff g) {
    ;
}

void print_person (struct person *p) {
```

This is how CodeBlocks displays files that are included from a path outside the Project directory – these files have not been copied to the project directory – they still only exist in the demo folder – the project refers to them by their original path.

Ensure that the **Debug** target is selected, and build the project (e.g. press **Ctrl-F9**).

When you do so, you will receive an error – don't worry, this is expected! We will use this error to demonstrate another place you may need to configure things in Code::Blocks. The error will appear similar to that below:


Program Files\Catalina\demos\debug\debug_functions_2.c [Debug_Example] - Code::Blocks 10.05
File Edit View Search Project Build Debug wxSmith Tools Plugins Settings Help
Build target: Debug
Management
Projects Symbols Resources
Workspace
hello_world
Sources
main.c
Debug_Example
Sources
Program Files
Catalina
demos
debug
debug_functions_1.c
debug_functions_2.c
debug_main.c
Headers
Program Files
Catalina
demos
debug
debug_test.h
Program Files\Catalina\demos\debug\debug_main.c
Program Files\Catalina\demos\debug\debug_functions_2.c
Logs & others
Code::Blocks
Search results
Build log
Build messages
Debugger
C:\Program Files\Catalina\demos\debug\debug_functions_2.c
WINDOWS-1252
Line 3, Column 1
Insert
Read/Write
default

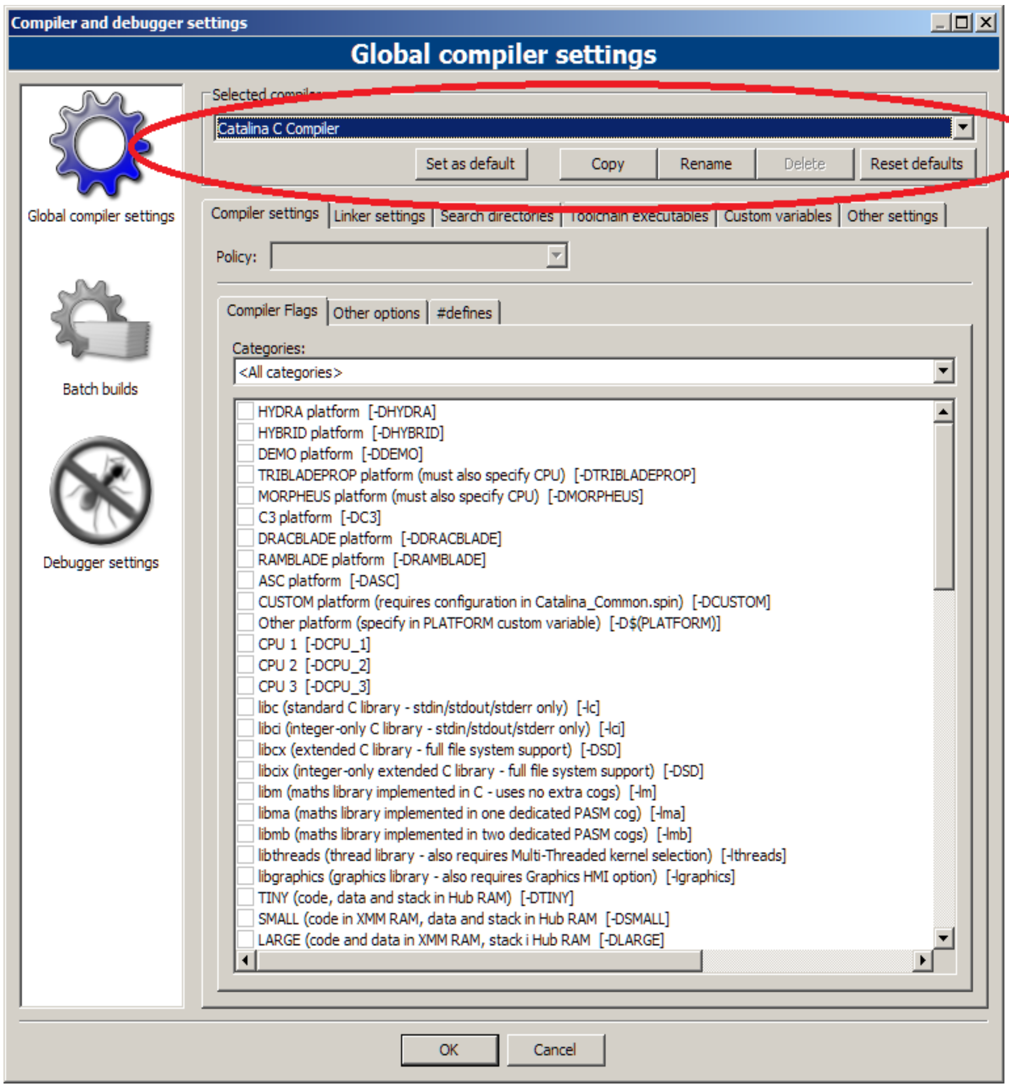
```
#include <stdio.h>

#include "debug_test.h"

int non_static_int;
static int static_int;

void test_11(int i, int j)
{
   print_i_and_j("Test 11 :", i, j);
   print_result("i+j", i+j);
   print_result("; i-j", i-j);
   print_result("; i*j", i*j);
   print_result("; i/j", i/j);
   print_result("; i%j", i%j);
   print_result("; i<<j", i<<j);
   print_result("; i>>j", i>>j);
   print_result("; ~i", ~i);
   print_result("; i|j", i|j);
   print_result("; i&j", i&j);
   print_result("; i^j", i^j);

   printf("\nTest 11 : Done\n");
}
```

```
File               Line   Message
..\..\..\..\.....  3      Could not find include file "debug_test.h"
                          === Build finished: 1 errors, 0 warnings ===
```

This error is because the project contains an *include* file, but (by default) Code::Blocks does not include the appropriate path to *find* the include file - since the files do not not actually exist within the project structure Code::Blocks has built – these are merely links to the files in their original location[2]. This is quite normal behavior for Code::Blocks, and has nothing to do with Catalina.

2 This can be an important thing to remember – if you edit this file in this project, but the file is also used in another project, *any changes you make will appear in that project as well!* If you do not want this behavior, you should instead manually make a copy of the file in the project directory first, and *then* include it in the project.

To rectify this problem, select **Setting->Compiler and Debugger** from the main menu. You will see a dialog box similar to the following:



Selected compiler: Catalina C Compiler

Policy:

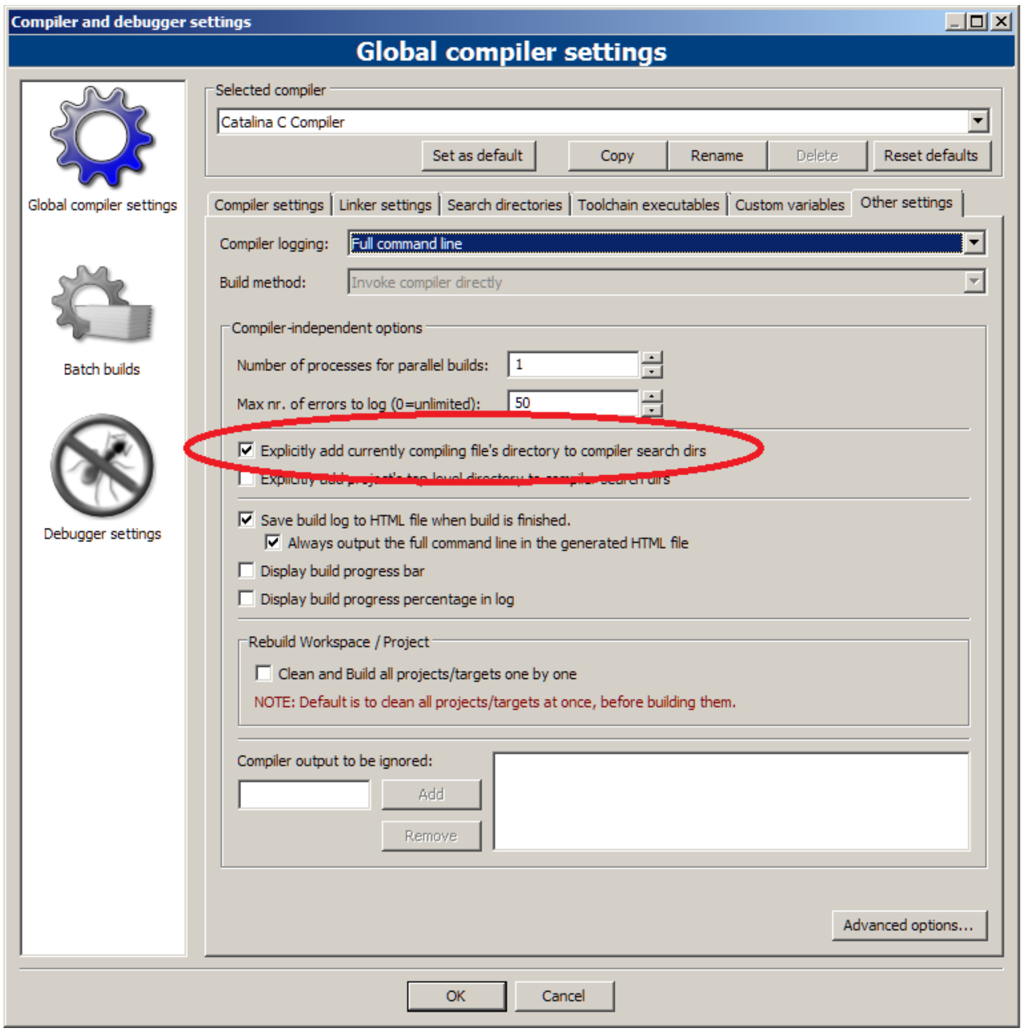
Categories: <All categories>

HYDRA platform [-DHYDRA]
HYBRID platform [-DHYBRID]
DEMO platform [-DDEMO]
TRIBLADEPROP platform (must also specify CPU) [-DTRIBLADEPROP]
MORPHEUS platform (must also specify CPU) [-DMORPHEUS]
C3 platform [-DC3]
DRACBLADE platform [-DDRACBLADE]
RAMBLADE platform [-DRAMBLADE]
ASC platform [-DASC]
CUSTOM platform (requires configuration in Catalina_Common.spin) [-DCUSTOM]
Other platform (specify in PLATFORM custom variable) [-D$(PLATFORM)]
CPU 1 [-DCPU_1]
CPU 2 [-DCPU_2]
CPU 3 [-DCPU_3]
libc (standard C library - stdin/stdout/stderr only) [-lc]
libci (integer-only C library - stdin/stdout/stderr only) [-lci]
libcx (extended C library - full file system support) [-DSD]
libcix (integer-only extended C library - full file system support) [-DSD]
libm (maths library implemented in C - uses no extra cogs) [-lm]
libma (maths library implemented in one dedicated PASM cog) [-lma]
libmb (maths library implemented in two dedicated PASM cogs) [-lmb]
libthreads (thread library - also requires Multi-Threaded kernel selection) [-lthreads]
libgraphics (graphics library - also requires Graphics HMI option) [-lgraphics]
TINY (code, data and stack in Hub RAM) [-DTINY]
SMALL (code in XMM RAM, data and stack in Hub RAM [-DSMALL]
LARGE (code and data in XMM RAM, stack i Hub RAM [-DLARGE]

The first thing to notice is that this dialog box has a set of compiler flags that looks a lot like the Project **Build Options** dialog box – except that this set of options will be used for **ALL** projects built using the selected compiler. We don't want to set any of these options here (although if you only have *one* Propeller platform and use it for *every* project, setting your preferred options here may be simpler than using the Project Wizard each time!).

Instead, just make sure the **Catalina C Compiler** is selected in the top drop down box (and note that - if you have not done so already - you can press the **Set As Default** button here to make your selection the default compiler). Then select the **Other Settings** Tab.

The dialog box will then appear as shown below:



Selected compiler: Catalina C Compiler

Compiler logging: Full command line

Build method: Invoke compiler directly

Compiler-independent options

Number of processes for parallel builds: 1

Max nr. of errors to log (0=unlimited): 50

- [✓] Explicitly add currently compiling file's directory to compiler search dirs
- [ ] Explicitly add project's top-level directory to compiler search dirs
- [✓] Save build log to HTML file when build is finished.
  - [✓] Always output the full command line in the generated HTML file
- [ ] Display build progress bar
- [ ] Display build progress percentage in log

Rebuild Workspace / Project

- [ ] Clean and Build all projects/targets one by one

NOTE: Default is to clean all projects/targets at once, before building them.

Compiler output to be ignored:

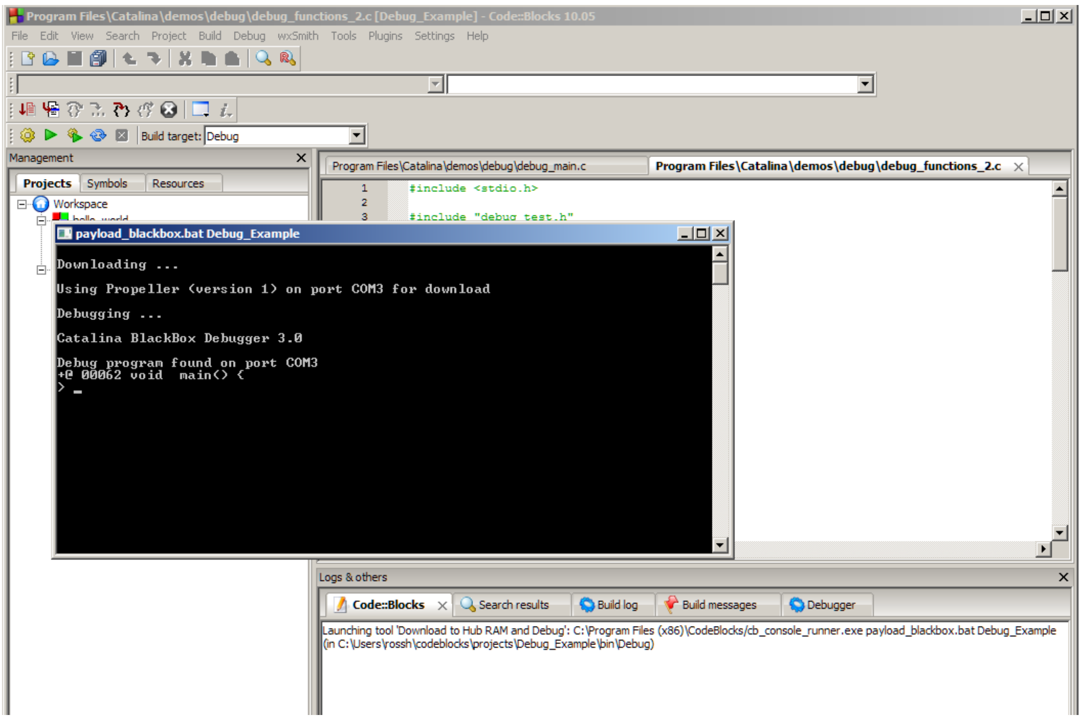
The option we need to select to rectify our problem is **Explicitly add currently compiling File's directory to compiler search dirs** – make sure this option is selected.

However, while we are here, there are a couple of other options it is worth knowing about. You may want to select **Full Command Line** for the **Compiler Logging** option, and also the **Save Build Log to HTML file when compile is finished** option – both of these are useful in diagnosing problems when your selected compiler options don't seem to be doing what you expected.

When you have selected the necessary options, close this dialog box and rebuild the **Debug_Example** project.

This time the project should build without errors (if you get an error message saying **Too Big by xx Longs** error, double check to ensure that you are building the project to use the **Integer-only C Library** rather than the **Standard C Library**).

Next, select **Tools->Download to Hub RAM and Debug** from the main menu. A DOS command window (or a Linux command shell) will open, displayed in front of Code::Blocks, similar to the following:


Program Files\Catalina\demos\debug\debug_functions_2.c [Debug_Example] - Code::Blocks 10.05
File Edit View Search Project Build Debug wxSmith Tools Plugins Settings Help
Build target: Debug
Management
Projects Symbols Resources
Workspace
Program Files\Catalina\demos\debug\debug_main.c
Program Files\Catalina\demos\debug\debug_functions_2.c
payload_blackbox.bat Debug_Example
Logs & others
Code::Blocks Search results Build log Build messages Debugger

```
#include <stdio.h>

#include "debug_test.h"
```

```
Downloading ...

Using Propeller (version 1) on port COM3 for download

Debugging ...

Catalina BlackBox Debugger 3.0

Debug program found on port COM3
+@ 00062 void  main() {
> _
```

```
Launching tool 'Download to Hub RAM and Debug': C:\Program Files (x86)\CodeBlocks/cb_console_runner.exe payload_blackbox.bat Debug_Example
(in C:\Users\rossh\codeblocks\projects\Debug_Example\bin\Debug)
```

This command box displays the progress of the **payload** loader, and is then used to run the **blackbox** debugger (initiated after the program has been downloaded). Once the debugger starts, press **n <ENTER>** a few times to step to the **next** line, and you should see the program output appear on whatever screen is connected to your Propeller. When you have finished debugging, press **e <ENTER>** to **exit** the debugger and return to Code::Blocks. Or at any time you can simply close the command window.

If you would prefer to use a graphical debugger instead, then (on Windows only) you could easily add another **Tool** to invoke the **BlackCat** debugger.

Select **Tools->Configure tools...** to see how to add another **Tool** to Code::Blocks.

Notes about the Catalina Tools

General Note

The Catalina items in the **Tools** menu will not work until you have a Catalina C Project open in Code::Blocks. This is because the path to the executable tools is not specified in the tools themselves – it is only set once you have specified the Compiler to use.

The Build XMM Utilities Tool

One of the items in the **Tools** menu represents a utility developed specifically to simplify the use of Code::Blocks in conjunction with Propeller platforms with XMM RAM. This is the **Build XMM Utilities** menu item.

When you invoke this tool (make sure you have your Catalina XMM project open first!) you will be guided through a set of questions much like the **Catalina Project Wizard**. The purpose of these questions is to build the appropriate XMM utilities necessary to download and debug programs on the XMM platform you are using. Before you use any of the other XMM tools, you will need to invoke this tool (*once)* when you build projects for a Propeller platform with XMM RAM. If you subsequently need to download or debug projects on a *different* XMM platform, you should re-invoke the tool. Also, note that your Catalina Project must use the *same options* as you specify in this tool, or the other XMM RAM tools will not work correctly.

Environment Variables and Code::Blocks

The environment variables used by the Catalina command-line compiler also apply when Catalina is invoked by Code::Blocks. If you set these environment variables at the system or user level[3] then this this may cause unwanted conflicts with the Catalina options set in your Code::Blocks project.

If in doubt, enable logging of the full command line in Code::Blocks, and also add the Verbose (**-v)** option to Catalina to list what options are actually in effect when building your project.

However, if you installed Catalina to a non-standard location, setting at least the **LCCDIR** environment variable at the system level is very useful. It not only simplifies usage of Catalina via the command-line, it also allows Code::Blocks to auto-detect where Catalina is installed.

Also, when Code::Blocks successfully auto-detects Catalina, it will import the current settings of the following two environment variables. They will be configured as the default search directories in the Code::Blocks Compiler configuration. The purpose of this is to try and make projects built in Code::Blocks continue to build correctly even if you subsequently change these variables for using Catalina in command-line mode:

CATALINA_INCLUDE

CATALINA_LIBRARY

[3] For example, in Windows you do this from **System Properties** - e.g. in Windows 7, select **Start->Control Panel->System and Security->Advanced System Settings** (the precise location differs in other versions of Windows). In the resulting dialog box, select the **Advanced** Tab and press **Environment Variables**.

These settings can be manually modified later in the **Search Directories** tab of the **Settings->Compiler and Debugger...** dialog box if required.

Code::Blocks Limitations

Code::Blocks was initially developed for self-hosted development. While it is increasingly being used for cross compilation and embedded development its roots are sometimes still evident.

For example, in the current release (**10.05**) it is *not* possible to configure the *extension* of the final executable program built by Code::Blocks – it always assumes this will be **.exe** (for Windows) or blank (for Linux). Since the convention for Propeller executables is to generally use **.binary** or **.eeprom**, this leads to Code::Blocks believing it has not yet built the final executable, so the **Build** command will *always* redo the final link step, even if the executable is actually up to date.

It is possible to change this behavior, but not from within the Compiler plugin – it has to be changed in the base Code::Blocks program. This means that to fix this limitation in release 10.05, the *entire* Code::Blocks program itself would have to be redistributed. However, this limitation is likely to be addressed in a future release of Code::Blocks.

Also, the Code::Blocks Debugger plugin does not currently support the Catalina debuggers - the **blackbox** and **blackcat** debuggers can be *launched* from Code::Blocks, but will run in separate windows. This may be added in a future release of Catalina, but at present it is probably best to disable or remove the Code::Blocks Debugger plugin altogether (unless you also use it in conjunction with other compilers).

What's Next with Code::Blocks?

Code::Blocks is a very sophisticated IDE, and has many more features than have been discussed here. For more information, consult the Code::Blocks documentation.

Also, there are many additional plugins available for Code::Blocks that are designed to assist in the development of C programs. These plugins will generally work perfectly well with Catalina.