

SimpleIDE User Guide



Table of Contents

What's New?	2		
SimpleIDE v0-9-40 to v0-9-43	2		
SimpleIDE v0-9-26 to v0-9-40	2		
Overview	4		
Features	5		
Software Requirements and Installation	5		
Downloads and Installation Instructions	5		
USB Drivers	5		
First Run	6		
Start-up "About" Window.....	6		
Hint: Simple View vs. Project View	6		
SimpleIDE Properties	7		
Initial View and First Program.....	8		
Workspace and IDE Controls	9		
File Menu	10		
Project Menu	10		
Edit Menu	11		
Tools Menu	12		
Program Menu	15		
Serial Dropdown Menu (Button Bar)	16		
Help Menu	16		
Button Bar	16		
Status Bar	17		
Build Status	18		
Project Manager	19		
File Manager Tab	20		
Project File Types	21		
Project Options	21		
		Board Type.....	22
		Compiler Type.....	22
		Memory Model.....	22
		Compiler Options	23
		Linker Options.....	24
		SimpleIDE Terminal	25
		Features	25
		Options.....	25
		Simple Libraries	26
		How to add a Simple Library to a Project	26
		Alternative	27
		How to Create a Simple Library	29
		Simple Library Directory Structure	30
		Recommended Features.....	30
		Board Types	31
		Special Clock Boards	31
		Basic Board Types	32
		EEPROM Board Types	32
		External Flash Board Types	33
		External RAM Board Types	33
		SDLOAD Board Types	33
		SDXMMC Board Types	33
		SimpleIDE SDLOAD and SDXMMC Attributes	33
		Configuration Files	34
		Configuration Variable Patching	34
		Future Improvements	35
		Support	35
		Revision History	35

What's New?

If you have used a previous version of SimpleIDE, this section explains what differences to expect with the new version of the software.

SimpleIDE v0-9-40 v0-9-43

- Fixed bug causing SimpleIDE to crash upon targeting a Simple Library with Zip Project.
- Fixed bug causing newly-created Simple Library to be unrecognized by other projects until restarting the SimpleIDE session.

SimpleIDE v0-9-26 v0-9-40

- Enhanced Zip Project feature to include any Simple Libraries used by the project. Also it now stores all libraries in a subfolder within the archive and adjusts project settings to match.
- Added "Auto Include Simple Libraries" option (enabled by default) to simplify Propeller C projects that use them. Now, only an appropriate "#include" statement need be added to code to use a Simple Library in the project; no -I and -L Project Manager settings necessary.
- Compile process for projects including one or more Simple Libraries is now much faster.
- Added "Enable Pruning" setting to have the compiler and linker remove unused code from the program image. See Compiler Options on page 23 for more information.
- "32-bit doubles" option is enabled by default since 64-bit doubles are impractical for most applications.
- Installer automatically removes previous PropGCC path entry that prevented clean roll-backs to previous versions of SimpleIDE.
- Installer and application graphics updated.
- Fixed bugs causing SimpleIDE to crash upon deleting items from Project Manager.
- Many other small and behind-the-scenes enhancements.
- Includes PropGCC v1.0.0.2054 which updates the following since PropGCC v0.3.5.1998:
 - Added ability to "garbage collect" unused functions to remove functions that are never called and variables that are never referenced, even if they are in files that do contain useful functions. Depending on the project this may save run-time space. To use this feature, add:
`-ffunction-sections -fdata-sections -Wl,--gc-sections`
to the compiler and linker switches.
 - Added the ability to declare a function with `__attribute__((fcache))`. This will cause the compiler to attempt to place the entire function into the FCACHE area when it is called. This will only work if the function is small enough (less than 1K) and a leaf function (does not call any other functions). A function declared this way will have more predictable timing, since it will be loaded into the FCACHE area in COG memory before it is run. This feature is useful for device drivers and other timing sensitive functions.
 - Added support for non-blocking reads. Terminal style drivers now support the ability to attempt to read byte and return immediately if none is available. This is accomplished by setting the `_IONONBLOCK` bit in the `_flag` field of the `FILE` structure. For example, to check to see if a byte is available from stdin, you could do:

```
stdin->_flag |= _IONONBLOCK;
c = getchar();
```

If a character is available then "c" will be set to that character, otherwise it will be set to -1. Note that this is the same as EOF; in non-blocking mode there is no way to distinguish between end of file and no character currently available (for terminals there never is really an end of file).
 - Functions may now be declared with `__attribute__((native))` to make them be compiled into COG memory. Such functions must be extremely small, and are not available in CMM mode (there isn't any COG space left). There is no guarantee of how much space is

available in any kernel mode, so this feature should be used very sparingly. Use of `__attribute__((fcache))` instead is generally better.

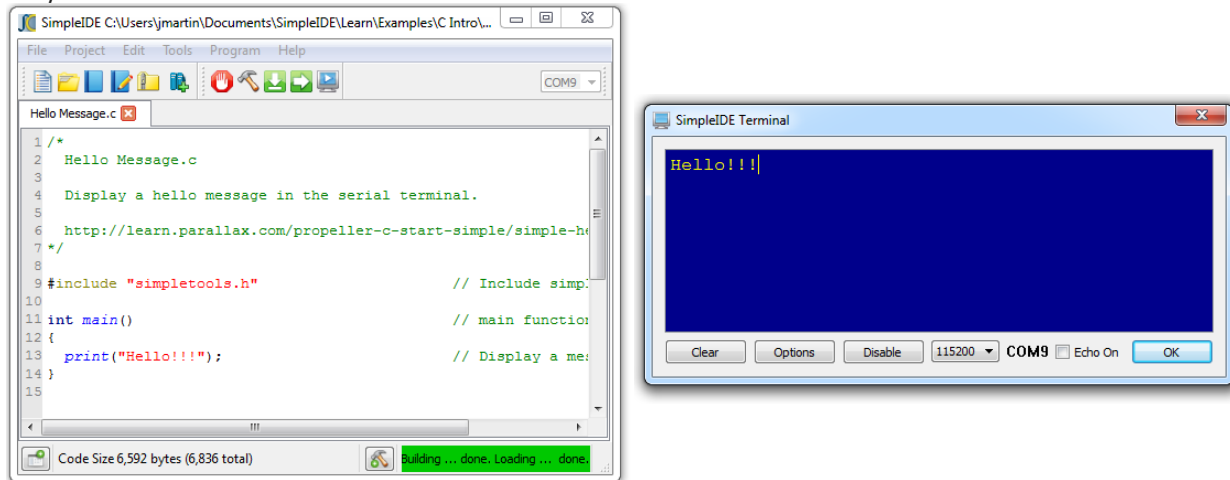
- Added a new preprocessor define, `__PROPELLER_USE_XMM__`, which will be set in all XMM related modes (both `-mxmm` and `-mxmmc`).
- Implemented basic OpenMP support. OpenMP is an optional way to have the compiler parallelize code, using `#pragmas`, and is supported by a number of compilers (including gcc and Visual C++). The Propeller implementation of OpenMP is still experimental, and may have bugs. To enable OpenMP, pass `-fopenmp` to the compiler. Without this option any OpenMP `#pragma` directives are ignored. With it, the compiler will attempt to parallelize code and run it on as many COGs as are available (not in use) at run time. This only works in LMM and CMM modes at present.
- Various library bug fixes/improvements:
 - Added `pthread_getspecific()` and `pthread_setspecific()` macros.
 - Added stub implementations of pthread key and cond variables.
 - Fixed the 32-bit-double version of the math `frexp` function.
 - Added the `sbrk()` function for porting some Unix programs.
 - Added the `strdup()` function to duplicate strings.
 - Added support for '+' in `fopen()` mode string.
 - Made the `remove()` function work correctly on SD cards.
 - Fixed a bug which caused some large floating point numbers to be printed incorrectly by `printf`.
 - Some of the header files were not set up for C++ compatibility; now they should work for both C and C++
- Added a propeller-load configuration file for the Propeller Activity Board. This will allow XMM programs to be run from SD cards on this board.
- Added a propeller-load configuration file and cache driver for the RamPage2 board.
- Fixed the linux version of propeller-load to restore terminal settings after a control-C.
- Updated the GNU Assembler documentation to reflect propeller specific features.
- Improved I2C driver so that it works in XMM modes.

Overview

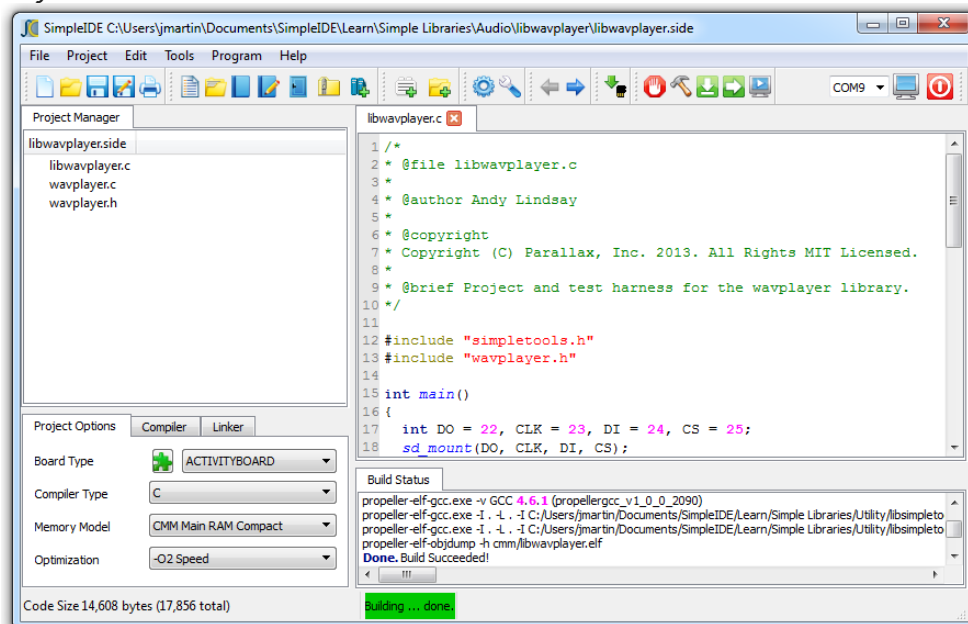
SimpleIDE is a multi-platform, open source, internationalized code development environment for the multicore Propeller microcontroller. SimpleIDE supports the C, C++, Spin and Propeller Assembly (PASM) programming languages. It comes packaged with the PropGCC compiler for C and C++ and the BSTC (Brad's Spin Tool Compiler) for Spin and Propeller Assembly (PASM).

SimpleIDE has a Workspace with two View options: Simple, and Project. It also has an integrated serial terminal that can be launched into a separate window for exchanging information between user and Propeller microcontroller.

Simple View



Project View



Features

- Two development workspace options:
 - Simple View for introductory levels in the Parallax Propeller C Education Program
 - Project View allows file-level operations for custom library and project setups
- Menu Bar with File, Project, Edit, Tools, Program, and Help menus
- Button Bar with frequently used Menu Bar operations and COM port selector
- Streamlined New/Open/Save/Save As Project dialogs
- One-step Zip Project enables sharing with other users and the community
 - Unzipped projects are portable and may be copied to other drives or computers
- Optional Project Manager pane with right-click menu for project file settings, and Project Options, Compiler, and Linker settings tabs
 - Project Options tab enables selection of Board Type, Compiler Type, Propeller GCC Memory Model and Optimization level
- Optional Build Status pane for viewing compiler messages and build progress
- Tabbed text editor with configurable syntax highlighting
- User-selectable font size and family
- Browse Code feature navigates to declarations
- Status bar shows compile size, summary messages, and progress bar
- Available for Linux, Mac OSX, and Windows
- Multiple language interface and code support in comments and strings

Software Requirements and Installation

- Windows XP/7/8, 32/64-bit (install package)
- Mac OS X 10.6-10.8, 64-bit (install package)
- Linux 32/64-bit (compile source from Open Source <https://code.google.com/p/propside/>)
- 600 MB available drive space
- Available USB 2.0 or 3.0 port for programming
- Internet access for obtaining the SimpleIDE software and web tutorials

Downloads and Installation Instructions

If you wish to try Parallax Propeller C tutorials, step-by-step installation instructions can be found here: <http://learn.parallax.com/propeller-c-set-simpleide>

USB Drivers

USB drivers are not included in the installer package and must be installed before connecting Propeller hardware. Download them free:

- Windows: <http://www.parallax.com/usbdriers>
- Mac and Linux: <http://www.ftdichip.com/Drivers/VCP.htm>

First Run

Start-up "About" Window

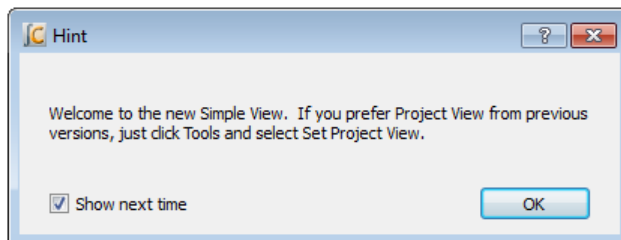
At first startup, the SimpleIDE "About" window appears. This window shows the software version (which may be later than shown here) and provides links to official SimpleIDE resources.

If the "Show this window at startup" box is checked, the window will appear on every startup. Clear the checkbox to disable showing this window at every startup.



Hint: Simple View vs. Project View

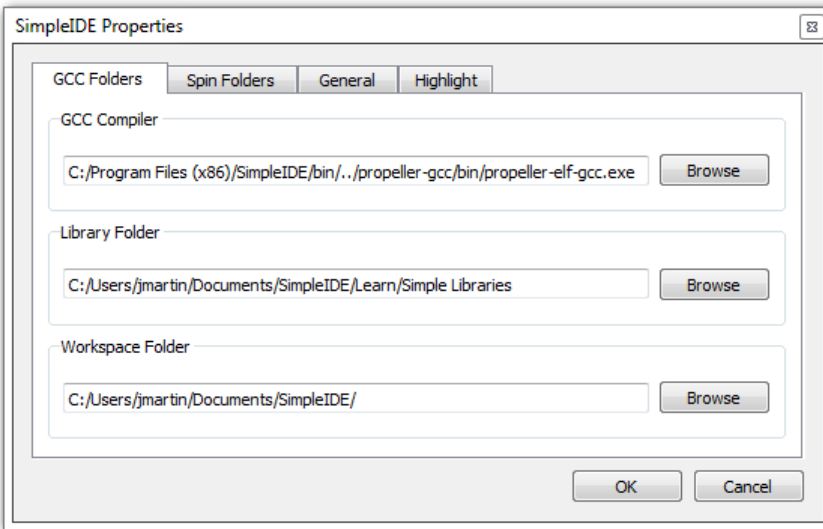
Next, a hint dialog explains that the software is using the Simple View configuration, which is the default upon installation. The Project View configuration is available under Tools > Set Project View. Clear the "Show next time" checkbox to prevent displaying this message at startup.



SimpleIDE Properties

At first startup, the SimpleIDE Properties dialog window appears. It is preconfigured with the recommended settings.

Make a note of the Workspace Folder location. This folder has subfolders for user projects, tutorial examples and demos, and the Propeller C Simple Libraries. These settings can be changed or updated later from the Tools > Properties (F6) menu.



In most cases, the GCC Folders tab will already have the fields properly set. The back-slash '\' path separators commonly used in Windows are replaced by forward-slashes '/' in the IDE.

In the unlikely event that the window reopens after clicking OK, it means that critical GCC/Spin Compiler or Loader Folder information is not correctly set. In this case, use the Browse buttons next to each field to correct its entry.

- GCC Folders tab: GCC Compiler field should contain path to propeller-elf-gcc.exe.
- Spin Folders tab: Spin Compiler field should contain path to bstc.exe.
- General tab: Loader Folder field should contain the path to the propeller-load subfolder of PropGCC (propeller-gcc).

See the Tools Menu section starting on page 12 for examples of default paths in Windows.

Initial View and First Program

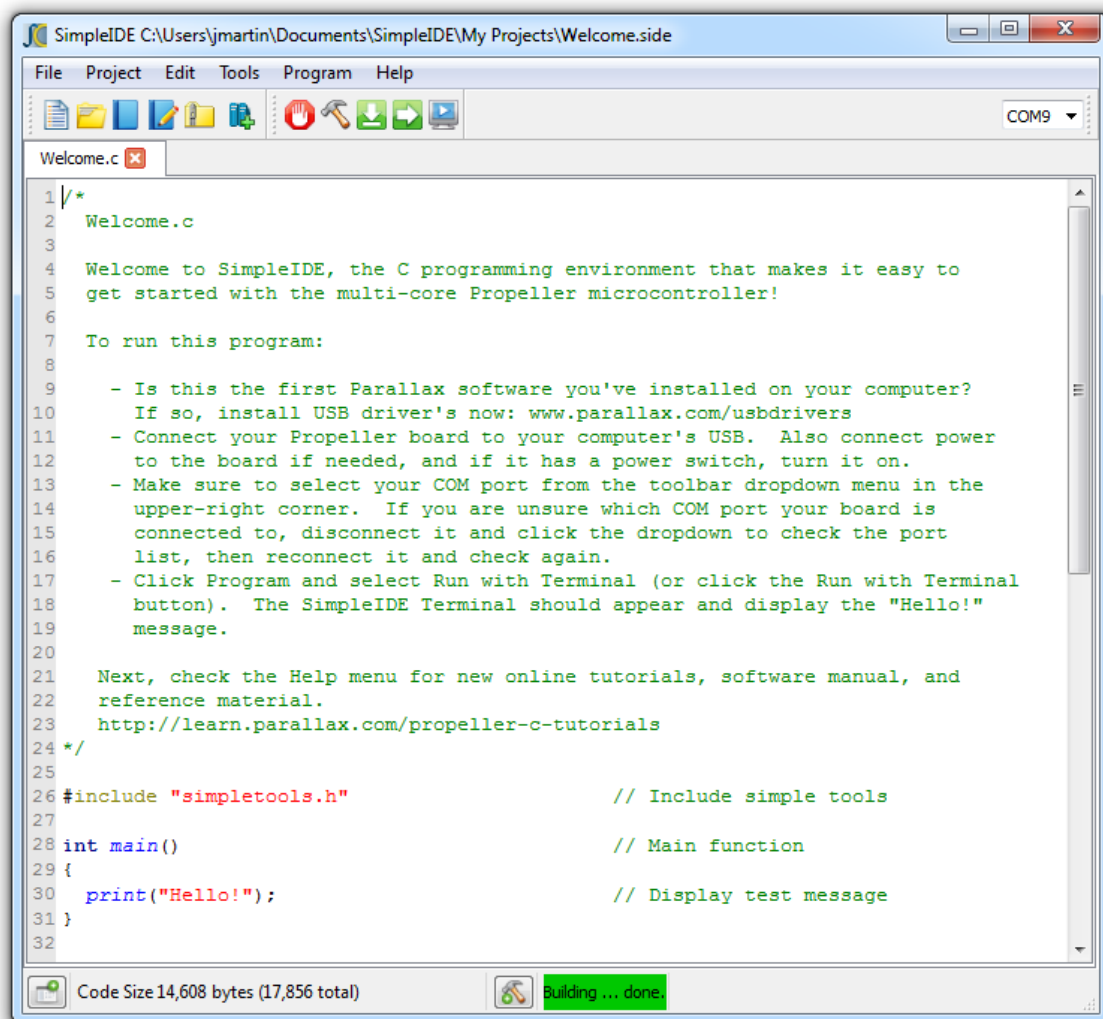
The first time SimpleIDE runs, it will open a Welcome project in Simple View. If you prefer Project View, just click the Tools menu and select Set Project View. View settings persist – the software will start up next time using the last view selected before closing the program.

TIP: If you want to periodically view the Project Manager or Build Status, just click the Show/Hide Project Manager and Show/Hide Build Status buttons in the status bar at the bottom of the IDE window.

For more info about how to set up your board and run this test application, go to:

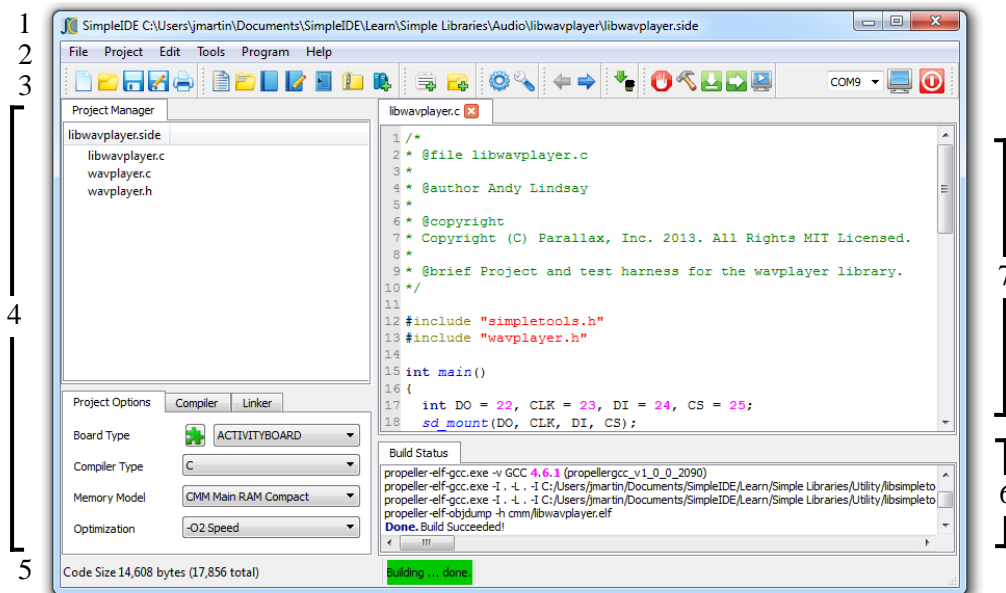
<http://learn.parallax.com/propeller-c-set-simpleide>

Then, follow the link for your operating system for instructions.



Workspace and IDE Controls

SimpleIDE's workspace consists of seven major elements shown here and listed below.



1. Title Bar: Shows project path and filename.
2. Menu Bar: File, Project, Edit, Tools, Program, and Help menu options.
3. Button Bar: Buttons for most commonly used menu items, and COM port selection dropdown.
4. Project Manager: File List and Project Options tabs.
5. Status Bar: Displays code size, build information, and download progress. Buttons for Show/Hide Project Manager and Show/Hide Build Status are also available in Simple View.
6. Build Status: Displays GCC compiler, Linker and download messages.
7. Editor: Text editor for C/C++/Spin code; supports multiple tabs in a single project.

Note that each instance of the software contains one active project that may have one or more tabs. If you want to work on more than one project at a time, run a second instance of SimpleIDE. In Windows you can open another instance by double-clicking a shortcut to SimpleIDE (Desktop, Start -> All Programs), or you can simply double-click the .side project file you want to work with, and it will open into a new SimpleIDE window.

File Menu

IMPORTANT: The first five controls here are only available in Project View, not in Simple View. Simple View only relies on the Project Menu versions of these same controls. This helps prevent confusion in beginner-level C language classes and tutorials, where all these operations have to be performed at the Project level, not the File level.



New [project view only]: Creates a new file called "untitled" in the tabbed editor space.



Open [project view only]: Opens an existing file into a new editor tab.



Save [project view only]: Saves the text of the active editor tab to the file shown on the tab.



Save As [project view only]: Saves the active editor tab's text to a filename of your choice.



Close [project view only]: Closes the currently visible tabbed editor.



Close All: Closes all files and projects.



Print: Prints the current document to a selected printing device.

(Previous file names): Lists the last 5 opened files.



Exit: Asks to save any unsaved files and exits the program.

Project Menu



New Project: Opens a dialog to select folder, project name, and project type (C, C++ or Spin).



Open Project: Opens an existing project file (this with a .side extension).



Save Project: Saves all of the source files in the project with their current names and locations. Note that the project is also saved automatically whenever it is compiled.



Save Project As: Save the current project with a new name to a specified folder. This will save all project settings and relative paths.



Zip Project: Creates a zipped archive of the project that includes all the source, header, and archive files the project needs to compile. After the file is unzipped, the resulting folder contains a portable version of the project that can be run on other computers that have SimpleIDE installed without being dependent on libraries that the other computer might or might not have.



Add File Copy [Project View Only]: Adds a copy of an existing file to the current project.



Add File Link [Project View Only]: Adds a link to an existing file to the current project settings.



Add Include Path [Project View Only]: Adds a path to the current project settings pointing to a folder containing header files the project will use.



Add Library Path [Project View Only]: Adds a path to the current project settings pointing to a folder containing memory model folders, which in turn contain archived library (.a) files. See the File Manager Tab on page 20 for more info. This feature is not for including source libraries (.c/.cpp/.cogc); use Add File Copy or Add File Link for those.



Close Project: Closes the project. Projects are automatically saved before they are closed.



Set Project: [Project View Only]: The Set Project button (F4 or Project->Set Project) makes a project use the currently visible user program. For C language projects, this needs to be the main file. For Spin language projects, if you open a Spin file that is the top object file in the project, make sure to select Set Project while the top file is the active tab before compiling.



Add Simple Library: Simplifies adding a preconfigured library to a project. This opens a dialog for selecting a folder that starts with the letters "lib." Browse subfolders of "...SimpleIDE\Learn\Simple Libraries\" for examples.

When a Simple Library folder is selected using this feature, SimpleIDE automatically adds an **#include** statement into the code, adds include and library paths to the Project Manager's file list, and adds -lname (where name is the characters following "lib" in the folder name) to the Linker tab's Other Linker Options field in the Project Manager. See How to add a Simple Library to a Project on 26 for more information.



Add Tab to Project: Creates a new file and adds it as a file link in the Project Manager. Make sure to choose the file type from the Files of Type dropdown menu. File type options are: .c, .cpp, .spin, .h, .cogc, .ecogc, .espin, and any file (*).



Open Tab to Project: Opens an existing file and adds it to the project. This supports the same file types as Add Tab to Project.

(**Previous projects list**): Lists the last 5 opened projects.

Edit Menu




Copy: Copies selected editor text to the clipboard.

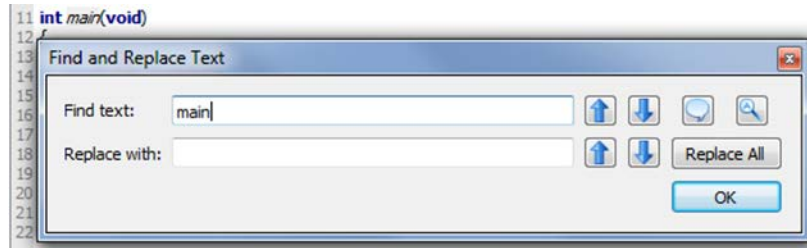


Cut: Copies selected editor text to the clipboard and deletes the selected text.




Paste: Pastes text from the clipboard to the editor at the current cursor position.


 **Find and Replace:** Opens a dialog window that allows searching and replacing text in the editor.





Find text: When a word is entered in this field, the tool will try to find it in the editor. To find more instances of the word, click the Find Previous or Find Next buttons to the right of this field.

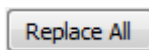
Replace with: When a word is entered in this field, it will be used to replace the word found from the find operation when either the Replace and Find Previous or Replace and Find Next button to the right of this field is clicked.


 **Find Previous or Replace and Find Previous:** Finds the previous word, or replaces the current word then finds the previous word.

 **Find Next or Replace and Find Previous:** Finds the next word, or replaces the current word then finds the next word.

 **Whole Word:** Finds only whole words rather than portions of words that match the Find text field's content.



 **Case Sensitive:** Finds only words that match the case of the Find text field's contents.


 **Replace All:** Replaces all instances of the word in the Find text field with the word in the Replace with field.


 **Redo:** Reverses the last undo operation.

 **Undo:** Undoes the last edit.

Tools Menu

  **Set Simple View/Set Project View:** Toggles between Project View and Simple View workspace options.

 **Go Back:** Make the cursor jump back to the line with the function call or variable name that was just browsed, see Browse Declaration below.

 **Browse Declaration:** If cursor is on a reference to a user-defined function or variable, this button will jump to the declaration of that function or variable. Library functions such as

`printf` that are not in the project file-list (because they are built into PropGCC) cannot be browsed.



Next Tab: Has the same effect as clicking the tab to the right of the one that is currently active in the editor pane.



Font: Allows selecting of an editor font.



Bigger Font: Increases the editor font by 20%.



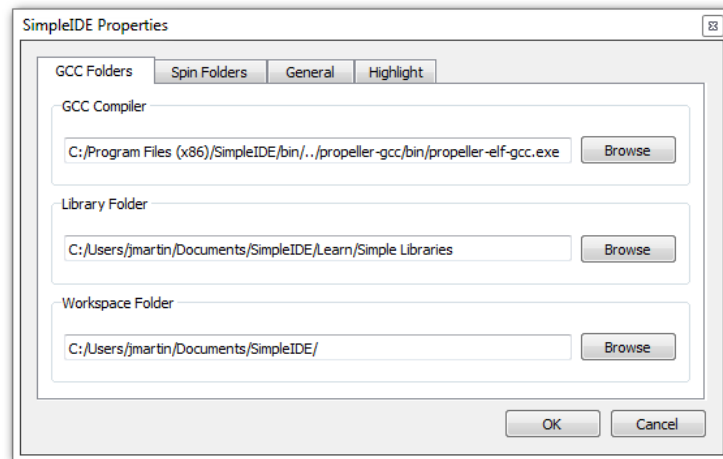
Smaller Font: Decreases the editor font by 20%.

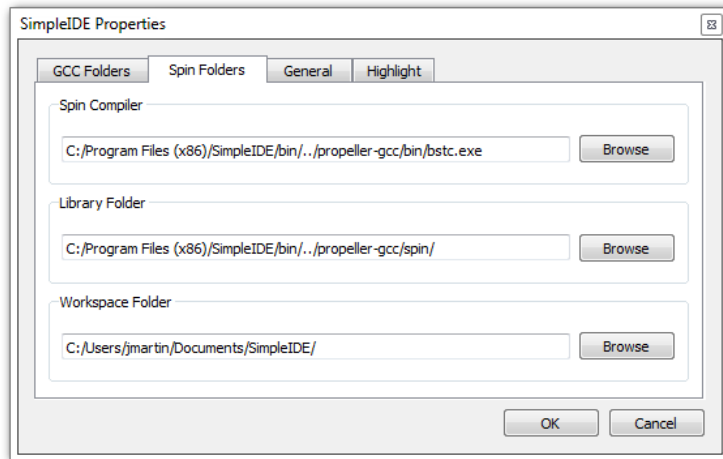


Properties: Open SimpleIDE Properties (F6).

The SimpleIDE Properties dialog window allows changing key Folders, General properties, and Highlights. The controls are consistent over all operating systems and may be changed while the IDE is running to allow using another compiler directory.

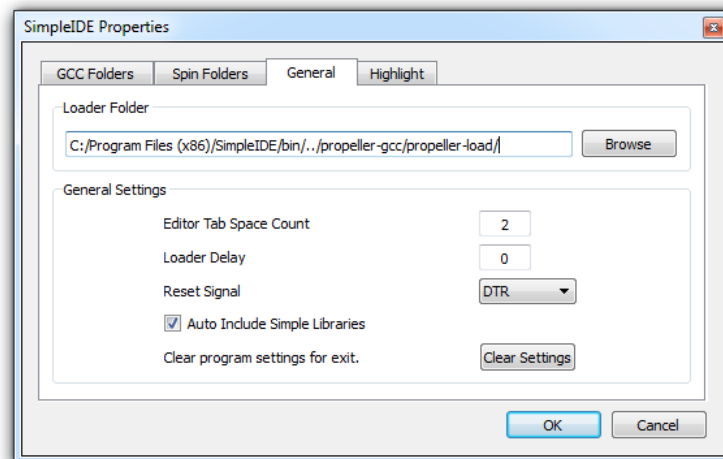
GCC Folders and **Spin Folders:** With a default installation, the fields in the Folders tabs will be set similar to what is shown below. If the window reopens after clicking OK, it means the critical "Compiler" or "Loader Folder" information is not correctly set; use the Browse button to correct the entry. See the SimpleIDE Properties, page 7 for more info.



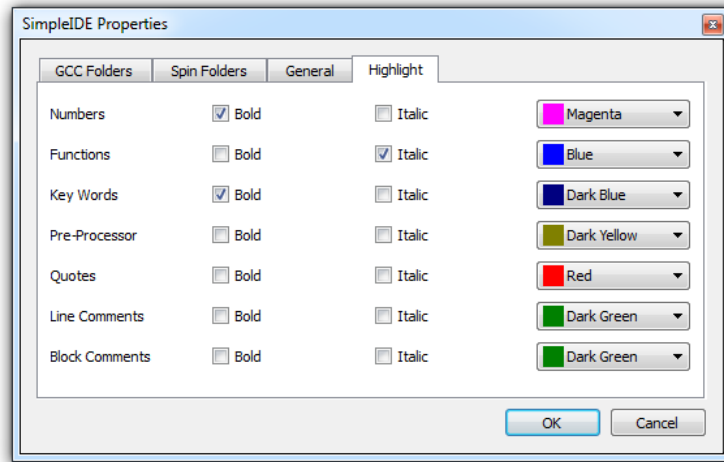


General Tab: Some General property details may need to be changed for different boards.

- **Loader folder:** Board type configuration files are located in the Loader folder. The contents of this folder affect the Project Options Board Type field in the Project Manager.
- **Editor Tab Space Count:** Set the number of spaces to be inserted when the Tab key is pressed. The editor will convert "tabs" to spaces. If existing source code has hard tabs in it, SimpleIDE will not touch the tabs unless the user changes them.
- **Reset Signal:** Options are DTR (default), RTS, and CFG. Some USB serial devices do not have DTR for controlling Propeller reset and should use RTS instead. The CFG option chooses the reset signal type specified in the board's configuration file.
- **Auto Include Simple Libraries:** (default) Enables automatic inclusion of Simple Libraries in projects that use an "#include <...>" statement that references them; without requiring additional -I and -L settings in Project Manager.
- **Clear program settings for exit:** Clears the General tab settings to restore defaults upon the next startup.



Highlight Tab: Editor syntax color and bold/italic settings can be changed here. Only a select set of system colors are available.



Program Menu



Run with Terminal (F8): Build, load, and run program on Propeller RAM (or external flash for XMM). It automatically opens a serial port console terminal window.



Build Project (F9): Build program only. Build status information (such as success, or compiler error information) will appear in the Build Status pane.



Load RAM & Run (F10): Build, load, and run program on Propeller RAM (or external flash for XMM). Development boards that include SD cards (board types with SDLOAD or SDXMMC in their name) will have the program saved to the SD card first.



Load EEPROM & Run (F11): Build and load program into Propeller EEPROM (and external flash for XMM), then run. Development boards that include SD cards (board types with SDLOAD or SDXMMC in their name) will have the program saved to the SD card first.



Stop Build or Loader: Stop the current project build or Propeller loader operations.



File to SD Card: If the development board includes an SD card (its board type has SDLOAD or SDXMMC in its name), this button can be used to send any file to the SD Card.



SimpleIDE Terminal: Toggle the display of the SimpleIDE Terminal. Press to show and connect the terminal window to the selected port. If pressed (dark square appears around the button), the terminal is visible and connected and can be hidden and disconnected by pressing the button again.



Reset Port: This red "power switch" button allows resetting of the board, which causes the Propeller to grab the most recent program image from its EEPROM and start running it.

Serial Dropdown Menu (Button Bar)




The serial dropdown menu is the field beginning with "COM..." at the far right of the button bar. When clicked, it automatically rescans the ports to provide a current list of known ports.

Some computers have serial ports based on Bluetooth. Bluetooth serial port programming is possible, but not recommended for new users. The serial port type can be verified by hovering the mouse cursor over the port name without clicking; the port name and type will appear in the tool tip that appears next to the mouse cursor. Make sure the port type shows "USB Serial ..." or "FT232 ..." and select one of them.

SimpleIDE does not automatically detect and use the port that the Propeller is connected to. The user must specify the port manually via the serial dropdown menu.


Help Menu


This menu provides links to the documentation for SimpleIDE, Propeller C, and Propeller GCC.


 **SimpleIDE Manual** (PDF): Opens this document.

 **Propeller C Tutorials** (Online): Opens <http://learn.parallax.com/propeller-c-tutorials>.

 **Simple Library Reference**: Opens the Simple Library documentation.

 **PropGCC Reference** (Online): Opens <http://www.parallax.com/propellergcc>.

 **About**: Shows the about window, SimpleIDE version number, startup disable checkbox, SimpleIDE online link, and the SimpleIDE support email.

 **Credits**: Shows links to third-party information and translation credits. License texts are distributed with the SimpleIDE package. Translations are greatly appreciated.

Button Bar

The button bar has buttons for the most frequently used menu items.

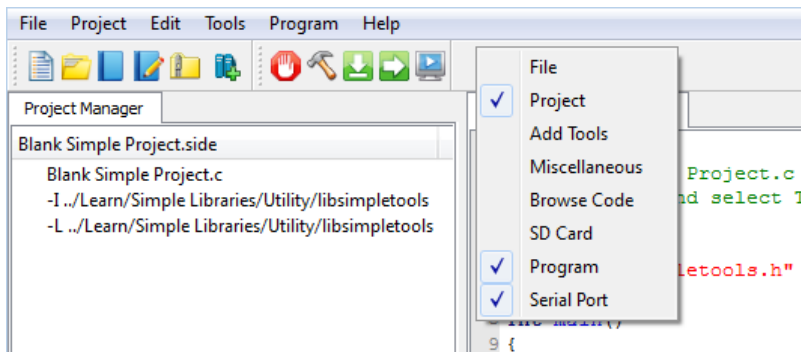
Project View



Simple View



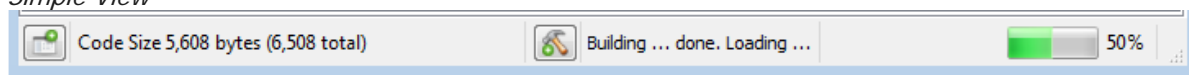
To add additional buttons to Simple View's button bar, simply right-click the button bar and select a button group to add. The change will last until the window is closed.



Status Bar

The Status Bar shows status information like code size, program build, and load progress.

Simple View



While in Simple View, two additional buttons appear (shown above): Show/Hide Project Manager and Show/Hide Build status. Those two buttons are not visible in Project View.



Show/Hide Project Manager (Simple View Only): If Project Manager is not visible, this button causes it to expand to the left of the Editor pane. If it is visible, the button causes it to collapse.



Show/Hide Build Status (Simple View Only): If Build Status is not visible, this button causes it to expand below the Editor pane. If it is visible, the button causes it to collapse.

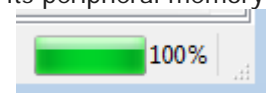
Code Size: This lists the program image size, followed by program image + RAM in parentheses.

Code Size 5,608 bytes (6,508 total)

Build Status: Summary messages display whether a build and/or download succeeded or failed. This feature also displays certain steps such as "Building..." and "Loading...". A green background indicates a successfully completed operation; a red background reports failure. In Simple View, failure also causes the Build Status pane to automatically open, which displays detailed information about the failure.

Building ... done. Loading ... done.

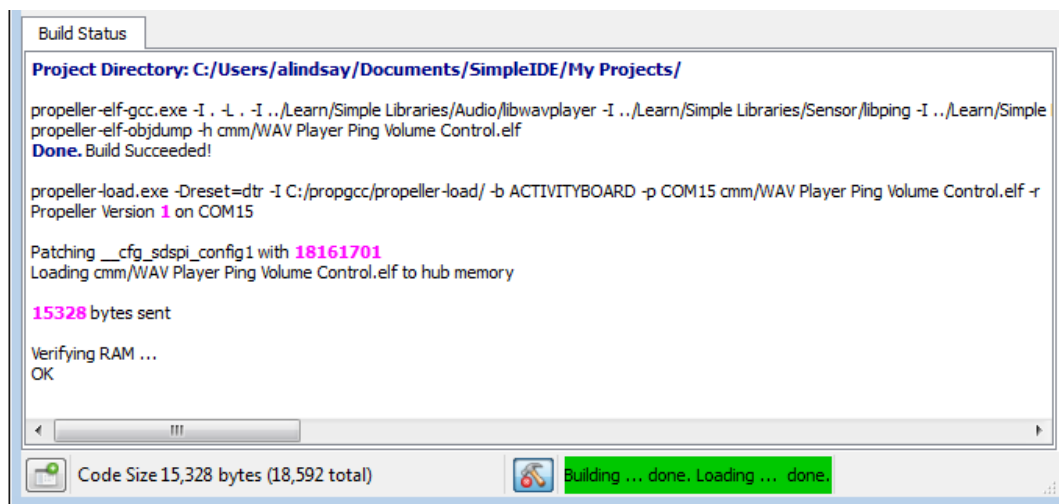
Download progress: Displays percent completed of code/data being transferred to Propeller chip and its peripheral memory.



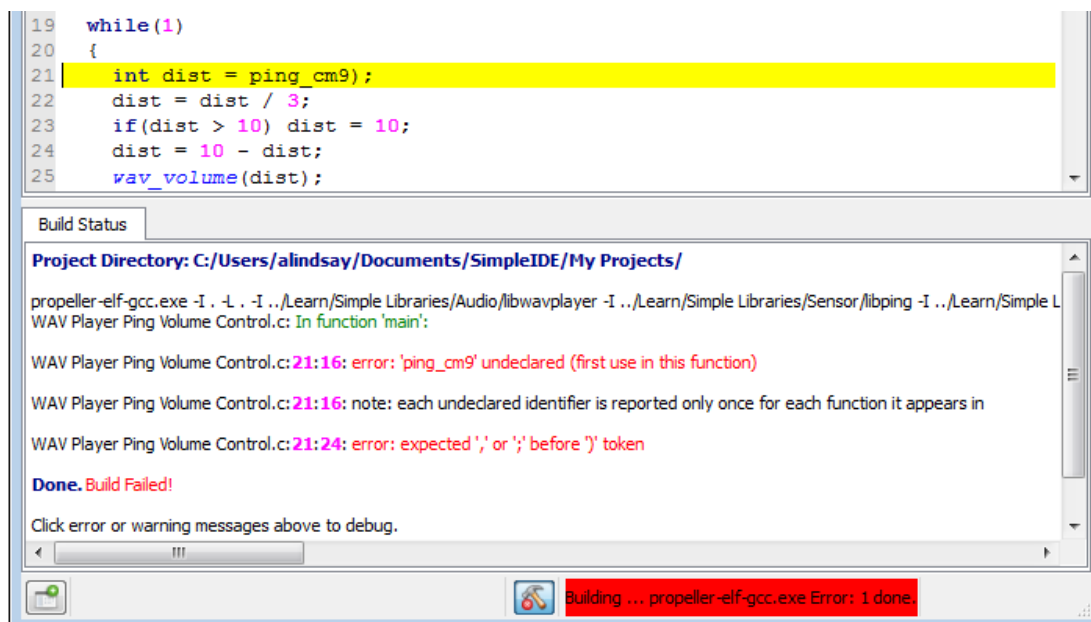
Build Status

The Build Status pane displays detailed reports from programs that cover the processes for code compilation and transfer to the Propeller system.

In this example, the project directory is displayed first. After that, it shows information passed to the Propeller GCC compiler (propeller-elf-gcc.exe), and its report back, "Done. Build Succeeded!" It then shows the information passed to Propeller Load, the status of the program image transfer, bytes sent and verification that the program was properly transferred to RAM/EEPROM.

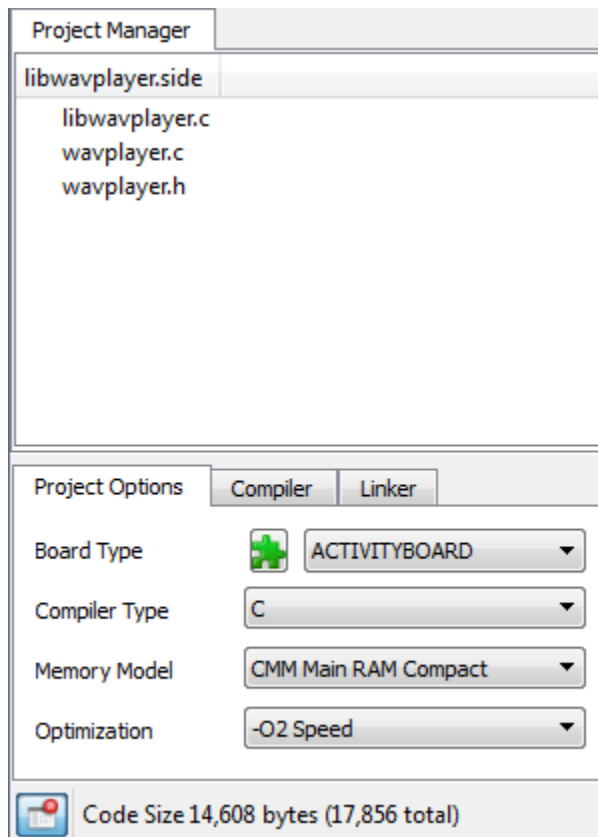


If there is a compiler error, the Build Status pane will report the line and character number where it detects each error, and the first detected error will be highlighted in the program. In this example, an opening parenthesis was deleted on line 21 to cause the compiler error. It should read `int dist = ping_cm(9);`.



Project Manager

Projects typically consist of several files. Those files must be listed in the Project Manager in order for the project to be built and run. The minimum required file in a project is the .c file with the main function (where code execution will start). In the example below, the project's name is libwavplayer.side. The .side extension is an abbreviation of SimpleIDE, and it stores the project manager settings. The file with the main function has the same name, but ends in the .c extension, and is at the top of the Project Manager's file list.



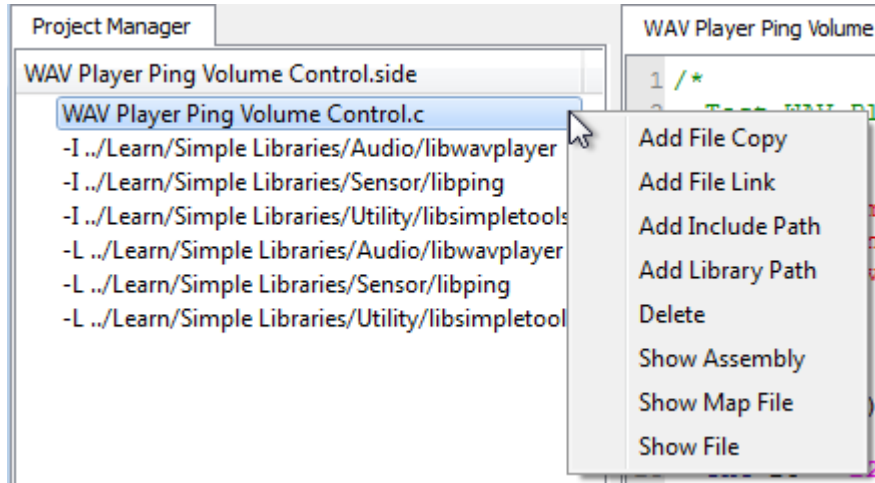
You can see how SimpleIDE uses the Project Manager to keep a list of any library, folder, and file resources the project utilizes that are not part of Propeller GCC. Libraries that are not part of Propeller GCC, or Simple Tools, have to be added to the project so that the compiler can find them. For example, you will not see stdlib listed here because that is part of Propeller GCC, so a simple `#include <stdio.h>` in the program's source code would be all that's required.

NOTES

- For C/C++ and Spin, the project manager defines what is built regardless of what is shown in the editor tabs.
- There is generally no issue using a main .c type file for projects that include C++ in Propeller GCC, provided the Compiler Type is set to C++.
- Although .h files are not listed by name, they must exist in the project folder or in one of the specified include paths for successful builds.

File Manager Tab

To see the contents of a file, left-click on the file name to open the file in the editor. Entries starting with -I and -L specify paths, not files – they will not open any content in the editor. Similarly, clicking the .side file (the project file) does nothing. Controls for adding files and paths to the file list are available by right clicking one of the entries.



- **Add File Copy:** Adds a copy of an existing file to the project. If the file is outside of the project directory, the file will be copied to the directory. File names are added in alphabetical order except the main project file which must always be listed first.
- **Add File Link:** Adds a link to an existing file into the project. Unlike the above option, a linked file will not be copied to the project directory. This is useful for reusing existing code maintained outside of this project; however, a disadvantage is that the project's code is not all in same folder. Linked files are displayed in the Project Manager as the short file name with "->" and the full path and filename afterward.
- **Add Include Path:** Adds a path into the current project pointing to a folder containing header files (.h) that are not already in the project folder or tool-chain library folder. This adds an -I path to the project.
- **Add Library Path:** Adds a path into the current project pointing to a folder containing one or more memory model folders for a particular library. The memory model subfolders, with memory model names (cmm, lmm, xmmc, etc.), each contain one or more .a files that have been pre-compiled for that memory model. Assuming each folder contains libname.a, -lname should be manually entered into the Linker tab's Other Linker Options field. This feature is not for including source libraries (.c/.cpp/.cogc); use Add File Copy or Add File Link for those.
- **Delete:** This option deletes a file or path from the project settings. It does not delete a file itself; rather, it just removes it from the project settings. The top file cannot be deleted since it has a special meaning to the project. The top file in the project must always have the main() function required by C/C++. Note that if another top file is needed, either create it or open it in Project View, and then use Project -> Set Project to set it as the top file.
- **Show Assembly:** Displays the Propeller GCC assembly with C source comments for a particular file.
- **Show Map File:** Displays the Propeller GCC code map for a particular file.
- **Show File:** This is the same as left-clicking on a file name; it will either open that file or activate the editor tab that already contains it.

Project File Types

Source Files: These files can be added to the project manager by right clicking an existing project file and using the popup menu.

- C files (.c, .cc, .cpp) typically contain function or class method implementations.
- COG C files (.cogc) are a special type of C file that will compile to an image that will run in a cog.
- SPIN files (.spin) contain PASM that can be compiled and extracted for starting in a cog.
- GAS files (.s, .S) contain PASM-like GNU Assembly that can be compiled and extracted for starting in a cog.

Include Files: These files are also known as header files, and usually contain interface information.

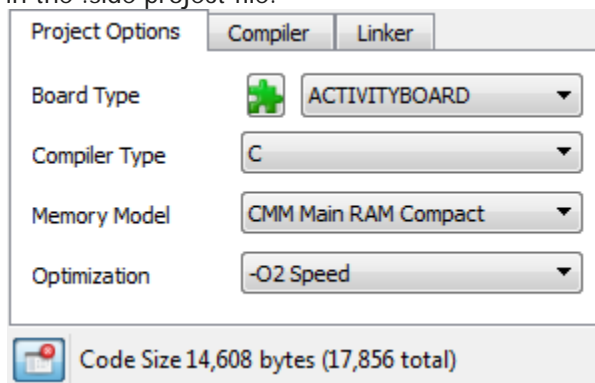
- Header files (.h) are used to define data types and declare functions that may be in libraries.
- Header files are included in C source using `#include` statements and do not need to be added to the Project Manager specifically. However, it is a good idea to add a file link because it can be used to conveniently open the header into a tab.
- If the header file is not in the same folder as the project, a path to the folder that contains the header file should be specified.
- An include path can added by right-clicking any item in the Project Manager and selecting Add Include Path. Also, while in Project View, the Project menu's Add Include Path item can be used.
- An include path can also be added with "-I folder" in the Other Compiler Options box.

Object Files: These files are generated by the build process and are not added to the Project Manager.

- Object files (.o): Object files are always generated by the compiler.
- COG Object file (.cog): This is a generated object file created from a .cogc file.
- Dat files (.dat): This type of file is generated by BSTC or other Spin compilers used for making PASM COG code.

Project Options

The Project Options tab is for choosing common project settings. These options are automatically saved in the .side project file.



Board Type

This field, to the right side of the green puzzle piece, selects the target development board type. Most commercially available Propeller boards are listed here and an abbreviated list is included in Simple View. Click this dropdown to select the board you will be loading the program into. The selected board type is saved in the .side project file.

Board types with SDLOAD or SDXMMC features are special, and when selected tell the IDE that certain functions have to be performed to load the program into external memory. In some cases, modes like RCFast or RCLow select system clock settings that have special purposes, but can be used on any board. See the Board Types section on page 31 for more info.



Reload Board Types: If board configuration (.cfg) files have been added to or deleted from Propeller GCC's Propeller Load folder (<SimpleIDE install folder>\propgcc\propeller-load\), this button will refresh the list of board names in the Board Type dropdown. If the boards.txt file (inside the Propeller Load folder noted above) has been modified, this button will update the truncated boards list shown in the Board Type dropdown while in Simple View.

Compiler Type

Three compiler types are supported:

- C
- C++
- Spin

C and C++ projects are compiled by the Propeller GCC compiler that is installed with SimpleIDE.

Spin is a custom object-based language (not related to C or C++) that Parallax developed for the Propeller microcontroller's multicore architecture. It is compiled by BSTC (Brad's Spin Tool Compiler), which is also bundled with SimpleIDE.

Memory Model

The memory model options allow you to select where to store code that gets executed and data that gets accessed. There are six memory model options that utilize various combinations of the Propeller microcontroller's 2 KB Cog Ram, 32 KB Main RAM (shared by all cogs) and various external memories including flash, SD, and even the 64 or 128 KB EEPROMs built into many Propeller boards.

Keep in mind that most memory models do not preclude other memory from being used. For example, the compact memory model (CMM) does not use SD data, but a CMM project can use libraries to read from and write to an SD card.

- **LMM Main RAM:** The Large Memory Model (LMM) stores the program image and variable data in Main RAM with machine codes fetched and executed by one or more cogs.
- **CMM Main RAM Compact:** The Compact Memory Model (CMM) is just like the LMM Main RAM model except that the program image is compiled into a size-optimized form. This is recommended for most applications that fit entirely into the Propeller's 32 KB Main RAM. There is very little difference in performance between LMM and CMM, and the memory savings is significant.
- **COG Cog RAM:** The COG model stores both program image and data in Cog RAM. COG programs are very limited with only small amounts code and local variable data able to reside in

cog memory. VGA-Pong and VGA-driver demo code are some surprising examples that will run in COG mode using Main RAM for buffering. PASM is not required.

- **XMMC External Flash Code Main RAM Data:** The Extended Memory Model Code (XMMC) mode stores program images in external flash memory, an SD card, or EEPROM. Data such as variables are placed in Main RAM.

XMMC is the best performing external memory model. It is faster than Spin programs in many cases when using SPI Flash. Code can be forced into Main RAM for best performance of time-critical tasks. This is a solution for the XMM-determinism problem – XMM code relies on a cache for performance and fetching cache lines can get in the way from one compile to the next depending on the code being used.

- **XMM-SINGLE External RAM:** The Extended Memory Model Single mode stores both program image and data on an external RAM. Data can also be forced into Main RAM.
- **XMM-SPLIT External Flash Code + RAM Data:** The Extended Memory Model Split External Flash Code + RAM Data mode stores the program image on an external flash, SD card, or EEPROM memory, and variable data on an external RAM.

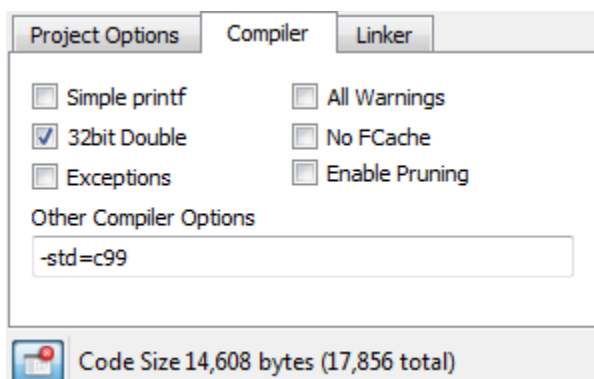
Note: XMM is compatible with libraries that launch COGC or PASM code into other cogs; however, it is not currently compatible with code that launches LMM or CMM code into other cogs.

Project examples that use a variety of memory models can be found in ...Documents\SimpleIDE\Propeller GCC Demos. Since the project stores the memory model settings, you will probably notice that most of those examples are already set to the optimal memory model for the project and target board. For example, the C-VGA demo is a COG-only program, so its memory model has been set to COG Cog RAM. Additionally, the Graphics demo can run on CMM or LMM, but it can also run on XMMC with board type EEPROM selected (a 64 KB or greater EEPROM must be used; two 32 KB EEPROM ICs will not work); approximately 6 KB of EEPROM code is used for program resources with EEPROM XMMC.

Optimization: Typically we want to optimize for size, but there are some programs that we want to optimize for speed at the cost of a larger program. Use -O2 Speed mode for speed optimizations.

Compiler Options

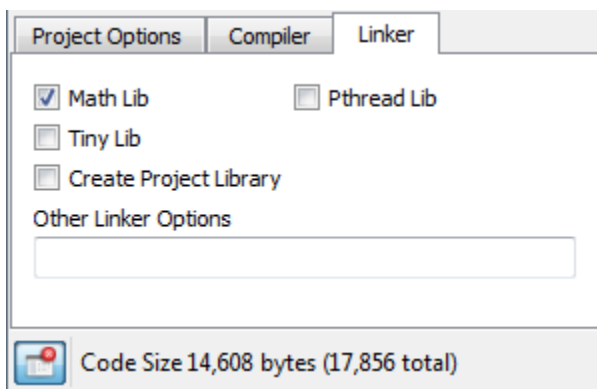
Select these options as needed for a project. The defaults are fine for most projects.



- **Simple Printf:** Deprecated. See Linker tab's Tiny Lib option.
- **32-bit Doubles:** (default) Use 32-bit doubles for floating point double variables. 64-bit doubles are too big for most LMM programs.

- **All Warnings:** When checked, the compiler will generate all possible warnings for issues in code that may cause trouble.
- **No Fcache:** Check this option to prevent the compiler from using Fast Cache (Fcache). Fcache generally improves performance but it can be disabled.
- **Exceptions:** This should be enabled only for C++ programs that use try/catch exceptions. Using exceptions may cause code size to increase.
- **Enable Pruning:** Enable this option to have the compiler and linker remove unused code from the program image. This option saves the most space in projects using non-optimized libraries; Propeller GCC uses optimized libraries, so the savings are usually minimal with this option enabled.
- **Other Compiler Options:** This allows adding -D flags for programs that may need them. There are other flags that can be added here when using libraries. Use the Project Manager's Add Include Path for using prebuilt libraries. One may need -I <path-to-library-headers> for using prebuilt libraries.

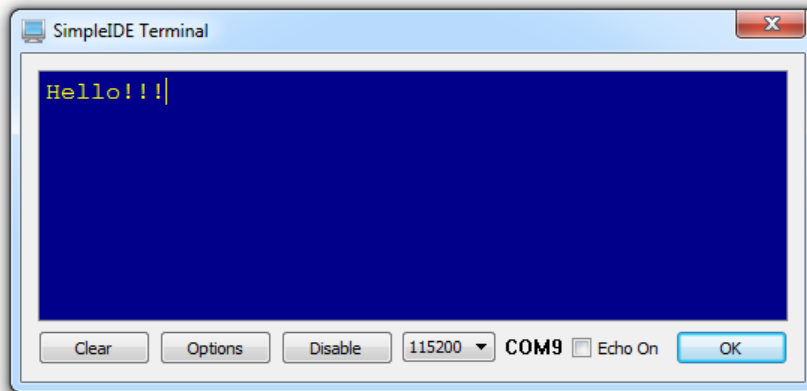
Linker Options



- **Math Lib:** Must be checked if using single or double precision floating point in the program. The program will compile without checking this, but it will not run correctly. **The Math library must be included for floating point to work.**
- **Tiny lib:** Significantly reduces code size for programs that need `printf`, but not file I/O or math library support. For example, the Welcome Application's simple hello message drops from 8.21 KB to 3.16 KB.
- **Pthread Lib:** This option must be checked if using Pthreads for running multiple threaded programs in one cog or many cogs. The number of threads available is limited by memory. XMM/XMMC programs will run all threads on the main program cog. LMM programs can run M number of threads on N cogs. For Pthreads, N cogs is limited to 8 for LMM programs, and is limited to 1 for XMM/XMMC programs. The size of M is limited only by memory available. This is an advanced feature; see <http://www.parallax.com/propellergcc> for more information.
- **Other Linker Options:** This allows adding linker-specific options. For example, "-lname" may be added for using a prebuilt library. A prebuilt library has subfolders for memory models like cmm, lmm, xmmc, etc., each containing a precompiled binary archive for the target memory model named libname.a. Special Linker scripts can be added here if a board does not fit a built-in memory layout.

SimpleIDE Terminal

The SimpleIDE Terminal displays the output from various text functions, such as the Standard C Library's `printf`, and `putc` functions, and Simple Library's `print` function. The terminal can also take keyboard input and transmit text to the Propeller to be received by functions like `scanf` and `getc`. Data from this window can also be shaded, copied, and pasted into other applications.

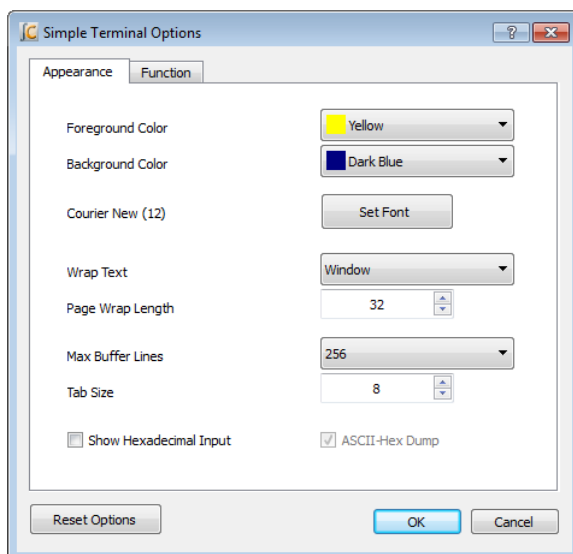


Features

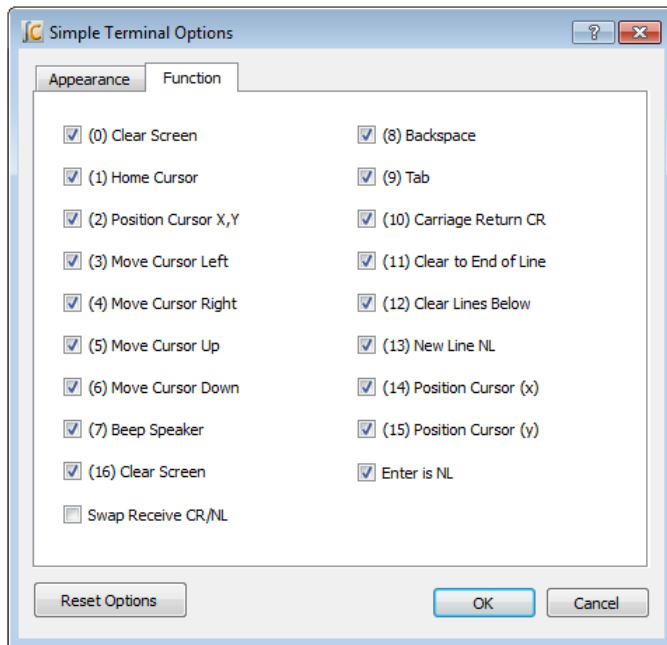
- A Clear button for clearing the terminal of any previously printed text
- An Options button for setting appearance and function
- Disable button to stop displaying incoming messages
- A baud rate selection dropdown
- A display indicating the COM Port that is/was connected
- Echo On checkbox for echoing transmitted text on the display

Options

Appearance Tab: Set color, font and tab settings. Increase the maximum buffer size setting to capture larger amounts of data.



Function Tab: Define the functions of char values that are transmitted to the terminal. For example, if checked, the value 1 sends the cursor to its top-left Home Cursor position.



Simple Libraries

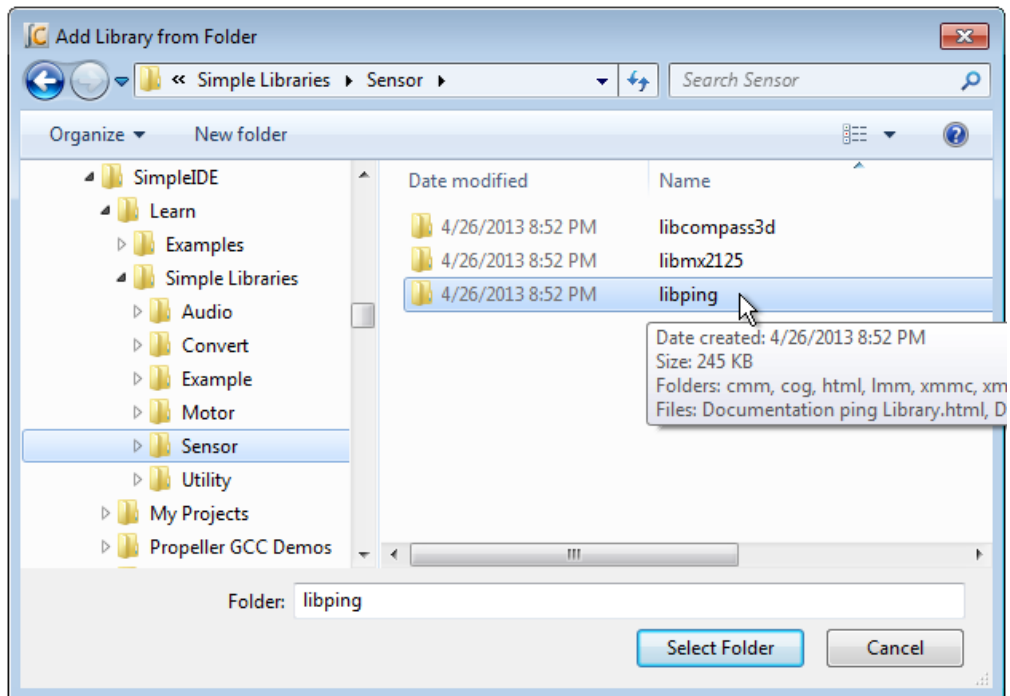
Simple Libraries are designed to be easy to add to a project with SimpleIDE. A number of Simple Libraries, including simpletools, are included in ...Documents/SimpleIDE/Learn/Simple Libraries/.

How to add a Simple Library to a Project

1. Click the Add Simple Library button.



2. Browse to the library you want to add, select it, and click the Select Folder button. Note that if you accidentally double-clicked the desired folder, you can still click the Select Folder button (without selecting any subfolder first) in order to choose the desired folder.

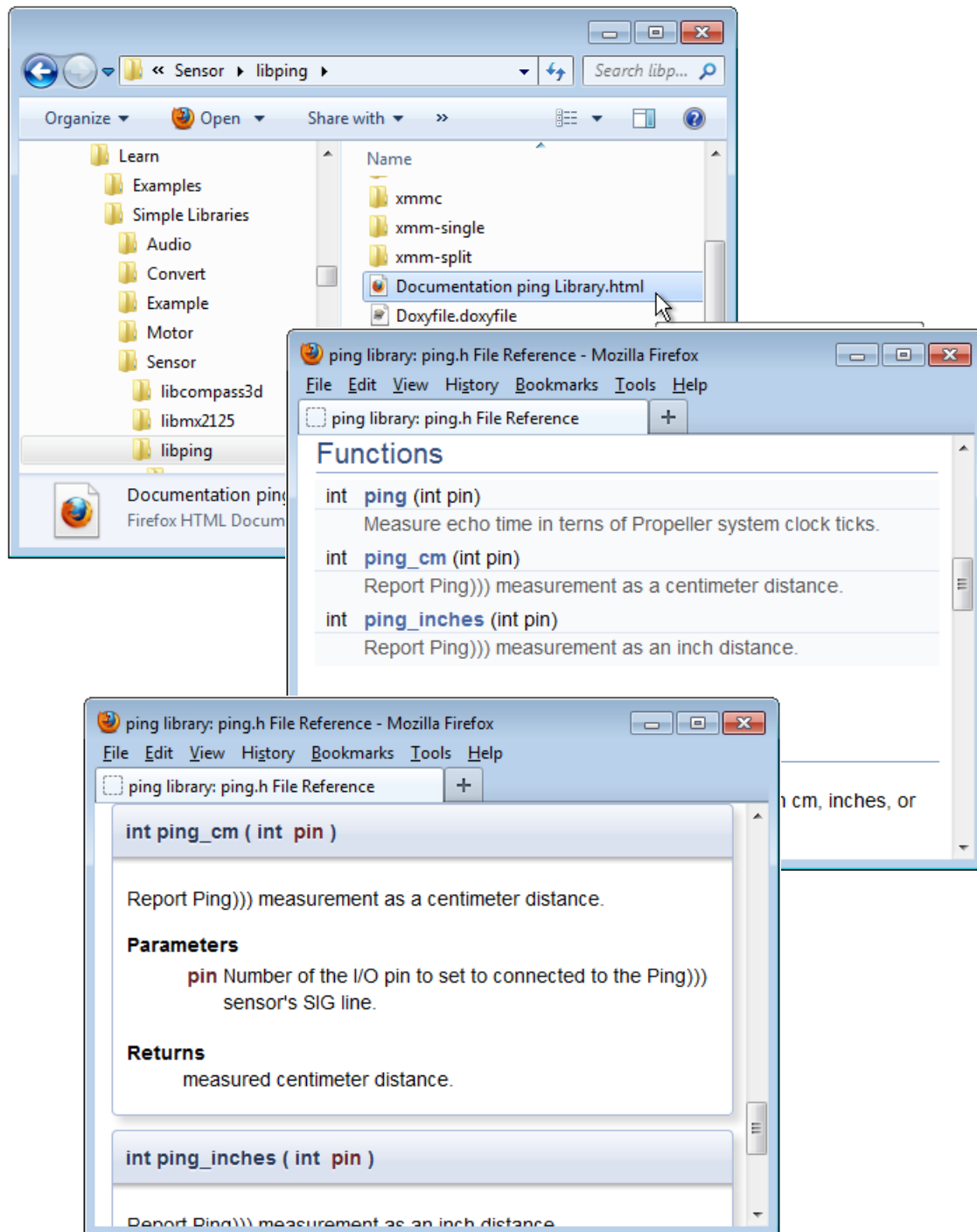


After clicking the Select Folder button, SimpleIDE will add an `#include "..."` directive for the library to your code and make the necessary additions to the Project Manager's file list and Linker tab.

Alternative

As of SimpleIDE v0.9.40, as you become familiar with Simple Library names, you can also simply insert the `#include` directive (mentioned above) into your code since the "Auto Include Simple Libraries" feature (if enabled) removes the need to adjust Project Manager's settings.

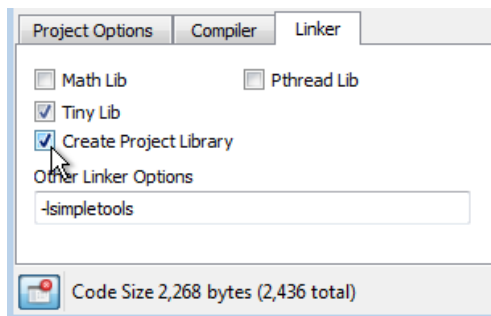
Simple Library Documentation: Use your operating system's file browser (Windows Explorer, Mac Finder) to look inside the Simple Libraries folder, and desired subfolders, and open the Documentation...Library.html file. Among other things, this documentation lists the functions the library puts at your disposal, describes what they are for, the parameters they expect, and what they return.



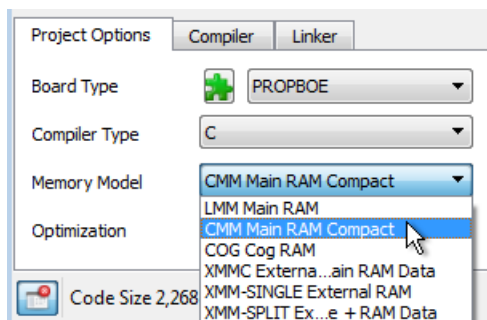
How to Create a Simple Library

To create a Simple Library (assuming it's named "name") follow these steps:

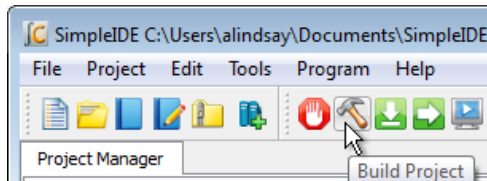
1. Select Project > New.
2. Using the New Project dialog, navigate to the desired location for your new library.
3. Click the Create New Folder icon near the upper-right of the dialog. This will add a folder called "New Folder" to the current location.
4. Rename the new folder to libname.
5. Navigate into the newly-created libname folder.
6. Change the filename (in the File name field) to libname and click the Save button.
7. Select Project > Add Tab To Project.
8. Change the filename to name, set the Save as type field to C Header File (*.h), then click the Save button.
9. If necessary, use the Project > Add Tab To Project feature to add as many .c files as needed, and try to make the names descriptive of the functions they contain. Also try to make each .c file contain as few functions as possible because the archived version will optimize out uncalled functions (.c files) if you do.
10. If you're in Simple View, click the Show Project Manager button.
11. On the Linker tab, set the Create Project Library option. This will generate a prebuilt library file (.a) in a subfolder of your new library folder every time you compile.



12. In the Project Options tab, select the desired memory model. Every time you compile the library, SimpleIDE will create an archive file (.a) in a subfolder of your library folder. For example, if CMM Main RAM Compact is the selected Memory Model, a compile of the library will create a subfolder called "cmm" with a file called libname.a inside of it. You must manually set the Memory Mode and recompile for every memory mode you want your library to support.



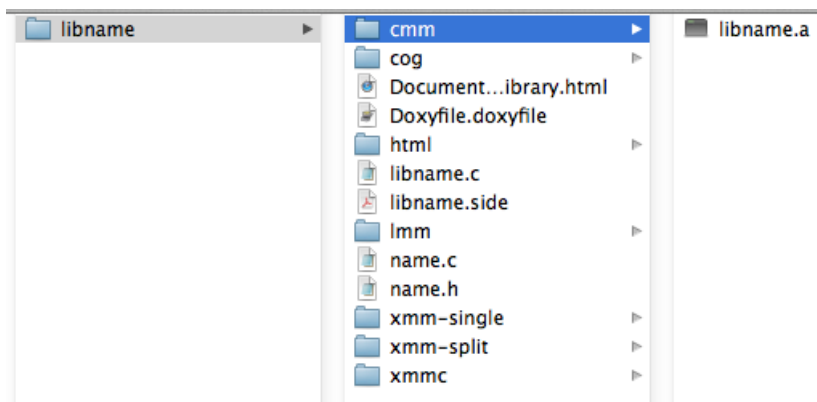
- Click the Build Project button whenever you want to compile and build the library's archive file for the currently selected Memory Model.



The selected Board Type does not affect the library's archive file (.a); it may be set to any board type while building the desired memory models.

Simple Library Directory Structure

Here is an example of how the directory structure might look for a library called "name." Note that the libname folder contains name.h and memory model subfolders, each with libname.a file. Each libname.a file is a binary archive compiled for the memory model of the folder it's in.



A Simple Library should only contain memory model subfolders for memory models it supports. For example, if the library supports only CMM and LMM, it should have cmm and lmm subfolders, but no cog, xmmc, or other memory model subfolders. Check subfolders in ... \SimpleIDE\Learn\Simple Libraries for examples.

Recommended Features

Try to make each .c file in the library project atomic. In other words, keep one function, or a group of interdependent functions, in each .c file. When the .a file is created, it will be organized so that the compiler and linker can optimize out any functions (from separate .c files) that are not called by the project being compiled; expanding a project's image size only by the size of the functions that are called (along with any they depend on). In contrast, if the library included a single, all-encompassing .c file, with every function the library supports, then even a single function reference to the library will increase the program image size by the size of the entire library; the size of all the code for all functions within the library.

Doxygen (<http://www.doxygen.org>) is the recommended documentation tool. Try to fully document each element in the header file with Doxygen comments, and then create an html folder for your library using Doxygen. The examples in \SimpleIDE\Learn\Simple Libraries also copy name_8h.html to the libname folder and rename it Documentation name Library.html. Then the following search/replaces are performed on Documentation name Library.html with a text editor:

Search: href=""
Replace: href="html/"

Search: href="html/#"
Replace: href="#"

Board Types


SimpleIDE allows specifying board types from a simple dropdown box in Project Options. There are many board types and variations. Board type configuration files (.cfg) contain information for the loader to use when starting the program. The loader needs the main serial port, the clock mode, the clock frequency, cache driver for external memory programs, and SD card pins.

Board-specific pins can be defined in the .cfg file and “stuffed” into the Propeller program at load time so that several board types can use the same program without recompiling it. More details can be found in the propeller-load document on the PropGCC open source project website:
<https://code.google.com/p/propside>.

Special Clock Boards

If the Propeller development board does not use an 80 MHz system clock, or has an odd crystal configuration, a board type can be created to set the clock frequency (typically crystal frequency times wind-up multiplier; specified by PLLx mode). Board configuration files can be placed in the project folder, but it is best to put them in the installation's propeller-load subfolder.

Assuming <SimpleIDE Installation Folder>\propgcc is the installation folder, define a custom board as follows:

1. Copy the <SimpleIDE Installation Folder>\propgcc\propeller-load\hub.cfg file to <SimpleIDE Installation Folder>\propgcc\propeller-load\custom.cfg (or choose a name that does not have spaces).
2. Change clock frequency or clock mode in custom.cfg, and save the file.
3. If you want the board type to be available in the Project Manager's Board Type dropdown (even in Simple View), open boards.txt (from the same folder) and add your custom board's filename to the list. Boards.txt is a filter; if boards.txt is removed then all board types will be available in Simple View upon refresh.
4. Click the puzzle piece  near the Board Type field in the Project Options tab to refresh the board types list.
5. Choose CUSTOM from the board type drop down box to compile and download for the CUSTOM board you just created.

Note: RCFAST and RCSLOW board types are available in the board types list, but these should never be picked for programs that communicate through SimpleIDE Terminal because the timing is not accurate enough for serial communications.

Basic Board Types

Basic board types can work on many boards that have only a Propeller, crystal, and an EEPROM. A crystal is not necessary for RCSLOW and RCFAST board types. An EEPROM is only required if the Propeller needs to boot without the help of download from SimpleIDE.

- **GENERIC:** Supports basic hardware features of the most common Propeller boards including: Propeller Activity Board, Propeller Board of Education, PE Platform, Propeller Demo Board, and Propeller QuickStart. Although all these boards are supported, they also have their own entries in the Board Types dropdown. Each individual board type may also have extra features not included in this generic setup, so you should select the exact board type whenever it is known in advance. GENERIC settings include a 5 MHz crystal with PLL set to 16x for an 80 MHz system clock, 32 KB or larger EEPROM connected to P29 (SDA) and P28 (SCL), and 115 baud communication with a terminal via P30 (host computer Rx) and P31 (host computer Tx).
- **RCFAST:** This board type makes the Propeller microcontroller rely on its internal 12 MHz oscillator. Although it can actually boot up on any Propeller board, it is designed for applications that do not require the clock precision of an external oscillator. This mode is impractical for serial communication, servo control, precise pulse measurement, or video.
- **RCSLOW:** RCSLOW is like RCFAST except that it uses the slowest and most energy efficient clock mode in the Propeller, 20 kHz.
- **HUB:** The HUB board type specifies an 80 MHz system clock and an external 5 MHz crystal with PLL16x clock mode. Any board that has a 5 MHz crystal should work with HUB board type. Good serial communications and TV/VGA output are possible with the HUB board type and an accurate 5 MHz crystal. Some of the other board types related to HUB are ACTIVITYBOARD, C3, QUICKSTART, PEKIT, and PROPBOE.
- **SPINSTAMP:** The SPINSTAMP board type is like the HUB board type except that it relies on a 10 MHz crystal to produce an 80 MHz system clock by using PLL8x in the clock mode. Any Propeller board that has a 10 MHz crystal should work with the SPINSTAMP board type. HYDRA is a related board type.

EEPROM Board Types

The EEPROM board type is like the HUB board type except that it has a cache driver defined for running XMMC memory model programs. With the EEPROM or similar board types, a program loaded into the EEPROM can be fetched and run with XMMC.

This mode is usable with a single 32 KB EEPROM, 64 KB EEPROM, 128 KB linear address space EEPROMs, and 256 KB EEPROMs from ST.

Any set of 64 KB+ EEPROM can be configured to add more code space to the program. Using two separate 32 KB EEPROMs will not work because address 0x8000 to 0xFFFF maps to address 0x0000 to 0x7FFF. All devices should be the same type. MCP29FC1025 devices do not have a linear memory addressing and will not work.

The XMMC model is the only extended (XMM*) memory model that will work with EEPROM board types. Other board types related to EEPROM that run XMMC programs are ACTIVITYBOARD, PROPBOE, QUICKSTART, and ASC+.

External Flash Board Types

External Flash board types like C3F and SSF will run XMMC memory model programs from Flash memory.

- **C3F:** C3F XMMC programs are stored in the on-board 1 MB SPI Flash. C3F is a variant of the C3 board type that uses all of cache for program code and is thus faster than C3. (The C3 type splits cache between Flash and SRAM storage and is less efficient).
- **SSF:** SSF is the SpinSocket-Flash board type that uses 2 Winbond W25Q* QuadSPI parts for storing and running XMMC memory model code in. This is a fast practical external memory solution because of low cost, low pin count, high density, and high relative XMM performance.

External RAM Board Types

External RAM boards have external SRAM, SPI-SRAM, or SDRAM. Boards having only SRAM will only boot stand-alone if the code can be loaded from SD card or some other non-volatile storage. SRAM-only boards can be loaded by the PC and SimpleIDE using the serial loader protocol for testing.

- **C3:** The C3 type allows using the XMM (or XMM-SPLIT) memory model to store program code in SPI Flash and data in a device like external SPI-SRAM. It is possible for a cache driver to be written for other boards that will use the XMM memory model. The C3F board type will only use C3 Flash and only works with XMMC memory model programs.
- **DRACBLADE:** This board has SRAM and SD card. It can be loaded by the IDE for testing, but must be programmed using SDLOAD for stand-alone boot.
- **SDRAM:** This board has SDRAM and SD card. It can be loaded by the IDE for testing, but must be programmed using SDLOAD for stand-alone boot.

SDLOAD Board Types

SDLOAD board types are typically RAM board types that do not have an on-board Flash. With the SDLOAD board type, the program to be run is sent to the SD card. The Load RAM & Run, Load EEPROM & Run, or Run with Terminal buttons will cause the AUTORUN.PEX output to be downloaded to SD card and then booted to RAM and run. Once the Load EEPROM & Run button has been used, the AUTORUN.PEX file can be replaced using the Send File to SD Card feature or by connecting the SD card to your computer and replacing it with file management tools like Windows Explorer or Mac Finder.

SDXMMC Board Types

SDXMMC board types are used to download XMMC memory model AUTORUN.PEX programs to an SD card and run them using the Load RAM & Run, Load EEPROM & Run, or Run with Terminal. Any board that has an SD card can use SDXMMC; however, code execution may be very slow.

SimpleIDE SDLOAD and SDXMMC Attributes

How does a board type include SDXMMC or SDLOAD attributes? This is added to the .cfg file for a board. If a board has an SD Card, the SDXMMC option can be added to the .cfg file with the line "# IDE:SDXMMC". If a board has an SD Card and a supported RAM cache driver, SDLOAD can be added to the .cfg file as "# IDE:SDLOAD". Some examples follow.

Below is an example of the pusb.cfg file. Note that it has IDE:SDXMMC in a comment as described above. Other interesting items are cache-driver and sd-driver. The cache driver in this case is the eeprom_cache.dat driver. Today, the SDXMMC cache driver is part of the sd_driver.dat.

To use SDXMMC for this example, the board type PPUSB-SDXMMC should be selected. To use the EEPROM cache for XMMC, the board type PPUSB should be selected.

```
# ppushb
# IDE:SDXMMC
  clkfreq: 80000000
  clkmode: XTAL1+PLL16X
  baudrate: 115200
  rxpin: 31
  txpin: 30
  cache-driver: eeprom_cache.dat
  cache-size: 8K
  cache-param1: 0
  cache-param2: 0
  eeprom-first: TRUE
  sd-driver: sd_driver.dat
  sdspi-do: 0
  sdspi-clk: 1
  sdspi-di: 2
  sdspi-cs: 3
```

Configuration Files

Board configuration files provide customization for Propeller GCC board hardware. A Propeller GCC program is loaded to the hardware with the propeller-load program (see <http://www.parallax.com/propellergcc/>). The loader will scan the board type .cfg file to use with the compiled Propeller GCC program and patch properties to the program if necessary before loading.

Configuration Variable Patching

Numeric properties found in the .cfg file can be applied to a Propeller GCC program to let the program run on boards with different hardware connections. Any variable can be defined. See <http://www.parallax.com/propellergcc/> for more information.

As a simple example, one board can have an LED on pin 15 and another board can have an LED on pin 20. The same code can run on boards using different .cfg files.

```
# file: led15.cfg
# LED pin 15 example
  clkfreq: 80000000
  clkmode: XTAL1+PLL16X
  baudrate: 115200
  rxpin: 31
  txpin: 30
  ledpin: 15

# file: led20.cfg
# LED pin 15 example
  clkfreq: 80000000
  clkmode: XTAL1+PLL16X
  baudrate: 115200
  rxpin: 31
  txpin: 30
  ledpin: 20
```

The Propeller GCC program that uses **ledpin** can look like this:

```
#include <propeller.h>
/* Config variables must be global.
 * If a variable is patched it will not have the value -1.
 */
int _cfg_ledpin = -1;
int main(void)
{
    if(_cfg_ledpin > -1) {
        DIRA |= (1 << _cfg_ledpin);
        while(1) {
            waitcnt(CLK_FREQ/2+CNT);
            OUTA |= (1 << _cfg_ledpin);
            waitcnt(CLK_FREQ/2+CNT);
        }
    }
    while(1);
    return 0;
}
```

Future Improvements

Opportunities for improving SimpleIDE are listed below. Email ideas to SimpleIDE@parallax.com.

1. Auto-install the FTDI USB driver for communication with Parallax boards.
2. Propeller auto-detect.
3. Add a built-in browser for allowing F1 Context sensitive help and opening the SimpleIDE User Guide and library documentation.
4. Code autocomplete - Add library function and object "dot" lookup to make programming easier.
5. Debugging tool.

Support

Report problems with SimpleIDE or Propeller-GCC via email to SimpleIDE@parallax.com, or by adding issues to <http://code.google.com/p/propside/issues/list>.

Revision History

There have been many untracked enhancements and bug fixes during the life of this document. In the future, significant bug fixes may be tracked here. For complete details, look at <http://propside.googlecode.com> for complete software revision history and source files.

- Version 0.9.43 September 5, 2013 untracked.
- Preliminary August 6, 2013 untracked.
- Preliminary May 26, 2012 untracked.
- Preliminary Beta June 16, 2012 untracked.
- Beta April 30, 2013 untracked.
- Version 0.9.26, first released with SimpleIDE version 0.9.26.