

# Eclipse/CDT for eCos application development

John Dallaway

## Introduction

Eclipse is an open-source framework and workbench for the implementation of Integrated Development Environments (IDEs). CDT is a set of additional code modules (plug-ins) for the Eclipse framework which provide C/C++ editing and debugging facilities. This document describes the process of configuring Eclipse and CDT for eCos application development. Some familiarity with Eclipse and CDT is assumed. More general user guides for the Eclipse workbench and for C/C++ development using CDT are accessible via the *Help->Help Contents* menu item within the Eclipse workbench.

*eCosCentric* provides modified versions of certain CDT plug-ins and additional plug-ins suitable for the building and debugging of eCos applications running on remote hardware using GDB. Enhancements include:

- The facility to download executable code to remote targets
- The management of remote target communication timeouts during long downloads
- The facility to use a BDM parallel port interface for debugging (specific targets only)
- The robust handling of remote target communication failures
- The presentation of eCos thread information while debugging
- The facility to insert hardware-assisted breakpoints where supported by the GDB stub
- The integration of GNU cross-development toolchains for C/C++ projects

## Creating eCos application projects

eCos application development requires the use of GNU toolchain components including the GCC compiler, LD linker and GDB debugger. CDT assumes that the locations of all compilation and debugging tools are either specified absolutely within makefiles and dialog boxes or that their locations are listed in the *PATH* environment variable. The *eCosCentric* installer will ensure that the GNU toolchain supplied with the eCosPro Developer's Kit is on the *PATH* when you invoke Eclipse via the appropriate eCosPro start menu item.

CDT supports the creation of both *Managed Make* and *Standard Make* C/C++ projects for eCos application development. The use of Managed Make projects is recommended for new users. Standard Make projects provide greater flexibility for those familiar with the creation of GNU makefiles.

### Managed Make projects

To create a new Eclipse *Managed Make* project for eCos application development, invoke the *New Project* wizard using the *File->New->Project...* menu item. Select *C Project* or *C++ Project* from the tree of project types within

the *New Project* wizard depending on whether the project will contain any C++ code. On the next page of the wizard, enter the project name and select the *Executable* or *Static Library* project type. Due to limitations within the GNU toolchain, the project name and project location should not contain any spaces. Finally, select the eCos GNU toolchain for the relevant processor architecture and click *Finish*.

The eCos install tree containing the eCos library and header files is specified by modifying the value of the `eCosInstallDir` build variable within the project's *Properties* dialog box. Select the project name in the *C/C++ Projects* view and then invoke the *Properties* dialog using the *File->Properties* menu item. Select *C/C++ Build->Variables*, check the *Show system variables* box, select the `eCosInstallDir` variable and then click on the *Edit* button. Specify the path to an existing eCos install tree as the value of the build variable and click *OK*. You will need to repeat the above procedure for each configuration of your project. Select a configuration using the *Configuration* drop-down list box near the top of the dialog box and modify the `eCosInstallDir` build variable appropriately.

## Standard Make projects

To create a new Eclipse *Standard Make* project for eCos application development, invoke the *New Project* wizard using the *File->New->Project...* menu item. Select *C Project* or *C++ Project* from the tree of project types within the *New Project* wizard depending on whether the project will contain any C++ code. On the next page of the wizard, enter the project name and select the *Makefile project->eCos Application Project* project type. Due to limitations within the GNU toolchain, the project name and project location should not contain any spaces. Finally, select the eCos GNU toolchain for the relevant processor architecture and click *Finish*.

The resulting project will include a `Makefile` file containing two makefile variables which the user should edit to adapt the makefile to a specific project:

- *TARGET* - the application executable file name (with no suffix) or static library filename (with `.a` suffix).
- *SOURCES* - a space-separated list of C, C++ and assembly source files (`*.c *.cc *.cpp *.cxx *.s`) to be built.

The makefile includes rules to generate and use header file dependency information automatically. This dependency information is stored in files with a `.d` suffix.

The eCos install tree containing the eCos library and header files is specified by defining a *build variable* within the project's *Properties* dialog box. Select the project name in the *C/C++ Projects* view and then invoke the *Properties* dialog using the *File->Properties* menu item. Select *C/C++ Build->Variables* and then click on the *Add* button. Enter the variable name `eCosInstallDir` and set the *type* to *directory* within the *Define a new build variable* dialog box. Specify the path to an existing eCos install tree as the value of the build variable and click *OK*.

## Debugging eCos applications

### Setup

Prior to debugging an eCos application, it is necessary to generate a suitable Eclipse debug configuration. The debug configuration specifies the name of the executable file and how the contents of this file may be downloaded

to the remote target and launched. To create a debug configuration, invoke the *Debug* dialog box using the *Run->Open Debug Dialog...* menu item. Select *C/C++ Local Application*, click on the *New launch configuration* icon to create the new configuration and then give it a meaningful name such as "ARM target on /dev/ttyS1". Then proceed with the following steps:

- On the *Main* tab, click on the *Browse...* button to select the project from which the executable file is created and then click on the *Search Project...* button to select the C/C++ application executable file to be debugged.
- On the *Debugger* tab, select *gdb Remote Debugger* as the debugger, enter the name of the GDB debugger (eg `arm-elf-gdb`) on the *Main* pane and then select the appropriate connection parameters from the *Connection* pane.

On Windows-hosted installations, it is also necessary to specify a path mapping for POSIX-style file paths before debugging. Invoke the *Preferences* dialog box using the *Window->Preferences...* menu item and select the *C/C++->Debug->Common Source Lookup Path* item from the tree of preferences. Click *Add...* and choose *Path Mapping*. A new path mapping entry will immediately appear in the *Common Source Lookup Path* tree. Click *Edit...* to modify this item, give the path mapping a meaningful name such as "Cygwin root" and then click *Add...* to specify the mapping paths. Enter a single forward slash character ("/") as the *Compilation path* and browse to the *Local file system path* directory corresponding to the POSIX-style root directory (typically `C:\cygwin`). This path mapping ensures that the POSIX-style source code references in eCos application executable files are interpreted correctly by CDT.

## Hardware debuggers

When using hardware debuggers such as those using a JTAG or BDM interface, it may be necessary to override the default behaviour of CDT for launching a debugging session. For example, an external tool may be required to install code into Flash memory prior to debugging. The eCosCentric plug-ins allow the user to perform such customisation by specifying a *GDB command file* on the *Main* pane of the *Debug* dialog box. The following optional GDB macros may be defined within the GDB command file:

- *setup* - this macro is invoked by CDT before connection to the target hardware is attempted. Within the macro, `$arg0` can be used to refer to the ELF executable file which is to be debugged and `$arg1` will provide the connection parameters.
- *preload* - this macro is invoked by CDT following connection to the target and before code download.
- *doload* - this macro is invoked by CDT to download code to the target hardware. The presence of this macro will suppress the default CDT code download behaviour.
- *postload* - this macro is invoked by CDT immediately following code download to the target hardware.

