UCSD Embedded Linux online
Week 3

Slide 1

Welcome to week 3 of Embedded Linux online. This week we'll take a close look at the Eclipse Integrated Development Environment (IDE) with particular emphasis on how it can help you be more productive in writing embedded applications.

If you've played with any commercial embedded tool vendor's IDE lately, you've already been introduced to Eclipse. All of the major tool vendors such as Wind River, Monta Vista, Lynux Works, and so on, have adopted Eclipse as the basis for their tool sets.

Slide 2

First, a little bit of background. Eclipse grew out of a project that began in 1998 at Object Technology International (OTI), a subsidiary of IBM now known as the IBM Ottawa Lab. The project was intended to address complaints raised by IBM's customers that the company's tools didn't work well together.

Then in 2001, IBM established the Eclipse consortium and released the entire code base, estimated to be worth $40 million at the time, as open source. The idea was to let the open source community control the code and let the consortium deal with commercial relations. The initial nine members of the consortium included both partners and competitors of IBM like Rational and TogetherSoft at the time.

As Eclipse grew and evolved, IBM wanted more serious commitment from vendors, yet vendors were reluctant to make a strategic commitment as long as they perceived that IBM was in control. This problem was addressed in 2004 with the creation of the Eclipse Foundation, a not-for-profit organization with a professional staff and a large and growing roster of commercial software vendors as members.

Slide 3

In week 1 I made the point that there are a number of Open Source licenses, each with slightly different provisions, yet all conforming to the Open Source Initiative's definition of what constitutes Open Source. The Eclipse Public License is one of these.

Remember with the GPL, any code that in any way makes use of, or relies on GPL code is by definition also GPL. In the interest of encouraging commercial adoption, the Eclipse Public License allows "plug-ins" to remain proprietary. We'll define plug-in in just a little while.

Slide 4

The Eclipse Foundation requires a rather high level of commitment from its larger members. Because of that, the project seems to be very well managed, and I would go so far as to say that in my opinion, Eclipse is probably the most professional, highly polished Open Source software package around.

In addition to the Eclipse platform itself, Eclipse comprises dozens of tool-oriented and application-oriented projects that operate as independent open source projects. For the past several years, usually in the June time frame, the foundation has organized a coordinated major release of the platform along with a large number of the constituent sub-projects. This allows users to try out new features without worrying about version incompatibility among the various tools.

The 2015 release, called Mars, contains 79 different Eclipse projects and 65 million lines of code developed by 352 contributors.

The Eclipse sub-projects are grouped into ten major project areas. Of particular interest to embedded developers are the Device Software Development Platform (DSDP) and the Development Tools projects. Under Development Tools is the C/C++ Development Tool (CDT) project, which is a major focus of this week's lesson and the basis for commercial IDEs using Eclipse.

Slide 5

So what is the thing called Eclipse? Technically, it's not an IDE, but is rather a platform for building IDEs and what are termed "rich client" applications. I haven't come across any definition of just what a rich client application is, but it would seem to involve at the very least a nice graphical interface. If anyone has some insight into what rich client means, please post it to the discussion board.

The Eclipse Foundation's website describes it as "an extensible development platform, runtimes and application frameworks for building, deploying and managing software across the entire software lifecycle." One early technical overview paper described it this way: "The Eclipse Platform is an IDE for anything, and for nothing in particular."

The Eclipse platform itself is largely written in Java and, in fact, it was originally designed as an IDE for Java development. So as a Java app it runs anywhere, in principle anyway, as long as you have a Java runtime environment. There is a Windows version. But there is a problem with doing C development in Eclipse under Windows. The Microsoft tools don't' "play nicely" with Eclipse and so you have to install another package called Cygwin, spelled C-y-g-w-i-n, that provides the GNU tool chain in a Windows environment. Quite frankly, it's a little klutzy, but if you really insist on doing your development in Windows, you can.

The basic Eclipse platform provides only very generic capabilities. All of the application-specific functionality comes from plug-ins. Plug-ins, also written in Java, conform to a very specific set of APIs and are dynamically loaded at run time as they are needed.

Slide 6

This slide attempts to capture graphically the basic structure of Eclipse. The platform consists of a *Workbench*, which as we'll soon see, is the user interface. It's built on a couple of standard graphic toolkits. Other elements of the platform include the *Workspace* where your project files are stored, a *Help* subsystem, and a *Team synchronizing* component. All of this sits on top of an architecture and operating system-specific *Platform runtime*.

All of the other functionality is provided by plug-ins.

Slide 7

The actual installation of Eclipse is trivial and was done as part of the install process for the class software. For the record, all it involves is downloading a rather large tar file from eclipse.org, currently pushing 150 MB in the case of the C development version, and then just untarring it in a suitable location. The installation script put it under /usr/local.

As shown here, there are basically two ways to start Eclipse, indeed to start any program from the graphical desktop. You can type the appropriate command into a shell window. Note that the ampersand means run this program in the background. The shell returns to the command prompt. Or you can double-click the executable in a file manager window.

To avoid a lot of typing, I create a shell script named eclipse in /usr/bin that just has the first line of this slide. Then all I have to type in the shell is eclipse. Be careful though because your distro may include its own version of Eclipse that probably isn't the C development version that we're using.

So go ahead and get it started and we'll take a tour of Eclipse features.

Slide 8

The first thing Eclipse does when it starts up is ask you to select a *workspace*. As I mentioned a couple of slides back, the workspace is where all your project files are stored. By default the workspace is in a directory named workspace/ under your home directory.

Each project gets its own directory under workspace/.

You can have multiple workspaces, but only one can be open at a time. When you switch workspaces Eclipse effectively restarts itself.

If the workspace didn't exist, it will be created. In this case, the next thing you see is the Welcome screen. This offers the opportunity to learn more about Eclipse before jumping right into it. We'll skip it for now.

The next time you start Eclipse in the same workspace the Welcome screen won't be displayed but you can always get back to it by clicking **Help > Welcome**. For now click the **Go to the workbench** icon.

Slide 9

What you see now is the *workbench*, the primary user interface for Eclipse. This is the Eclipse "personality" if you will, and supplies the structures that allow tools to interact with you the user. The workbench is synonymous with the Eclipse Platform UI as a whole and with the main window you see when Eclipse is running. Right now of course, the workbench is pretty empty. We'll fix that in the next slide.

This slide identifies the basic elements of the workbench. Like any good graphical program it has a menu bar across the top with familiar entries like **File, Edit, Search, Window,** and **Help** as well as some menu items specific to Eclipse like **Refactor, Navigate, Run,** and **Project**. Below that is a tool bar with icons representing frequently used functions. These icons change depending on which perspective is visible and which view or editor has the focus.

At the far right of the tool bar is a box that identifies the currently displayed perspective. Just to the left of that is an icon for switching perspectives.

The Workbench window contains one or more Perspectives that are, in turn, collections of Views and Editors. A *Perspective* defines an initial set of views, and the layout of those views, to accomplish some specific task on a particular set of resources, or files. The workbench is currently displaying the C/C++ perspective typical of Eclipse CDT.

A workbench may have several perspectives open but only one perspective is visible in a window. To make additional perspectives visible, open additional windows using the **Window -> New Window** command.

The large space in the center of the workbench is the *Editor*. As you might expect, the editor allows you to open, modify, and save files. The editor window is the central feature of virtually all Eclipse perspectives. Different editors can be associated with different file types. Opening a file then starts up the corresponding editor, which may also change the contents of the menu and tool bars. The editor associated with C source files has a number of useful features that we'll look at shortly.

Views support editors and provide alternative presentations of the information in a project as well as ways to navigate that information. Views most often appear in tabbed stacks to the right and left of the editor window and sometimes beneath it. Icons on the right end of the tab bar allow the currently visible view in that stack to be minimized or maximized. Views also have their own menus represented by the down arrow icon at the far right of the view tab. Frequently used menu items may be represented by other icons in the tab.

A view can be moved around anywhere in the Workbench by dragging its title bar. As you move the view around, the mouse pointer changes to one of a set of *drop cursors* that indicates where the view will be docked if you release the mouse. Try it with the Outline view on the right.

Slide 10

It's helpful to have a project open in order to discuss Eclipse basic concepts. Select **File > New > C Project**. In the **Project name**: field enter "hello". Under **Project types:** click the right arrow next to **Executable** and select **Hello World ANSI C Project**. Click **Next** to bring up the Basic Settings dialog. Enter your name as the Author and change the Copyright notice and Hello world greeting if you choose. Click **Finish**. The hello project now shows up in the Project Explorer window on the left side of the workbench.

Click the right arrow next to hello in the Project Explorer to expand it. Double-click hello.c to open it in the editor.

Slide 11

As we've just seen, the Project Explorer view provides a hierarchical view, or roadmap if you will, of the resources contained in a project. It allows for adding or importing new files or directories, deleting or exporting files, and opening files for editing.

The Outline view displays an outline of the structural elements of the file currently visible in the editor window, in our case hello.c. In the Outline view, select the main() function. Note that the extent of the main() function is highlighted in the Editor window in the vertical bar on the left known as the *marker bar*. The extent of main() is also highlighted if you click anywhere in the function.

Right-click on one of the header file entries and note that one of the options is to open the file. The other thing we'd see in the Outline view, if we had any, is global variables.

Slide 12

Let's take a look at the four views that show up beneath the Editor. The Problems view lists any problems encountered while building the project. Incidentally, as soon as you created the hello project Eclipse went ahead and built it using the default gcc compiler. There's nothing in the Problems view because there were no problems.

It's easy enough to introduce an error so we can see what the Problems view looks like. Delete the semicolon at the end of the puts statement. Save the file with File > Save from the menu, or click the Save icon in the tool bar, then build the project by selecting Project > Build Project, or click the Build icon, the hammer. You should see two error messages show up in the Problems view and the location of the error is identified by icons in the editor's marker bar.

It's worth noting at this point that by default Eclipse does not automatically save any modified files when you build the project. There is a preference option to save automatically before a build. We'll look at preferences a little later.

The Tasks view is effectively a "to do" list. The interesting thing about it is that you can link tasks to specific lines in source files. Try this. Right-click in the marker bar next to the copyright line in the header comment. One of the menu options is Add Task. Select that. This brings up a Properties dialog. In the description field enter something like "change copyright notice." The other text boxes identify what this particular task points to. Click OK and the new task shows up in the Task view. An icon representing a task also shows up in the marker bar next to the line in question. If you double-click the entry in the Task view, the corresponding line is highlighted in the editor.

The Console view serves two purposes. It displays the output of the build process and also serves as the terminal for a program being run or debugged locally. Right now it should show the output of the failed build.

The Properties view, in my estimation, is not terribly useful. Select the Properties view and then click on something in the Project Explorer. You'll see some very basic information about the item you selected. Every object in Eclipse has certain properties, the nature of which depends on what the object is. There is another, more useful and editable view of properties available from an object's context menu.

Slide 13

Let's take a quick look at some of the basic features of the editor. The syntax coloring is clear. We'll see additional coloring elements when we look at more extensive examples.

Click just to the right of the opening brace in main() and note that the closing brace is highlighted. Try the same thing with the left parenthesis of the puts() statement.

Roll your mouse over puts. A help window pops up showing the function declaration from the header file along with any comments associated with that declaration.

With such a simple example, the auto completion and fill features are a little hard to demonstrate so we'll look at them a little later.

Code folding is an interesting feature. It's a way of hiding non-essential details, thus making more of the useful content visible on the limited screen. Note the top line of hello.c. It appears to be a comment but there aren't any comment delimiters visible. Click the plus sign in the marker bar. It changes to a minus and now the whole comment is visible. Click the minus next to the line that starts main(). The function disappears. So folding helps you focus on what's important at any given time without being distracted by unrelated elements.

Slide 14

Now let's take a look at the Eclipse menu. My purpose here is to highlight some of the menu items that are, if not exactly unique to Eclipse, at least not often found in common applications.

Starting with the File menu, there's an item that determines how the editor treats end of line. There are three conventions in the computing world for how a line of text is terminated. Sadly, there are even situations where these conventions are mutually incompatible. Windows programs terminate a text line with both the carriage return and line feed characters. Unix and Linux use only the line feed character and I believe the Mac OS uses only the carriage return. So you can choose the line termination policy to match the environment you're working in.

There are facilities to import resources, that is, files, into a project and to export an entire project as an archive file. The File menu also has an entry to switch workspaces, which, as I mentioned earlier, effectively restarts Eclipse.

The Refactor menu in CDT has expanded substantially in recent releases and I have to admit I haven't really done anything with it.

The Search menu has some interesting features. The first entry brings up a tabbed dialog box that offers several different ways of searching. The C/C++ Search tab lets you restrict a search to specific C/C++ elements and/or specific occurrences. The File Search tab implements the more common form of searching for a text string in a set of files. Personally, I'm not clear on the purpose of the Task Search tab. It appears to have something to do with Eclipse development and bug tracking.

Slide 15

The Edit menu has a number of interesting features. Incremental Find Next and Previous incrementally jumps to the next or previous match as you type the search expression. Add Bookmark places a so-called bookmark on the line where the cursor is. This makes it easy to jump to specific locations in a group of files. Add Task adds a task at the cursor location.

Word Completion supposedly attempts to complete the word currently being typed. I haven't had much success with that. More helpful is Content Assist. This opens a dialog at the current cursor location to offer assistance in the form of proposals and templates. Add a new line to hello.c after the puts statement and type "sw". Now select content assist. In this case, there's probably only one thing you mean and that's a switch statement, so Content Assist simply completes the word for you.

Quick Fix offers suggestions on correcting certain errors when the cursor is on a line that has an error. For the most part I don't find it terribly helpful.

Parameter Hints is kind of cool. Type the name of a library function, fopen for example, and the opening left parenthesis. Note incidentally that the editor automatically adds the closing right parenthesis. Now select Parameter

Hints and you'll get the parameter portion of the function's declaration from the corresponding header file. Note that the appropriate header file must be included for this to work.

Format reformats the source file to match the currently selected coding style. We'll talk more about coding styles when we come to configuring Eclipse.

Slide 16

The Navigate menu offers a number of useful ways to navigate around a large project. Open Declaration opens the file containing the declaration of the selected object. Try it with puts().

Open Type Hierarchy opens a type hierarchy view provided the selected object resolves to a defined type.

Open Call Hierarchy displays the Call Hierarchy view for the selected function. The Call Hierarchy can show what functions call this function and/or what functions this function calls. Try it with main and click the Show Callees icon. Needless to say, this isn't terribly interesting or useful in our trivial example here, but it could be very helpful in a large, multi-file project.

The Include Browser is an interesting view. It displays a hierarchy of all the included header files. This shows at a glance the header files that are included by the header files that you've included.

Open Resource brings up a dialog that lets you easily select any resource in the workspace to open. This encompasses all projects in the workspace.

Last Edit Location moves the cursor to the line that was most recently edited.

Slide 17

We're going to skip over the Run menu for the moment. We'll come back to it when we get into debugging.

The Project menu, perhaps not surprisingly, is concerned with managing projects. You can open or close a project. When you close a project, any resources that are currently open are closed and the project's hierarchy in the Project Explorer view is collapsed.

There are a number of ways to build one or more projects. Build All completely rebuilds all projects in the workspace. Build Project completely rebuilds the project selected in the Project Explorer view. Build Configurations lets you select and manage build configurations, of which there are normally two: Debug and Release. Build All and Build Project build the currently active configuration, which in most cases is Debug.

Clean cleans either all projects or selected projects as determined by a dialog. The selected projects are then rebuilt.

By default, the only target that Eclipse knows about is All. Make Target allows you to create and build other make targets. We'll see this in detail next week when we create a project that has its own makefile.

Properties opens a rather extensive dialog that lets you configure a wide range of options for the project. Among other things, you can set options for the build tools, the compiler and linker, and so on.

Slide 18

The Window menu offers a number of options for managing the Eclipse user interface. You can open a new window, which is essentially a second copy of Eclipse. This is what allows you to have two or more perspectives visible simultaneously.

You can display additional views with the Show View menu. This lists views that are commonly associated with the currently visible perspective, but you can always select Other to see a complete list of views.

You can open or close perspectives and even close all perspectives, which leaves you with a blank screen and just the Open Perspective icon.

Customize Perspective lets you change the contents of menus and the tool bar. Having created a custom perspective, you can save it. Reset Perspective returns the perspective to its original layout. Navigation offers another way to navigate around a large project.

Finally, Preferences brings up a dialog box where you can configure Eclipse to your liking. There are a lot of configuration options. I won't even try to describe them all. Browse through the various categories and see what's there.

One interesting option is Code Style that was mentioned briefly in slide 15 on the edit menu. The CDT text editor incorporates "smart typing" features that include things like auto-indentation and formatting that are controlled by the Code Style selection. Select Window > Preferences and expand the C/C++ category.

Expand the C/C++ category and select Code Style > Formatter. There are four built-in styles that differ primarily in the location and indentation of opening and closing braces. The default is K&R where the opening brace is on the same line as the expression or key word that introduces the block. This seems to be the preferred style among Linux programmers.

For what it's worth, my personal preference is BSD/Allman where the opening brace is on the next line and indented to line up with the introductory expression. There are options to edit the built-in styles, import a style, and create a completely new one.

Slide 19

The final entry on the menu is Help. Welcome brings up the welcome screen that you saw the first time you started Eclipse. This gives you a chance to try out the examples and tutorials. Help Contents opens a new window with a navigation pane on the left that lets you browse through the user's guides. The documentation is for the most part well-written and genuinely helpful.

Search is one of several Help actions that bring up a Help view in the current perspective. Use this to search the help documentation. Dynamic Help also brings up the Help view with content that changes as you click different elements of Eclipse.

Key Assist pops up a window that lists all of the shortcut keys.

Software Updates provides a neat mechanism to find and install new Eclipse features as well as update currently installed software. There are preferences to set up an automatic update schedule.

Slide 20

Again, like any good windowing application, Eclipse offers several different ways to invoke its functionality. In addition to the menus, many Eclipse actions are available from the toolbar just below the menu bar. The actions available in the toolbar will change depending on which perspective is visible and which view has the focus.

In addition to that, just about every object has a *Context menu* accessed by right-clicking on the object. The Context menu includes actions derived from the other menus that are commonly performed on the selected object.

Right-click on the project name hello in the Project Explorer view to see a typical context menu. Now right-click on the marker bar in the editor to see a totally different context menu.

Slide 21

Let's try something just a little more ambitious that will get us into debugging. We're going to create a simple record sorting program that will sort a data file either by name or ID number. And to keep it really simple, we'll replace spaces in the name field with underscores.

The files for this project aren't on the CD as they're not part of the Embedded Linux Learning Kit. So download and untar record_sort.tar.gz from the Week 3 folder.

Create a new C project and name it record sort. The project is currently empty. Eclipse tried to build it, but of course there's nothing to build. Take a look at the record_sort/ directory under the workspace. Right now it contains just two files, `.cproject` and `.project`, both of which are XML code describing the project. `.project` provides configuration information to the base Eclipse platform while the more extensive `.cproject` file provides information to CDT. It's not necessary to understand the contents of these files, but it is useful to know they exist.

Slide 22

There are basically two ways to get content into a project. You can import existing files into the project or you can create new ones. Follow the instructions under the first bullet point here to import the files you downloaded.

There's one file missing: record_sort.c, the main function. The contents of record_sort.c are in the file record_sort.pdf. The idea here is to give you some practice with the features of the Eclipse editor. The second bullet point here provides instructions for creating a new file, which automatically opens in the editor. Note that it has a simple header comment.

So type the contents of record_sort.pdf into the new file trying out the various content assist features of the editor. Yes, you could "cheat" and just copy and paste the contents, but that defeats the purpose, doesn't it?

Save the file then build the project. Note by the way that all files that are created in, or imported into, a project automatically become a part of it and are built and linked into the final executable. Ah, it fails! A couple of simple compile time errors have been built into sort_utils.c to give you some practice in tracking them down in Eclipse. Correct the errors and rebuild.

Slide 23

To debug record_sort, we could just select Run > Debug or click the Debug icon in the toolbar. Run > Debug automatically creates a "launch configuration" and switches to the Debug perspective. That doesn't work in this case because we need to specify an argument to the program, the name of the file we want to sort.

Selecting Run > Debug Configurations brings up a dialog box for managing what Eclipse calls launch configurations. The "new" icon is at the left end of the toolbar above the navigation pane. You'll need to give it a name, specify a project and an application, and enter the program argument as shown here.

Take a look at some of the other tabs to see what's there. When you click Debug, Eclipse switches to the Debug perspective.

Slide 24

Eclipse provides a graphical wrapper, or front end, to GDB, the GNU debugger. The Debug view in the upper left is essentially a call stack depiction. Right now it shows a single thread that is suspended in the main function. The Eclipse toolbar now shows a set of icons that you'll be using a lot as you step through your program. Step Over means step over function calls. Step Into means step into a function and Step Return means go back to where this function was called.

Below the Debug view is an editor window with record_sort.c open. The first line of main is highlighted and there's a green arrow in the marker bar that identifies the current program counter location. The program stopped at the first executable line of main because that option is selected in the launch configuration.

In the upper right hand tabbed pane are two important views: Variables and Breakpoints. By default, Variables only shows the local variables of the function selected in the Debug view. There's an option to add global variables.

Click Step Over to get to the call to read_file(). Note that sort is highlighted in the variables view because its value changed. Now click Step Into. This opens sort_utils.c and puts the program counter at the first line of read_file(). This function reads a file and creates an array of data records.

Slide 25

At this point it would probably be useful to set a breakpoint in the for loop and watch the function read the file. It would also be useful to turn on line numbers because I'll be referring to them from time to time. Right-click on the marker bar and click Show Line Numbers. Now right-click the marker bar at line 33 and select Toggle Breakpoint. A green circle with a check mark shows up in the marker. This identifies an enabled breakpoint. Take a look at the breakpoints view and you'll see the new breakpoint listed there.

Click Resume. The program executes to the breakpoint. You can see the current value of any local variable or function argument by just rolling your cursor over it. Click Resume a couple more times and note that i is incrementing properly.

At this point it might be useful to set a breakpoint inside the if statement to see what happens when we reach the end of file. Set a breakpoint at line 36 and disable the breakpoint at line 33. There are two ways to disable a breakpoint. Right-click on the symbol in the marker bar and select Disable Breakpoint or open the Breakpoints view and click the check box next the breakpoint entry. The circle icon turns white to indicate it's disabled.

Click Resume and the program executes to line 36 where you'll see that we have read in 11 records. Step down to line 39. Note that the size argument to read_file is a pointer. When you look at size in the Variables view, you see the address value. Click on the triangle-shaped arrow next to size to expand its entry and you'll see the value that was just stored there.

Click Step Return to get back to main. You might want to step into sort_name() and set some strategic breakpoints to watch how it executes.

Slide 26

Breakpoints have a number of interesting properties that can help you focus in on a problem. In the Breakpoints view, right-click one of the entries and select Properties. You can attach one or more *actions* to a breakpoint. A breakpoint can log a message, play a sound, automatically resume after a specified delay, or execute some external tool. Suppose the particular code you're debugging only executes every 10 or 15 minutes. You could configure a breakpoint to play a sound and you could go off and do something else until you hear the sound that says the breakpoint has been hit.

You can have a breakpoint log some information, which can be in the form of an expression, and then automatically resume. This way you can develop an execution history.

Click Common. I consider that a poor name for this particular feature, but there it is. This dialog lets you set conditions on when the breakpoint is taken. You can specify a condition that evaluates to a Boolean value and the breakpoint is ignored until the condition evaluates to true. Or you can set an ignore count, which is the number of times the breakpoint will be ignored before being taken. Both of these features help you focus in on the problem by eliminating unnecessary breakpoints.

Filtering lets you restrict the breakpoint to specific targets and threads. I haven't played with that particular feature.

When you're finished playing with breakpoints and studying the code, click Resume to let the program complete. What's this? Only one record shows up in the Console output when we know that we read 11 records. And if you look at datafile, you'll see that it actually contains 12 records Yep, there are a couple of runtime bugs somewhere in sort_utils.c. So for more practice with the debugger, find and fix those bugs.

Slide 27

Let's review what we've done. We've looked at the background of Eclipse and touched on why it's important for embedded developers. Then we examined the elements of Eclipse to see how it all hangs together.

Slide 28

To make it all real, we ran through a real Eclipse project from creation through debugging.

That's it for this week. I strongly encourage you to play around with Eclipse some more. Browse through the documentation, maybe try some of the tutorials. Play with the various breakpoint properties.

Next week we'll use Eclipse to build and debug programs on the target board. Be sure to take the quiz this week.