Working with User-Defined Projects Lab

© 2013 Wind River Systems, Inc

WIND RIVER

Working with User-Defined Projects Lab

Objective

In this lab you will learn more advanced techniques for managing projects within Workbench. You will have the opportunity to work with one of the two fundamental build types provided by Workbench, and learn its unique advantages.

NOTE: This lab should take approximately 30 minutes.

Lab Overview

This lab focuses on the user-defined build type provided by the Workbench build system. In a user-defined project, the user is wholly responsible for managing the project's build infrastructure — Workbench merely calls out to this user-managed framework as needed.

While implementing and maintaining a build system may sound like a bad deal, there are many cases where this type of build is beneficial. Consider the following projects:

- the Linux kernel
- Busybox
- *any software package driven by* **autoconf** (that is, driven by a **configure** *script*)
- any project that includes a Makefile

Chances are, you've worked with more than one of these in the past. These all have one thing in common: they come with their own build system. If Workbench didn't work with user-defined build systems, you would have to spend time mapping flexible managed build constructs to duplicate functionality already provided by the project's existing build infrastructure.

User-defined projects let you leverage work that has already been done, and at the same time enjoy all the benefits Workbench has to offer.

In this lab, you will work with the source in the archive /Labs/workbench/vitetris-0.57.tar.gz. The archive contains the source code for an open-source clone of the popular game, Tetris, which renders on a standard text-based console. The archive is structured as follows:

- A subdirectory called **src**, which functions as the root for all source code, and contains a main **Makefile** that drives the build of the entire application
- Subdirectories within **src** which implement various subsystems (input, rendering, etc). Each subsystem is driven by its own makefile, and is built as a self-contained static library.

This structure is fairly representative of projects that are deeply entrenched in a build system of their own. Consider, for a moment, how you might import this project into your workspace using a flexible managed build.

In situations like these, the value of user-defined projects becomes apparent — if the software you're working with comes with a build infrastructure that works, why not use it?

Creating and Populating a User-Defined Project

- Begin by creating a new native user-defined project, similar to the way you created a native application project in the lab, *Getting Started with Workbench*. In the New Wind River Workbench Project wizard, select Host OS (Native Development).
- 2. Click Next.



- 3. In the **Build type** field, select **User-Defined**.
- 4. Click Next.

New Wind River Workbench Project		
Build Type		
Select the build type for	r the project.	
Build type: User-Defin	ned 🗘	
Description: Creates a p	roject, which allows managing and launching user-defined	
Setup information		
Resulting project type:	User-Defined Project	
Uses build tool:	None	
?	< Back Next > Cancel Finish	

- 5. Name this project vitetris and click Finish to create the project.
- 6. Now, import the source code into the project. In the **Project Explorer** view rightclick on the new **vitetris** project.
- 7. In the context menu, select Import to import the source code into the project.



- 8. In the **Import** dialog, select **General > Archive File**.
- 9. Click Next.



10. Click Browse to locate the archive from which Workbench will import resources.



- 11. In the **Import from Archive File** dialog, navigate to the **/Labs/workbench** directory, and select **vitetris-0.57.tar.gz**.
- 12. Click **OK**.



13. In the **Import** dialog, Workbench presents a view of the files in the selected archive. Verify that **all** files in the archive are selected (not just source files), and click **Finish**.

Im	port		
Archive file Import the contents of an archive file in zip or tar format from the local file system.			
From <u>a</u> rchive file: /Labs/workbench/vitetris-0.57.1	ar.gz		
▽ ☑ /	🗹 🗎 changes.txt		
🕨 🗹 🗁 vitetris-0.57	🗹 🗋 config.mk 📃		
	🗹 🗎 configure		
	🗹 🍠 icon.ico		
	🗹 🗎 icon.rc		
	🗹 🗎 INSTALL		
Filter Types Select All Deselect All			
Into fo <u>l</u> der: vitetris	Bro <u>w</u> se		
Overwrite existing resources without warning			
< Back	Next > Cancel Finish		

14. In the **Project Explorer** view, review the contents of your new project. Notice that:

- The imported source appears in the **vitetris-0.57** subdirectory within your project (mirroring the structure of the files within the source archive)
- Workbench has created a default **Makefile** in the root directory of the project
- There are no build targets, like you would see for a flexible managed build. This is because the build targets for a user-defined project are defined and managed differently than for a flexible managed build.



Building the Project

15. Build the project — in the **Project Explorer** view, right-click on the project and select **Build Project** from the context menu.



Not much happens. Why? Examine the contents of **Makefile** to investigate the cause. all :

```
@echo "make: built targets of `pwd`"
clean :
  @echo "make: removing targets and objects of `pwd`"
```

When creating a new user-defined project, Workbench supplies a very basic **Makefile** that essentially does nothing – it's meant to be populated by you.

There are different solutions to the problem:

- Rewrite the **Makefile** to call into the **Makefile** provided by the **vitetris-0.57** directory.
- Shift the contents of the **viteris-0.57** directory up one directory level, thereby overwriting the Workbench-supplied **Makefile**.
- Tell Workbench to call into the **Makefile** found in the **vitetris-0.57** directory.

For the purpose of this exercise, consider the last option, for the first two will not teach you anything about Workbench project management.

16. Access the **Makefile** options via the build properties for the project; in the **Project Explorer** view, right-click on the **vitetris** project and select **Properties** from the context menu.

Designet Fundament M	Jource	/
Project Explorer 🛛	🚵 Import	
	⊿ Exp <u>o</u> rt	
Vitetris (Host OS	Open Wind River Workbench 3.3.4 Development Shell	
▷ mild loci Lad	🖏 Re <u>f</u> resh	F5
👂 🗁 vitetris-0.57	Clo <u>s</u> e Project	
🗴 .cproject	lose Unrelated Projects	>
🗴 .project	Build Project	<pre>Ctrl+B Ctrl+P</pre>
🖹 .wrproject	🛃 Rebuild Project	
lo Makefile	Build Options	>
	Run Linux Application Process	
	Debug Linux Application Process	
< III	Index	>
릚 Remote Systems 없	Run Remote Application	
	🏇 Debug Remote Application	
▼ 🖻 Local	🛛 <u>R</u> un As	>
Wind River Reg	拳 <u>D</u> ebug As	>
👂 🔩 Local Files	<u>P</u> rofile As	>
🛃 Local Shells	Compare With	>
👂 🚵 LinuxHost_wruse	Restore from Local History	
	Configure	>
< 1 1	≫ Run <u>C</u> /C++ code Analysis	
] □◆]	P <u>r</u> operties	Alt+Enter
Mill		

- 17. In the **Properties for vitetris** dialog, in the navigation panel on the left, select **Build Properties**.
- 18. Click the **Build Support** tab to access build settings.
- 19. In the **Build command** field, add the option **-C vitetris-0.57**; this option will instruct **make** to use the **Makefile** in the **vitetris-0.57** subdirectory rather than the one contained in the top-level project directory.
- 20. Click Apply.
- 21. Click OK.



- 22. Attempt to build the project again, as you did in the first step of this section. Use the **Build Project** option, not the **Rebuild Project**. This time, your build will succeed.
- 23. Now, attempt to rebuild the project using the **Rebuild Project** option in the context menu.



The build fails this time, with the following error. Why?

make: *** No rule to make target `all'. Stop.

The error indicates that the target Workbench invokes (**all**) cannot be found in the **Makefile**. To see why Workbench is invoking this target, you will need to revisit the properties for the **vitetris** project as you did earlier in this section.

- 24. In the **Properties for vitetris** dialog, in the navigation panel on the left, select **Build Properties**, as you did before.
- 25. This time, click the **Targets** tab to access build target settings.
- 26. Focus your attention on the **Rebuild Project rule(s)** field. Notice that this is preset to a value of "**clean all**". In order for this to work, the Makefile must have both targets defined. Refer back to the **Makefile** in the **vitetris-0.57** directory to see which targets are defined, and you will find that **all** is not defined. Replace **all** in the **Rebuild Project rule(s)** field with an equivalent target that can be found in the **Makefile**.
- 27. When you've made your change, click Apply and OK.

Properties for vitetris			
type filter text 🔏	Build Properties	, , , , , , , , , , , , , , , , , , ,	
Resource	Specify all build properties.		
Binary Parser	Build Support Targets Specs & Variables		
Build Properties	Make sules		
Builders]	
▷ C/C++ General	Build Project rule:	le:	
Code Coverage Analy	Clean Project rule: Clean Clean Folder ru	ule: clean	
Project Info	Rebuild Project rule(s): Itlean all		
Project References			
Run/Debug Settings	Cuttom build targets:		
Task Tags			
Validation	Name Type Makerule / Jomi Refresh settings	New	
		Edit	
		Rename	
		Copy	
		Delete	
	24 25 26 27		
	Note: These custom build targets are accessible on any res in the "Build Options" context sub-menu.	ource and appear	
< III >	Restore Defau	Apply	
?	Cancel	ОК	
Ū.	Current		

28. Test your changes by rebuilding the project.

Adding Build Targets

You have seen how to edit the default targets that Workbench invokes in a user-defined build. Sometimes, you must add new targets in order to access additional functionality provided by a project's **Makefile**. In this section, you will learn how to do this.

29. Add the following code to the end of the Makefile in the **vitetris-0.57** directory, to allow you to play the game via a build target:

play:<Space>build<Enter><Tab>\$(TERM) ./\$(PROGNAME)<Enter>

NOTE: Makefiles are notoriously picky about line formatting and spacing. Type the lines exactly as shown, paying particular attention to the **Tab** in the second line.

When invoked, this target (after ensuring that the project is built) will start the built program in a shell defined by the **TERM** environment variable (typically this is set to something like **xterm**).

- 30. Now, add this new build target to the vocabulary for this project. As you did in the previous section, reopen the properties for the **vitetris** project and advance to the **Targets** tab.
- 31. In the Custom build targets area, click New...

Properties for vitetris			
type filter text 🔏	Build Properties $\Leftrightarrow \lor \Leftrightarrow \lor \checkmark \checkmark$		
Resource	Specify all build properties.		
Binary Parser	Build Support Targets Spece & Variables		
Build Properties	Build Support and in 900 with Specs a variables		
Builders	Make rules		
▷ C/C++ General	Build Project rule: Build Folder rule:		
Code Coverage Analy	Clean Project rule: clean 30 31 Folder rule: clean		
Project Info	Rebuild Project rule(s): clean default		
Project References			
Run/Debug Settings			
Task Tags	Custom build targets:		
Validation	Name Type Make rule / Com Refresh settings New		
	Edit		
	Rename		

- 32. In the **New Custom Build Target** dialog, in both the **Name** and **Make rule or command** fields, enter **play**.
- 33. Ensure that the **Type** field is set to **Rule**.
- 34. Click OK.



35. In the **Properties for vitetris** dialog, in the **Custom build targets** panel, notice the addition of your new target.

Properties for vitetris			
ty	pe filter text 🛛 🦧	Build Properties	⇔- ⇔- ◄
Þ	Resource	Specify all build properties.	
_	Binary Parser	■ Build Support 🖫 Targets 🐼 Specs 💲 Variables	
Þ	Build Properties	Mala milas]
	Builders	Make rules	
Þ	C/C++ General	Build Project rule: Build Folder rule:	
	Code Coverage Analy	Clean Project rule: clean Clean Folder rule:	clean
	Project Info	Rebuild Project rule(s): clean default	
	Project References	S3	
	Run/Debug Settings		
	Task Tags	Custom build targets:	
Þ	Validation	Name Type Make rule / Comi Refresh settings	New
		play Rule play Use project setting	Edit
			Rename

- 36. Now ensure that Workbench sets a value for the **TERM** environment variable. Click on the **Variables** tab in the **Properties for vitetris** dialog.
- 37. Click New...

Properties for vitetris			
type filter text 🛛 🧳	Build Properties	⇔~ ⇔~ ▼	
Resource	Specify all build properties.		
Binary Parser	Build Support Targets Specs Variables		
Build Properties			
Builders	Name Value	New	
▷ C/C++ General		Edit	
Code Coverage Analy			
Project Info			
Project References	36 37	Rename	
Run/Debug Settings		Сору	
Task Tags		Delete	
Validation		Detete,	

- 38. In the New Build Macro dialog, in the Name field, enter a value of TERM.
- 39. In the Value field, specify xterm.
- 40. Click **OK**.



41. In the **Properties for vitetris** dialog, click **Apply** and **OK**.



42. Now, right click on the **vitetris** project in the **Project Explorer** view.



43. In the context menu, select **Build Options > Play**.

44. Enjoy!



This concludes the lab. Do not proceed.