Turn-Key

Image building
Integrating connectivity, QoS, and security

Ying-Dar Lin
Agenda

● Backgrounds
  ● Turn-key, uCLinux, uClibc
  ● ToolChain, buildroot system
  ● CVS, SVN

● Introduction of the Wall Project

● Issues
  ● Procedures for adding a package into the turn-key
  ● Accelerate the turn-key building procedure

● Mini-project

● Project
Backgrounds

- Turn-key?
- uCLinux, uClibc, ToolChain?
- How to use the uClibc Buildroot system
- How to use the CVS Server/Client
What is a Turn-key?

- Pre-built computer "packages"
- Perform a certain type of task
- Everything needed is put together in
A turn-key example: A Linux-based System

**Hardware Platform:**
*Arm, i386, mips, PowerPC*

**toolchain**
- Library: *uClibc*
  - GNU binutils
gcc compiler

**kernel:** *uClinux*

**Root image**
- App1
- App2
- App3
- App4
- GUI
  - Busy Box

**Buildroot System**

**uClibc Buildroot System**

2009/2/18
Ying-Dar Lin
uClibc

- A C library for embedded Linux
- Much smaller than the GNU C Library (glibc)
- Compatible with glibc
- For MMU-less systems (µClinux)
- Also for standard Linux
- Supports shared libraries and threading
- Closed source commercial applications
A derivative of Linux kernel
for microcontrollers without Memory Management Units (MMUs).
Linux kernel releases for 2.0 2.4 and 2.6
http://www.uclinux.org/
uCsimm

- A microcontroller module built specifically for the uClinux Operating System
- An inch high, with a standard 30-pin SIMM form factor
- A Motorola DragonBall 68EZ328 processor
- 2 MB of FLASH and 8Mb of DRAM
- A 10Base-T ethernet and RS 232 high-speed serial
- [http://www.uclinux.org/ucsimm/](http://www.uclinux.org/ucsimm/)
BusyBox

- Combines tiny versions of common UNIX utilities into a single small executable
- Fewer options than their full-featured GNU cousins
- Size-optimization and limited resources
- Extremely modular and easy to customize
- http://busybox.net/about.html
BusyBox v1.1.0 (2006.03.08-23:51+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# help

Built-in commands:
________________________
. : alias bg break cd chdir continue eval exec exit export false fg hash help jobs kill let local pwd read readonly return set shift times trap true type ulimit umask unalias unset wait

#
**uClibc BuildRoot System**

- Select the packages you want
- Build your own
  - uClibc-based root file system
  - Or Development system
- Cross-Platform: arm, i386, m68k, mips, ...
- A time-consuming procedure
- [http://buildroot.uclibc.org/buildroot.html](http://buildroot.uclibc.org/buildroot.html)
- Pre-compiled devel. systems are available
Architecture in BuildRoot Sys

Dir: Package
- linux.mk
- squid.mk
- Busybox.mk
- customize.mk
- Yourown.mk
- ....

DIR: DL
- Downloaded packages

Dir: build_ARCH
- DIR: root
- rootfs.ARCH

Makefile
# OS

TARGETS := linux

# The default minimal set
TARGETS += busybox tinylogin

# a full development system
TARGETS += ed file gawk tar grep bzip2

# your own packages configuration start
TARGETS += libgmp freeswan
TARGETS += iproute2 iptables
TARGETS += dnsmasq
TARGETS += tripwire
TARGETS += zebra

# copy configuration
TARGETS += customize

# build image
TARGETS += ext2root
make menuconfig:
main menu

Buildroot Configuration
Arrow keys navigate the menu. <Enter> selects submenus -->. Highlighted letters are hotkeys. Pressing <Y> selectes a feature, while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search.
Legend: [*] feature is selected [ ] feature is excluded

- Target Architecture (i386) --->
- Target Architecture Variant (i686) --->
- Build options --->
- Toolchain Options --->
- Package Selection for the target --->
- Target Options --->
- Board Support Options --->

Load an Alternate Configuration File
v(*)

<Select>  < Exit >  < Help >
make menuconfig:
Package Selection

Package Selection for the target
Arrow keys navigate the menu. <Enter> selects submenus -->. Highlighted letters are hotkeys. Pressing <Y> selectes a feature, while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for Help. </> for Search.
Legend: [*] feature is selected [ ] feature is excluded

[ ] BusyBox
[ ] Use the daily snapshot of BusyBox?
[ ] Install symlinks for BusyBox applets
(package/busybox/busybox.config) BusyBox configuration file to use?
--- The minimum needed to build a uClibc development system
[ ] bash
[ ] bzip2
[ ] coreutils
[ ] "(*)

<Select> < Exit > < Help >
make menuconfig:
Platform Selection

Buildroot Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->

Target Architecture

Use the arrow keys to navigate this window or press the hotkey of the item you wish to select followed by the <SPACE BAR>. Press <?> for additional information about this option.

^(-)
(X) arm
( ) armeb
( ) cris
( ) i386
( ) m68k
( ) mips
( ) v(+)
Basic Functions in Source Versions Control System

- **Checkout:**
  - Distribute source codes

- **Version:**
  - File history, comparison, rollback

- **Update/Commit:**
  - Team Work

- **Branch:**
  - An experimental revision
  - Different platforms/purposes
Concurrent Versions System (CVS)

CVS Server

- Source Codes
- login
- logout
- import
- checkout
- update
- add
- commit

CVS Client

- Initial Version
- Working Version

codes/documents
Cases for using CVS (I):
Download Open Source Packages

- Only need a CVS client

Operating procedures:
- Set `CVSROOT`: to indicate where the server locates
  \[ CVSROOT = \text{pserver}: \text{username}@140.113.xx.xx:/CVSROOT \]
- Login
- Checkout: download all codes
- Update: update the latest codes
Cases for using CVS (II): Maintain Your Own Packages

- Need the CVS server and client

- Operating procedures:
  - Create Repository, where you all sources are stored
  - Set CVSROOT
  - Login
  - Import
  - Checkout
  - Commit, Update, Add
Tips for using CVS

- CVS is also available on MS Windows
  - e.g. Server: CVSNT, Client: WinCVS
  - http://www.wincvs.org/
- Login into the CVS: not another shell
- CVSROOT is an environment variable
  - The configuring is shell-dependent
  - TCSH: `setenv CVSROOT :pserver:username@140.113.88.149:/CVSROOT`
  - BASH: `export CVSROOT=:pserver:username@140.113.88.149:/CVSROOT`
- “cvs add” does not copy files to the server.
  “cvs commit” is needed after “add”
- “cvs add *dirname*” does not add all files
  - you have to cd to this directory
  - keyin “cvs add *”
- http://www.nongnu.org/cvs/
Screenshot of WinCVS client

Python 2.4.1 (#65, Mar 30 2005, 09:13:57) [MSC v.1310 32 bit (Intel)] on win32
Tk is available, Tk-macros are enabled
TCL is available, shell is enabled : help (select and press enter)
Screenshot of WinCVS client
**SubVersioN (SVN)**

- Most current CVS features
- Directories, renames, and file meta-data are versioned
- Commits are truly atomic
- Choice of database or plain-file repository implementations
- Efficient handling of binary files
- http://subversion.tigris.org/
Wall Project

十機一體開道器於Linux平台

Security and QoS gateway

From 7-in-1/NetBSD To N-in-1/Linux
Evolution of Wall

- **7-in-1** (NetBSD)
  - Solving problems on *TCP/IP layer*
  - NAT, Firewall, VPN, Router, IDS, CF, BW management.

- **10-in-1** (NetBSD)
  - *Content-aware*
  - Anti-Virus, Anti-Spam, CF/Keyword
  - Reducing System Overhead

- **N-in-1** (**Linux**)
  - *Easy to add new module*
  - UPnP, APP Firewall, SSL-VPN,
  - Transparent Proxy, MSN log, POP3 proxy, Log rotation
# Specification

| Connection         | LAN, DMZ, WAN, DHCP, DNS relay, Dynamic DNS  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><em>Link load balance, Bridge mode</em></td>
</tr>
<tr>
<td>Security</td>
<td>IPSEC, PPTP, L2TP, SSL-VPN</td>
</tr>
<tr>
<td>Firewall</td>
<td>NAT, Firewall, APP firewall, UPNP, <em>Traffic profiling</em></td>
</tr>
<tr>
<td>Mail</td>
<td>Anti-Spam, Anti-Virus, <em>POP3 proxy</em></td>
</tr>
<tr>
<td>Web</td>
<td>URL, URL keyword, content keyword, <em>Transparent Proxy</em></td>
</tr>
<tr>
<td>IM</td>
<td>MSN log</td>
</tr>
<tr>
<td>BW Control</td>
<td>TC</td>
</tr>
<tr>
<td>Management</td>
<td>Web, SSL, FTP, <em>Log rotation, CRON</em></td>
</tr>
<tr>
<td>Platform</td>
<td>I386, <em>IXP (simple version)</em></td>
</tr>
</tbody>
</table>
Configuring Wall : Anti-Spam

Mail - Anti Spam

- Enable Anti Spam
- Score Threshold
  - Any mail above this threshold is marked as spam
    - Low Threshold (5.0, default)
    - Medium Threshold (7.5)
    - High Threshold (10.0)
    - Manual:
  - Any mail above this threshold will be rejected
    - Low Threshold (5.0, default)
    - Medium Threshold (7.5)
    - High Threshold (10.0)
    - Manual:

- Don't rewrite subject of spam mail
- Rewrite subject of spam mail using text [**SPAM**]
LAB Network Topology

Server (Media, Web)
140.113.88.10

Hacker
140.113.88.20

Server (Mail)
140.113.88.30

WAN
(140.113.88.*)

Private LAN C
(192.168.3.*)

Private LAN A
(192.168.2.*)

Private LAN B
(192.168.1.*)
Issues

- Adding a package into the turn-key
- Accelerating the building procedure
Procedures for Adding a Package

A. Select the open source package
B. Compile it with uClibc (Built in the uClibc development system)
C. Checkout from CVS and Write the xxxx.mk
D. Trim the code size
E. Set the configuration of the package
F. Build and Install the final image
G. Test its operation
H. Add and Commit all files into CVS
B. Compile the package in the development system (I)

1. Prepare development system
   - Download the pre-complied system
   - or Build your own system

2. Mount the development system
   - mkdir rootfs
   - su root
   - mount -o loop rootfs.ARCH.FS rootfs

3. Download the selected package

4. Unpack into a directory under rootfs
   e.g. rootfs/usr/local/src
B. Compile the package in the development system (II)

5. Switch into the development system
   - chroot rootfs bin/su –
   - using the bin utils and libraries in the image rootfs.ARCH.FS

6. configure, make, install the package

7. Test and Debug the package

8. Handle all the errors
C. How to Write the xxx.mk?

1. xxx.mk tells the buildroot how to make your package

2. Study other xxxx.mk files.
   - http://www.uclibc.org/cgi-bin/cvsweb/buildroot/make/
   - openssh.mk is a good sample.

3. Five parts are included in xxx.mk.
   - *unpacked*: download and unpack
   - *configured*: set the options for make
   - *strip*: downsize the code
   - *make* and *install*
   - *clean*: clean the built results or intermediate files
## Important Variables Referred in ‘xxxxx.mk’ (I)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUILD_DIR</td>
<td>All downloaded packages are unzipped in this working dir.</td>
</tr>
<tr>
<td>TARGET_DIR</td>
<td>after compiling, the results should be copied to this directory.</td>
</tr>
<tr>
<td>SOURCE_DIR</td>
<td>All patch files are put in this dir.</td>
</tr>
<tr>
<td>DL_DIR</td>
<td>All downloaded package are put in this dir</td>
</tr>
<tr>
<td>STAGING_DIR</td>
<td>Libraries and header files in this dir can be used as building</td>
</tr>
<tr>
<td>TARGET_CROSS</td>
<td>The cross compiler</td>
</tr>
<tr>
<td>TARGET_CC</td>
<td>$\text{(TARGET_CROSS)}\text{gcc}$</td>
</tr>
<tr>
<td>STRIP</td>
<td>The command to downsize the executing code</td>
</tr>
</tbody>
</table>

All the variables are defined in Makefile
Important Variables
Referred in ‘xxxxx.mk’ (II)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXXXX_SITE</td>
<td>Tell buildroot where to download the package</td>
</tr>
<tr>
<td>XXXXX_DIR</td>
<td>Tell buildroot where to be the working dir</td>
</tr>
<tr>
<td>XXXXX_SOURCE</td>
<td>the tar file name of the package</td>
</tr>
<tr>
<td>XXXXX_PATCH</td>
<td>the path and filename of patch file (optional)</td>
</tr>
</tbody>
</table>

the four variables are defined in xxx.mk
**Tips for Writing xxx.mk**

- **Remember to add**
  ```
  $(TARGET_CONFIGURE_OPTS) \ 
  LD=$(TARGET_CROSS)gcc \ 
  CFLAGS="$(TARGET_CFLAGS)"
  ```
  in the ‘configured’ part and set
  ```
  CC=$(TARGET_CC)
  ```

- **If some necessary libraries are missing,**
  - build the missing libraries as you build a package,
  - then copy them into the dir specified by $(STAGING_DIR).

- **if you fail to cross-compile,**
  - chroot to the development image
  - build the image manually
  - pack to a tar file
  - in xxx.mk you just ask unpack the tar file
D. Trim the Image Size (I)

- Cut symbol information by STRIP
- Cut help documents
- Remove the static libraries and include files

```bash
-$(STRIP) --strip-unneeded $(SQUID_DIR)/src/squid
rm -rf $(TARGET_DIR)/man
rm -rf $(TARGET_DIR)/include/*.{h,cc}
rm -rf $(TARGET_DIR)/lib/*.{a,so}
```

A partial code of squid.mk
Downsize for

“squid-2.5.STABLES”

Original Size: 9668KB

⇒ 9660KB (make optimization and remove .note & .comment)
⇒ 9648KB (remove man file)
⇒ 6140KB (remove html doc)

• Squid program 5504KB
Downsize for gawk-3.1.2

- Original Size: 3172KB
  ➔ 3170KB (make optimization and remove .note & .comment)
  ➔ 3074KB (remove man files)
  ➔ 1890KB (remove info doc)
  ➔ 1386KB (remove example files)
## Size Distribution (I)

<table>
<thead>
<tr>
<th>Directory</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td>1MB</td>
</tr>
<tr>
<td>sbin</td>
<td>0.25MB</td>
</tr>
<tr>
<td>etc</td>
<td>2.9MB</td>
</tr>
<tr>
<td>tmp</td>
<td>0.018MB</td>
</tr>
<tr>
<td>home</td>
<td>0.006MB</td>
</tr>
<tr>
<td>share</td>
<td>1.1MB</td>
</tr>
<tr>
<td>local</td>
<td>0.011MB</td>
</tr>
<tr>
<td>www</td>
<td>3.0MB</td>
</tr>
<tr>
<td>lib</td>
<td>0.883MB</td>
</tr>
<tr>
<td>libexec</td>
<td>0.032MB</td>
</tr>
<tr>
<td>usr</td>
<td>128MB</td>
</tr>
</tbody>
</table>

Total: 139MB
Size Distribution (II)

Files and Dirs in the \textit{usr} dir

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{lib}</td>
<td>\textbf{31MB}</td>
<td>\textit{docs}</td>
</tr>
<tr>
<td>sbin</td>
<td>4.2MB</td>
<td>local</td>
</tr>
<tr>
<td>share</td>
<td>2.4MB</td>
<td>postfix</td>
</tr>
<tr>
<td>\textit{include}</td>
<td>\textbf{9.6MB}</td>
<td>bin</td>
</tr>
</tbody>
</table>

Before Downsizing: 139 MB
After Downsizing:
\begin{itemize}
  \item \textbf{128MB} (-11MB for doc)
  \item \textbf{118.4MB} (-9.6 MB for include)
  \item \textbf{102.6MB} (-15.8 MB for \texttt{*.a})
\end{itemize}
G. Build and Install the Image

To test your image, you can install the images into a hard disk and then boot on a i386 PC.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Run ‘make’ to compile and build the kernel and the final image.</td>
</tr>
<tr>
<td>2</td>
<td>fdisk a clean disk to 2 partitions one for kernel and one for rootfs.i386</td>
</tr>
<tr>
<td>3</td>
<td>format and mount the 2 partitions wall_kernel wall_image</td>
</tr>
<tr>
<td>4</td>
<td>mount imgfile rootfs.i386.ext2 mount -o loop rootfs.i386.ext2 xxxx</td>
</tr>
<tr>
<td>5</td>
<td>cp bzImage to wall_kernel/boot cp –af xxxx/* wall_image</td>
</tr>
<tr>
<td>7</td>
<td>install boot loader grub mkdir wall_kernel/boot/grub cp /boot/grub/* wall_kernel/boot/grub write menu.lst in boot/grub ln –s menu.lst grub.conf</td>
</tr>
</tbody>
</table>
H. Add and Commit all files

- Add the files and dirs into cvs/svn
  - Package packed file (dl/xxxx.gz)
    - cd dl
    - cvs add xxx.gz
  - Package make file (package/abcd/abcd.mk)
    - cd package
    - cvs add abcd
    - cd abcd
    - cvs add abcd.mk
  - Configuration files (package/customize/source)
- Commit
  - cd buildroot
  - cvs commit
Testing the installed Image

- Add a new Package
  - CVS clean Checkout
  - Building repeatedly (3.5 Hr once)
  - Run Demo Case
  - Test OK
    - Y: Check and Accept
    - N: PktFlow or Configure Error
      - Modify & Run Demo Case
      - CVS add and commit
**Bottlenecks for getting an workable system**

**START**

- Source in CVS
  - 5 mins

**Build**

- 4 hours

**Build Environment**

- K hours
- KN hours

**Install, Test, and Modify**

- 1 hour X 3 person
- 6 hours X 3 person Long-tail dist.

**TIME**

- 0.5 hours

**END**

- Workable System

**K=3**

- Downloading Time
- Kernel-Building Time
- Human-Delay
- Install Time
**Consumed Time in Building Image**

<table>
<thead>
<tr>
<th>Compiled Items</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>All packages</strong></td>
<td>154m 47.393s</td>
</tr>
<tr>
<td><strong>Kernel only</strong></td>
<td>68m 22.161s</td>
</tr>
<tr>
<td>Squid</td>
<td>2m 49.102s</td>
</tr>
<tr>
<td>Zlib opssl/ssh</td>
<td>10m 55.443s</td>
</tr>
<tr>
<td>p3scan (only tar)</td>
<td>8.04s</td>
</tr>
<tr>
<td>Executor</td>
<td>59.936s</td>
</tr>
<tr>
<td>Dansguardian</td>
<td>17.687s</td>
</tr>
<tr>
<td>gcc</td>
<td>2m 6.387s</td>
</tr>
<tr>
<td>Pureftpd</td>
<td>2m 7.61</td>
</tr>
</tbody>
</table>

IPC (CPU 700MHz/256MB)
### Packages not related to kernel

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Squid</td>
<td>Web proxy cache</td>
</tr>
<tr>
<td>Zlib</td>
<td>lossless data-compression library</td>
</tr>
<tr>
<td>Openssl</td>
<td>Toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols</td>
</tr>
<tr>
<td>Openssh</td>
<td>the SSH protocol suite of network connectivity tools</td>
</tr>
<tr>
<td>P3scan</td>
<td>full-transparent proxy-server for email clients</td>
</tr>
<tr>
<td>Gcc</td>
<td>the GNU Compiler Collection</td>
</tr>
<tr>
<td>Bridge tools</td>
<td>IEEE 802.1d ethernet bridging</td>
</tr>
<tr>
<td>Dansguardian</td>
<td>web content filter. It filters the actual content of pages based on many methods including phrase matching, PICS filtering and URL filtering.</td>
</tr>
<tr>
<td>Pureftpd</td>
<td>a free (BSD), secure, production-quality and standard-conformant FTP server.</td>
</tr>
</tbody>
</table>
## Packages related to kernel

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iptables</td>
<td>the user-space command line program used to configure the packet filtering ruleset and NAT</td>
</tr>
<tr>
<td>Freeswan</td>
<td>an implementation of IPSEC &amp; IKE for Linux</td>
</tr>
<tr>
<td>Hotplug</td>
<td>lets you plug in new devices and use them immediately</td>
</tr>
<tr>
<td>iproute2</td>
<td>a collection of utilities for controlling TCP / IP networking and Traffic Control</td>
</tr>
<tr>
<td>pcmcia</td>
<td>a complete PCMCIA support package</td>
</tr>
<tr>
<td>linux</td>
<td>Kernel patch</td>
</tr>
<tr>
<td>Uclibc</td>
<td>Small c library</td>
</tr>
</tbody>
</table>
Automation Building Procedure

First-time use the Wall-building auto tools

Y

Checkout the autotools

Checkout wall from CVS into the backup dir

N

Copy Image into the storage device

Copy base image from the backup dir into the working dir

Build the wall system

Build and save the base image in the backup dir

Wallmaker.sh

Refresh.sh

Generate the “base image”

backup.sh
Mini-Project: A Linux-based HTTP server

- Purpose:
  - Download source codes by CVS or SVN
  - Be familiar with the uLibc BuildRoot System
**System Spec of Mini-Project:**

- A http server with port=48311
- The root dir of the server is /tmp/homepage
- The dir should have an video file
- The size of the video file > 4MB
- A user can play the file at another computer.
A example of the result
Procedures of Mini-Project (I)

- Install CVS or SVN client
- Download BuildRoot from uCLibc website
- Enable related items
- Copy a 4MB video file into the package/customize/source/xxx/xxx
- Build root image
- Chroot to the built image file
- Run the tthttpd (tthttpd –help)
- Connect from another PC
Discussions of Mini-Project

1. List your procedures and the used commands.
2. Show the screenshots for your results (3 pics).
3. Which packages are built in the image?
4. If without the video file, what is the length of the image file?
5. Do you select the “customize” package? What is its purpose?
Discussions of Mini-Project

6. After all procedures, show the dirs and files in the *package/customize* directory (ls –alR package/customize)

7. Besides SVN and BuildRoot, do you install any packages in your build system for the mini-project? What are their purposes?

8. How many times “make” do you type before successfully getting the final image? Why?

9. How much time do you spent for the project?
Topic of Term-Project: A Linux-based Firewall

- Purpose:
  - Build a project under your own cvs or svn server
  - Be familiar with the uCLibc BuildRoot system
  - Learn how to add packages into BuildRoot sys.
  - Simple configuration on iptables
System Spec of Term-Project:

- Your own CVS or SVN server
  - Have your own BuildRoot project
  - Some files are revised and have multiple versions

- Build a firewall which blocks all direct connections
  - is running on a (virtual) PC
  - provide a private LAN

- Provide a http proxy
  - Filter the pornography web pages
  - You may need the two open source packages: Squid and Dansguardian

- Downsize the image

- Alternating Platforms
Discussions of Term-Project

1. List your procedures and used commands. How you create a project in your cvs or svn server? How you add packages into BuildRoot sys?
2. Print your own xxxx.mk and explain it.
3. Show the screenshots for your results.
4. Which packages are built in the image?
5. The size of the built image? Before and after downsize? Describe what are removed after downsizing.
6. How can you provide the pornography filter?
7. How many times “make” do you type before getting the final image? Why?
8. How much time do you spent for the project?