Creating Your Own Project on-top of AGL

From minimal image to own project. How to reuse AGL in your own company.

Jan-Simon Möller, jsmoeller@linuxfoundation.org
Intro

Jan-Simon Möller
AGL Release Manager

jsmoeller@linuxfoundation.org
dl9pf @ freenode
Topics

- What and why?
- The Quick & Dirty
- The Nonpersistent
- The Good
- The Best
- Best Practices
- Summary
- Q/A

How to throw things in for testing...

The better way of throwing stuff in ...

How to make things more persistent ...

A good way to maintain your project ...
What and why ?!!
What and why?

- Show how to add software to AGL
- Point out common pitfalls
- Highlight best practices

- Changes in project can have a big impact on the outcome of the build
  - need to be careful and aware
- Vision and goal is to create
  - SDK per architecture (not per board)
  - Limit SDK variants !!
  - Common package feed
AGL repo checkout (since HH)

- bsp
- external
- meta-agl
- meta-agl-demo
- meta-agl-cluster-demo
- meta-agl-telematics-demo
- meta-agl-devel
- meta-agl-extra

- Board support
- External repositories (=upstream)
- AGL 'core' layers
- AGL 'demo' layers
- AGL 'devel' layers
The Quick & Dirty
The Quick & Dirty

- clone AGL
- just create/copy your recipe in some layer (or use devtool - same outcome)
- hack image recipe/packagegroup
- call aglsetup.sh & bitbake

```
repobuild-quick-and-dirty> repo status
project meta-agl-demo/  (** NO BRANCH **)
-- recipes-demo-hmi/foo-qnd/foo-qnd_git.bb
-m recipes-platform/packagegroups/packagegroup-agl-demo-platform.bb
```
The Quick & Dirty

- Simple and straightforward
- Not persistent
- Can't share
- Can't maintain
- Not updated with repo sync
The Nonpresistent
The Nonpresistent

- clone AGL and call
  
  ```bash
  meta-agl/scripts/aglsetup.sh agl-demo agl-devel agl-localdev
  ```

- then (within build/) we use bitbake-layers to create the layer folder
  
  ```bash
  bitbake-layers create-layer --priority 20 ..:/meta-localdev/
  ```

- add your files underneath meta-localdev/
The Nonpresistent

.  
  |-- bsp  
  |-- build  
  |-- external  
  |-- meta-agl  
  |-- meta-agl-cluster-demo  
  |-- meta-agl-demo  
  |-- meta-agl-devel  
  |-- meta-agl-extra  
  |-- meta-agl-telematics-demo  
  `-- meta-localdev

  meta-localdev/
  `-- COPYING.MIT  
  `-- README  
  `-- conf  
  |   `-- layer.conf  
  `-- recipes-platform  
  |   `-- packagegroups  
  |      `-- packagegroup-X.bbappend  
  `-- recipes-example  
       `-- example  
       `-- example_0.1.bb
The Nonpresistent

- Own changes in separate YP compatible layer
- Can be shared
- Can be reused

- Not persistent
- Need to redo 'every time' when cloned
- Not updated with repo sync
The Good
The Good

- clone AGL and
- clone your project as meta-localdev

```bash
git clone https://foo.bar/projects/meta-baz.git meta-localdev
```

- call

```bash
meta-agl/scripts/aglsetup.sh agl-demo agl-devel agl-localdev
```

- call bitbake
The Good

.  
|-- bsp  
|-- build  
|-- external  
|-- meta-agl  
|-- meta-agl-cluster-demo  
|-- meta-agl-demo  
|-- meta-agl-devel  
|-- meta-agl-extra  
|-- meta-agl-telematics-demo  
 `-- meta-localdev

meta-localdev/  (cloned from XYZ)  
|-- COPYING.MIT  
|-- README  
|-- conf  
|  `-- layer.conf  
|-- recipes-platform  
|  `-- packagegroups  
|     `-- packagegroup-X.bbappend  
`-- recipes-example  
   `-- example  
      `-- example_0.1.bb
 ● Own changes in separate YP compatible layer
 ● Can be shared
 ● Can be reused
 ● Recipes 'persistent' in git

 ● Need to redo 'every time' when cloned
 ● Not updated with repo sync
The Best
The Best

- clone AGL
- add your add-on manifest to .repo/local_manifests/
- call repo sync (again)
- call aglsetup.sh and bitbake as usual

```bash
repo init -u https://gerrit.automotivelinux.org/gerrit/AGL/AGL-repo.git
mkdir .repo/local_manifests/
curl https://raw.githubusercontent.com/dl9pf/meta-own-project/master/.project-manifest.xml > \\
        .repo/local_manifests/meta-own-project.xml
repo sync
```
.project-manifest.xml:

<?xml version="1.0" encoding="UTF-8"?>
<manifest>

  <remote name="github" fetch="https://github.com/" />
  <project name="dl9pf/meta-own-project" remote="github" path="meta-own-project" >
    <copyfile src=".project-manifest.xml" dest=".repo/local_manifests/meta-own-project.xml" />
  </project>

</manifest>
The Best

- Own changes in separate YP compatible layer
- Can be shared
- Can be reused
- Updated when syncing with repo

- Need to add once during initial checkout (but this is perfectly scriptable)
The Best (also)

Maintain **own repo manifest** with your projects added already.

Downside: **need to sync/rebase** your repo manifest with AGL all the time yourself!
Best Practices
Best Practices

- Create your own image recipe by including either
  - agl-image-minimal.inc
  - agl-image-ivi.inc
  - agl-demo-platform.inc
  - agl-cluster-demo-platform.inc
- Create a packagegroup if more packages are added

```bb
my-layer/recipes-platform/images/my-image.bb:
DESCRIPTION = "FOO image contains a simple FOO UI."

require agl-image-ivi.inc
LICENSE = "MIT"
IMAGE_FEATURES_append = " \n  "

# add packages for FOO (include foo pkggroup) here
IMAGE_INSTALL_append = " \n  packagegroup-agl-foo \n  "
```
AGL Profiles

Profiles build on each-other:

|-- meta-agl
 `-- meta-agl-profile- core
    |-- meta-agl-profile- telematics
    `-- meta-agl-profile- graphical
       |-- meta-agl-profile- graphical-html5 <- SDK w/ HTML5
       |    |-- meta-agl-demo-html5 <- soon: HTML5 IVI demo
       |-- meta-agl-profile- graphical-qt5 <- SDK w/ qt5
       |    |-- meta-agl-demo <- AGL IVI demo (qt5)
       |    `-- meta-agl-cluster-demo <- AGL cluster demo
       `-- meta-agl-profile- hud <- placeholder for hud
Best Practices

- A project sits on-top of a stack
- Rules:
  - Do not use .bbappends
  - Do not use .bbappends
  - Do not use .bbappends

Do not modify the stack from within the toplevel project.
Best Practices

- Your code needs to be **MACHINE-independent** (AGL has multiple arches!)
- If there are different optimizations (e.g. video decoder pipeline)
  - Have the **generic** variant (e.g. software decoding) as the **default**
  - Then either (in order of preference):
    - runtime-detect the available options and switch
    - provide a configuration file for the user to switch from default to optimized
    - and make it available as build-time switch as well!

  better generic first, then specific as option
Best Practices

- "Upstream first" whenever possible
- Submit your changes early
- Small changes are easier to review
- if there are commonalities, rework things to include the common set
Q/A
AGL is a great stack for your automotive solution.
Thanks!

Contact:

Jan-Simon Möller
jsmoeller@linuxfoundation.org

@dl9pf