What are we trying to solve

The situation is described in

Alice & Bob scenario is a real-world use case of proprietary software distribution for many companies.
Short comings of viable alternatives

- bin_package.bbclass
  - Need separate recipes for Alice and Bob.
  - Managing DEPENDS.
Shortcomings of viable alternatives

- `package_tar`
  - Just another packaging format
  - Can only create tar files after packages have been split.
  - Doesn’t honor the users’ package configuration options.
  - Can’t handle DEPENDS
  - Not possible to create separate tar files for Bob / Charlie
prebuilt.bbclass

A custom bitbake class for prebuilt needs
Proposed solution – prebuilt.bbclass

- Use same recipe at both sites.
- For Alice - generates tarball(s) out of ${D} directory suitable for each user.
- For Bob / Charlie - restores ${D} from shared tarball as sources are not available.

```bash
## prebuilt.bbclass ##
# Generate prebuilt tarball for Alice
do_generate_prebuilt[dirs] = "${D}"
do_generate_prebuilt[cleandirs] = "${PREBUILT_DIR}/${PACKAGE_ARCH}"
do_generate_prebuilt() {
    tar --owner 0 --group 0 --czvf ${PREBUILT_DST} -C ${D} . || [ $? -eq 1 ]
}

# Install Prebuilt tarball for Bob / Charle
do_install_prebuilt[dirs] = "${D}"
fakeroot do_install_prebuilt() {
    tar -xvzf ${PREBUILT_SRC} -C ${D}
}
```
prebuilt.bbclass - Alice workflow

- Generate tarball(s) out of `${D}` suitable for Bob / Charlie.
- Optionally strip libs before creating tarball(s).
**prebuilt.bbclass - Alice workflow**

- Conf file with details of files to be shipped for Bob / Charlie

```bash
## prebuilt.conf ##

PB_VARIANTS = "bob charle"

bob_PREBUILT_PACKAGES += " pkg1 pkg2 "
bob_PREBUILT_FILES_pkg1="/usr/bin/pkg1-bin"
bob_PREBUILT_FILES_pkg2="/usr/lib/pkg2-lib"
bob_PREBUILT_FILES_pkg2="/usr/include/pkg2-header1.h"
bob_PREBUILT_FILES_pkg2="/usr/include/pkg2-header2.h"

charle_PREBUILT_PACKAGES += " pkg2 pkg3 "
charle_PREBUILT_FILES_pkg2="/usr/lib/pkg2-lib"
charle_PREBUILT_FILES_pkg2="/usr/include/pkg2-header1.h"
charle_PREBUILT_FILES_pkg3="/usr/bin/pkg3-bin"
```
prebuilt.bbclass - Alice workflow

- Generate tarball(s) out of ${D} directory suitable for Bob / Charlie

```python
do_generate_prebuilt[outputdirs] = "${DEPLOY_DIR_PREBUILT}"

do_generate_prebuilt() {
    packages = (d.getVar("PB_PACKAGES") or "").split()
    variants = (d.getVar("PB_VARIANTS") or "").split()
    for p in packages:
        for v in variants:
            files = d.getVar(v + "_PREBUILT_FILES_" + p)
            stripped = d.getVar("PREBUILT_STRIP_" + p + package)
            cmd = "tar -cf %s -T /dev/null" % (tarball) # Create empty archive
            for f in files: # Append files
                cmd = "tar --owner 0 --group 0 -rvf %s ./%s" % (tarball, file)
            oe.utils.getstatusoutput(cmd)
}

python () {
    if d.getVar('USE_PREBUILTS'):
        ...
    elif d.getVar('DEPLOY_DIR_PREBUILT'): # generate prebuilt package
        bb.build.addtask('do_generate_prebuilt', 'do_package', 'do_install', d)
}```
prebuilt.bbclass - Bob workflow

- Make tasks like fetch, compile, install etc. as noexec.
- Install tarball into `${D}` and let post install tasks run as is.

```
<table>
<thead>
<tr>
<th>Task</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>do_fetch</td>
<td>❌</td>
</tr>
<tr>
<td>do_configure</td>
<td>❌</td>
</tr>
<tr>
<td>do_compile</td>
<td>❌</td>
</tr>
<tr>
<td>do_install</td>
<td>❌</td>
</tr>
<tr>
<td>do_pb_install</td>
<td>✔️</td>
</tr>
<tr>
<td>do_package</td>
<td>✔️</td>
</tr>
</tbody>
</table>
```

`${D}` of WORKDIR at Bob
prebuilt.bbclass - Bob workflow

• Make tasks like fetch, compile, install etc. as noexec.
• Install tarball into \${D} and let post install tasks run as is.
• Prune DEPENDS to drop source dependencies.

```python
# In case of prebuilt usage, these tasks are discarded
PREBUILT_BYPASS_TASKS += "\n  do_fetch do_unpack do_patch do_configure do_compile do_install do_populate_lic"

python () {
  if d.getVar('USE_PREBUILTS'):
    # use prebuilt package
    for task in d.getVar('PREBUILT_BYPASS_TASKS').split():
      d.setVarFlag(task, 'noexec', '1')
    bb.build.addtask('do_install_prebuilt', 'do_populate_sysroot', 'do_install', d)
}
```
prebuilt.bbclass - Bob workflow

- Install tarball into ${D} and let post install tasks run as is.

```python
fakerooot python do_install_prebuilt() {  
    arch = d.getVar('PACKAGE_ARCH')  
    dest = d.getVar('D')  
    pn = d.getVar("PN")  
    pv = d.getVar('PV')  
    tarball = ""  
    done = True  
    # Check if prebuilt tarball exist  
    for prebuiltsrc in (get_prebuilt_paths(d) or "").split():  
        ppackages = (d.getVar("PREBUILT_PACKAGES") or "").split()  
        for ppackage in ppackages:  
            tbpath = prebuiltsrc + "/" + ppackage + ".tar.gz"  
            if os.path.exists(tbpath):  
                tarball = tbpath  
                cmd = "tar -xvzf %s -C %s % (tarball, dest)  
                (retval, output) = oe.utils.getstatusoutput(cmd)  
                if retval:  
                    bb.warn("Errors in extracting prebuilt: %s (%s)" % ( tarball, output))
```
**prebuilt.bbclass - Bob workflow**

- Selectively remove build dependencies using PREBUILT_INHIBIT_DEPS

```bash
## myrecipe.bb ##
DEPENDS = "A B C-native D-toolchain"
...
PREBUILT_INHIBIT_DEPS = "C-native D-toolchain"
...
inherit prebuilt

### prebuilt.bbclass remove build-only deps ###
Python () {
    inhibit_deps = d.getVar('PREBUILT_INHIBIT_DEPS')
    if inhibit_deps == "1":
        d.setVar('DEPENDS_remove_pn-%s' % pn, d.getVar('DEPENDS'))
    elif inhibit_deps != "0":
        d.setVar('DEPENDS_remove_pn-%s' % pn, inhibit_deps)
}
```
Proposed solution – Conclusion

• Tested internally and want to consider upstream

• Known limitations
  • PACKAGECONFIGs are unsupported.

• Possible enhancements
  • Prebuilt tar files distribution.

• Related community discussion [https://lists.openembedded.org/g/openembedded-core/topic/72381872#128488](https://lists.openembedded.org/g/openembedded-core/topic/72381872#128488)
Thanks for your time