Packaging Go packages as shared libraries

Introduction

I'm a Canonical employee who's been asked to work on adding support for Go to the native (i.e. "not gccgo") toolchain for shared libraries. This work is upstream now and is in the Go 1.6 release. We can't actually use this work for what we want to do until there is support in the archive for it though, so I hope to get some changes into dh-golang.

Philosophical rambling

There is some tension between the viewpoints of Debian and the typical Go developer when it comes to distributing libraries (for example, around whether making releases is a useful thing).

`dh-golang` (it seems to me) works on the sensible idea that a path of the form "`host/user/name/...`" as passed to go get is the appropriate unit for packaging, and the package name should be derived from the import path. I intend to extend this idea to building such Go packages into shared libraries.

Packaging stuff (the "UI")

This is a typical `debian/control` file for a package that contains a library to be used by other Go programs (i.e. a package that does not generate any executables):

```
Source: golang-github.com-user-repo
Section: net
Priority: extra
Maintainer: Go Packager <go-pkg@example.com>
Build-Depends: debhelper (>= 8.0.0), dh-golang, golang-go
Standards-Version: 3.9.2
Vcs-Git: git://git.debian.org/collab-maint/golang-github.com-user-repo.git
Vcs-Browser:
http://git.debian.org/?p=collab-maint/golang-github.com-user-repo.git;a=summary
Xs-Go-Import-Path: github.com/user/repo

Package: golang-github.com-user-repo-dev
Architecture: all
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: Random Go library
 A random Go library that does something very useful.
```

It is used with a rules file like this:

```
#!/usr/bin/make -f

%:
	dh $@ --buildsystem=golang --with=golang
```

Today, dh-golang builds this into a dev package that simply contains the source which can be used when building another package that contains an executable. With my version of dh-golang, all that needs to be done is changing the control file to build both a shared library package and a dev package that contains both the source (so such packages as exist today continue to build) and the files necessary to link against the shared library provided by the shared library package. Here's an updated control file:

```
Source: golang-github.com-user-repo
Section: net
Priority: extra
Maintainer: Go Packager <go-pkg@example.com>
Build-Depends: debhelper (>= 8.0.0),
               dh-golang,
               golang-go,
               golang-any-shared-dev
Standards-Version: 3.9.2
Vcs-Git: git://git.debian.org/collab-maint/golang-github.com-user-repo.git
Vcs-Browser:
http://git.debian.org/?p=collab-maint/golang-github.com-user-repo.git;a=summary
Xs-Go-Import-Path: github.com/user/repo

Package: golang-github.com-user-repo-dev
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}, libgolang-github.com-user-repo1 (=
${binary:Version})
Description: Random Go library -- development files
 A random Go library that does something very useful.

Package: libgolang-github.com-user-repo1
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}
Provides: ${golang:Provides}
Description: Random Go library -- shared library
 A random Go library that does something very useful.
```

Let's step through the diff:

```
--- example/control-dev      2015-06-05 15:43:56.597360382 +1200
+++ example/control-dev-shared      2015-06-05 15:55:44.758890558 +1200
```

```
@@ -2,13 +2,20 @@
 Section: net
 Priority: extra
 Maintainer: Go Packager <go-pkg@example.com>
-Build-Depends: debhelper (>= 8.0.0), dh-golang, golang-go
+Build-Depends: debhelper (>= 8.0.0),
+               dh-golang,
+               golang-go,
+               golang-any-shared-dev
```

The `golang-any-shared-dev` contains the files needed to link against the shared library containing the go standard library. The `-any-` part is there because it depends on gccgo on platforms that lack “real go” (currently only powerpc). `golang-any-shared-dev` depends on `golang-any`, so we could just depend on this, but somehow depending on both feels right.

```
 Standards-Version: 3.9.2
 Vcs-Git: git://git.debian.org/collab-maint/dcs.git
 Vcs-Browser: http://git.debian.org/?p=collab-maint/dcs.git;a=summary
 Xs-Go-Import-Path: github.com/user/repo

 Package: golang-github.com-user-repo-dev
-Architecture: all
+Architecture: any
```

The `-dev` package now contains architecture dependent files -- they contain the .a files because the compiler needs to read the export data from them and we need them to be older than the DSO to avoid trying to rebuild the DSO itself. The compiler only needs the export data (`__.PKGDEF`) from the .a file, not the object code, so possibly I could strip out the object code. Currently I don't.

Although one might think the export data is platform independent (and often it is), it isn't in general because of build tags.

```
-Depends: ${shlibs:Depends}, ${misc:Depends}
+Depends: ${shlibs:Depends}, ${misc:Depends}, libgolang-github.com-user-repo1 (=
${binary:Version})
```

As usual, the `-dev` package depends on the shared library package.

```
-Description: Random Go library
+Description: Random Go library -- development files
  A random Go library that does something very useful.
+
+Package: libgolang-github.com-user-repo1
```

`dh-golang` decides whether or not to build a shared library by reading the control file to look for a package with a name matching `/^lib(golang-.*?)-?([0-9]+)$/`. Like C, the shared library package will install the shared library to `/usr/lib/${DEB_HOST_MULTIARCH}/libgolang-github.com-user-repo.so.1` and the -dev package installs a symlink to it. The symlink is not installed alongside the `.so.1` however – it is where the Go linker will look for it. `dh-golang` arranges for the usual `SONAME` trickery to ensure that dependent libraries link against the `.so.1` file.

```
+Architecture: any
+Depends: ${shlibs:Depends}, ${misc:Depends}
```

Go packaging uses `dpkg-shlibdeps` just like C packages.

```
+Provides: ${golang:Provides}
```

See the section on ABI issues below.

```
+Description: Random Go library -- shared library
+ A random Go library that does something very useful.
```

No changes to the `rules` file is required. "`/usr/lib/*/gocode`" should be added to the `.install` file for the `-dev` package (if the package previously only defined a `-dev` package and so got away without any `.install` files, you'll need to add one that contains "`/usr/share/gocode`" as well). No `.install` file is required for the shared library package.

More complicated `rules` files might need to change, as exactly where files end up during the build has changed a bit.

For a package that just builds one or more executables and does not build a `-dev` package, you can link the executables against any shared libraries from the build dependencies by setting `DH_GOLANG_LINK_SHARED` in the rules file.

ABI Issues

It is simply a fact that Go libraries will not have anything like the same ABI stability as a C library. Partly that might down to the relative immaturity of the toolchain but it also a consequence of having a richer runtime. In any case, for the next few years, we should expect that any non-trivial update to a library will break ABI and two libraries compiled with different versions of the toolchain will not be ABI compatible.

Upstream Go defines an "ABI hash" for a shared library and these are checked during process startup to ensure that the hash seen at runtime matches that seen at build time.

ABI-breaking updates to a library will be usually be handled by doing a transition in -proposed/unstable, i.e. rebuilding rdeps until britney is happy, although we also have the option of bumping the SONAME if that turns out to be more convenient for some reason or another.

Updates to the compiler will require rebuilding all dependent packages and will be handled in the same way. New major versions of the toolchain are expected every 6 months so this is a once a cycle task and the distro people assure me this isn't too bad. (By default, even minor revisions of the toolchain will need this treatment, but I think possibly we can patch things to avoid this. Open question for now).

dh-golang defines a substvar, `${golang:Provides}`, as seen above, to be the concatenation of the shared library package name and the ABI hash, and further it creates a shlibs file that causes dependent packages to depend on this hash-containing Provides rather than just the package name. The point of *this* is that if someone uploads an ABI-breaking change without bumping the sover, britney will prevent the package from making into testing/the `-release` pocket because it would make its dependencies uninstallable. An alternative would be to make the packager specify the ABI hash and *check* it during package building, but this would mean touching each and every Go shared library package during the transition to a new version of the toolchain and that isn't desirable.

This is all a bit like how haskell and ocaml packaging works, I think. Except that's all cdbs which is even worse than perl.

Implementation

I have a branch of dh-golang at https://github.com/mwhudson/dh-golang/tree/shlibs-for-real which works as described above.

I also have a PPA at

https://launchpad.net/~mwhudson/+archive/ubuntu/lxd-shlibs/+packages

that contains:

- a patched Go 1.6 modified to build a golang-go-shared-dev package as above
- my version of dh-golang
- lxd and its build dependencies converted to create shared libraries.

Conversion

I've written a tool that converts existing packages to the above form:

https://github.com/mwhudson/shlibify/blob/master/shlibify

In some cases some extra cleaning up is required, but it mostly works pretty well in the testing I have done so far.

I don't necessarily propose updating all libraries to be built as shared libraries immediately. It would probably be better to pick some binary-shipping packages we want to use with shared libraries (e.g. lxd) and start with their dependencies.

Documentation

http://pkg-go.alioth.debian.org/packaging.html will need to be updated when this work gets done in Debian.