Debian CLI Policy

Mirco Bauer <meebey@debian.org>
Brandon Hale <brandon@smarterits.com>
Sebastian Dröge <slomo@debian.org>
Dylan R. E. Moonfire <debian@mfgames.com>

Version 0.7

Abstract

This document lays out basic policies regarding packaging Mono, other CLRs and CLI based applications/libraries on Debian GNU/Linux.

Copyright Notice

Contents

Chapter 1

Policy History

Here are the changes to the Debian CLI Policy document.

Changes from 0.5.1 to 0.7:

- 'File Locations' on page 5: GAC libraries must now go in /usr/lib/cli/assembly_name-X.Y instead of /usr/lib/cli/upstream_package_name-X.Y, as one source package might ship many assemblies with different ABI versions. This would produce very confusing directory names.
- 'Naming & Versioning' on page 6: Late-GAC install is now mandatory.
- 'Build Dependencies' on page 6: Added CLI SDKs as alternative to the compiler.
- 'Signing' on page 7: Using the mono.snk key of cli-common-dev is now mandatory if upstream doesn't ship one.
- 'MONO_DISABLE_SHM' on page 10: Replaced MONO_SHARED_DIR workaround with cli.make and MONO_DISABLE_SHM.
- 'debhelper 7 Example' on page 14: Added debhelper 7 example.
- 'Policy Files' on page 7: Added reference to the mono-api-check tool and made raising clilibs version mandatory.
- "'.NET" or long "Microsoft .NET Framework"' on page 3: Updated URL to Microsoft .NET Guidelines.
- 'Package Names' on page 4: Made upstream tarball names clearer.
- 'File Permissions' on page 5: Replaced find commands with dh + cli.make.
- 'Naming & Versioning' on page 6: Removed ASP.NET as it's not a programming language and added IronPython and IronRuby.

Changes from 0.5.0 to 0.5.1:

- 'Package Names' on page 4: Added examples for the different meanings of package name.
- 'Naming & Versioning' on page 6: Explicitly name the "lib" prefix requirement for library packages.

Changes from 0.4.4 to 0.5.0:

- Removed DRAFT tag, the policy is now official.
- 'Build Dependencies' on page 6: Added C# 3.0 to the compiler list.
- 'File Permissions' on page 5: Added dh_clifixperms as alternative to the find command.

Changes from 0.4.2 to 0.4.3:

- 'dh_installcligac' on page 13: Fixed order of dh_installcligac calls.
- 'debhelper 5/6 Example' on page 14: Fixed debhelper example (order).

- ‘cdbs Example’ on page 14: Fixed cdbs example (order).

Changes from 0.4.1 to 0.4.2:

- ‘GAC Policy File Example’ on page 15: Fixed naming of the policy files.

Changes from 0.4.0 to 0.4.1:

- ‘debhelper 5/6 Example’ on page 14: Fixed typo.

Changes from 0.3.0 to 0.4.0:

- ‘Build Dependencies’ on page 6: Added nemerle to the compilers.
- ‘Packaging Policy’ on page 5: Added a packaging chapter that includes some of the old chapter and some new.
- ‘GAC Library Packaging’ on page 6: Added informations about signing and policy files.
- ‘dh_installcligac’ on page 13: Added and consolidated the information on `dh_installcligac`.
- ‘File Locations’ on page 5: Require that files are installed into `/usr/lib/package` or `/usr/lib/cli/package-X.Y` now.

Changes from 0.2.1 to 0.3.0:

- “‘.NET” or long “Microsoft .NET Framework”’ on the next page: Added URL for the “.NET” term.
- ‘GAC - Global Assembly Cache’ on the facing page: Added explanation of GAC.
- ‘GAC Library Packaging’ on page 6: Added section for naming of GAC packages.

Changes from 0.2.0 to 0.2.1:

- ‘Helper Scripts: cli-common-dev’ on page 13: Added examples for debhelper and CDBS.

Changes from 0.1.1 to 0.2.0:

- ‘Policy History’ on the previous page: Added chapter “Policy History”
- ‘Build Dependencies’ on page 6: Compiler dependency is no longer strict on mono-mcs
- ‘Helper Scripts: cli-common-dev’ on page 13: Note that dh_makeclilibs must be called before dh_clideps
- ‘Build Dependencies’ on page 6: Moved dh_clideps and dh_makeclilibs into their own subsections
- ‘File Permissions’ on page 5: Added chapter “File Permissions”
- ‘Migrating Existing Packages’ on page 15: cli-wrapper is now deprecated
- ‘Introduction’ on page 9: Added an external link for DllNotFoundException

Chapter 2

Used Terms

The ".NET" area uses its own set of abbreviations, which can look confusing to other people. This chapter lists some of the terms along with their explanations:

2.1 CLI - Common Language Infrastructure

This is what most people mean when they say ".NET". The CLI defines mainly the virtual machine, the bytecode and how everything works together. It is both an ISO (http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=36769) and ECMA (http://www.ecma-international.org/publications/standards/Ecma-335.htm) standard.

2.2 CLR - Common Language Runtime

The CLR is an implementation of the CLI (often with a lot of add-ons or tools for developers). Mono and Microsoft .NET Framework are CLRs.

2.3 CIL - Common Intermediate Language

The CIL is the format of the bytecode for binaries and libraries used by the CLI.

2.4 ".NET" or long "Microsoft .NET Framework"

The ".NET" word is a Microsoft marketing phrase and mostly is a CLR with added Microsoft technologies like: ASP.NET, VB.NET, System.Windows.Forms, Passport plus a lot of other things.

We highly discourage from using any form of the word ".NET", it is burdened by copyright and marketing. We advice to use the correct term instead, which is usually CLI.

If you really want to use the ".NET" term in a correct form please refer to the Microsoft .NET Guidelines (http://www.microsoft.com/about/legal/trademarks/usage/net.mspx).

2.5 GAC - Global Assembly Cache

The GAC contains and manages the libraries for the CLR. It allows users to install multiple versions of the same library and enables loading of the right version when an application is executed.

Mono stores the GAC at `/usr/lib/mono/gac`

DotGNU Portable.NET stores the GAC at `/usr/lib/cscc/lib`

2.6 Package Names

There are different set of package names we refer in this policy to. Here a list of examples for the different names:

- assembly (file) names:

```
gtk-sharp.dll

log4net.dll

FlickrNet.dll
```

- (debian) source package names:

```
gtk-sharp2

log4net

libflickrnet
```

- (debian) binary package names:

```
libgtk2.0-cil

libgnome2.0-cil

liblog4net1.2-cil

libflickrnet2.1.5-cil
```

- upstream package names (a good indicator is the pkg-config file name):

```
gtk-sharp-2.0

log4net

flickrnet
```

- upstream tarball names (without version and file extension):

```
gtk-sharp-2.12.9.tar.gz -> gtk-sharp

log4net-1.2.0-beta8.zip -> log4net

FlickrNet-25207.zip -> FlickrNet
```

Chapter 3

Packaging Policy

This section describes the additions to the Debian Policy (http://www.debian.org/doc/debian-policy/) that are required for CLI packages.

3.1 General Packaging

3.1.1 Architecture

For packages that consist of 100% managed code, "Architecture: all" *must* be chosen in debian/control.

Packages containing a mix of managed and native code *must* be "Architecure: any" or depending on the specific package a more restricted set of architectures is valid.

3.1.2 File Locations

The package's applications, libraries and meta-data *must* be installed into `/usr/lib/`*`upstream_package_name`*.

Libraries that will be installed into the GAC *must* be installed into `/usr/lib/cli/`*`assembly_name`*`-X.Y` (for more details about the X.Y version see GAC versioning). *assembly_name* is the assembly name without the file extension (.dll). The commonly seen `/usr/lib/`**`mono`**`/`*`upstream_package_name`* path should *only* be used for Mono project packages.

Example path for the `log4net` package:

```
/usr/lib/cli/log4net-1.2
```

Never install native "glue" libraries into `/usr/lib`, instead install them at `/usr/lib/cli/`*`assembly_name`*`-X.Y`. When moving libraries update the references to the new location using a DLL Map. See the Mono DLL maps section for an example.

The only exception here is for native libraries that are of wider use; can be used other packages. Native libraries should be packaged according to the Library Packaging Guide (http://www.netfort.gr.jp/~dancer/column/libpkg-guide/libpkg-guide.html) in a Debian Policy conformant way.

You *must not* install application files (`.exe`) directly into `/usr/bin`. Instead create a wrapper script into `/usr/bin` to allow them to be run without path and the `.exe` suffix.

3.1.3 File Permissions

Source code files (`*.cs`, `*.vb`, `*.boo`, etc.) should be non-executable.

Library files (`*.dll`) should be non-executable.

Debug symbol files (`*.mdb`) should be non-executable.

Assembly config files (`*.config`) should be non-executable.

Application files (`*.exe`) *must* have the executable flag (+x) set to enable compatiblity with direct invokation as `./foo.exe` using Linux's BINFMT support.

To ensure that all files have correct permissions, you *should* use Debhelper's `/usr/bin/dh` combined with `cli.make`. Otherwise you *should* add `dh_clifixperms` after `dh_fixperms` in the binary-* targets of `debian/rules`.

3.1.4 Build Dependencies

At a minimum, CLI packages *should* Build-Depends on `cli-common-dev` (>= 0.7) and the appropriate CLI compiler or CLI SDK package.

Current CLI compilers in Debian:

- C#: `mono-mcs` (>= 1.0) | c-sharp-compiler
- C# 2.0: `mono-gmcs` (>= 1.1.8) | c-sharp-2.0-compiler
- C# 3.0: `mono-gmcs` (>= 1.2.5) | c-sharp-3.0-compiler
- Nemerle: `nemerle` (>= 0.9)
- Boo: `boo` (>= 0.5.6)

Current CLI SDKs in Debian:

- Mono: `mono-devel` (>= 2.4.2.3)

Software that uses Mono via the C interface library (`libmono.so`) or requires the `/usr/lib/pkgconfig/mono.pc` file *must* Build-Depends on libmono-dev (>= 1.0)

Note that there are architectures for which no CLR is available and thus you may have to restrict the Build-Depends for your package to the architectures available.

If your package is `Architecture: all`, you should specify this as Build-Depends-Indep. Never put `debhelper`, `cdbs`, `dpatch` and `quilt` into Build-Depends-Indep. See the Debian Policy Manual (http://www.debian.org/doc/debian-policy/ch-relationships.html#s-sourcebinarydeps) for more information on this.

3.2 GAC Library Packaging

Libraries that are installed into the GAC should provide decent ABI stability and be useful for other packages. Otherwise, they should remain private to the package.

3.2.1 Naming & Versioning

Libraries that are installed into the GAC *must* be strong-named, i.e. signed.

Libraries *must* to be installed into the GAC at package install time (postinst) which is provided by the dh_installcligac tool of the `cli-common-dev` package.

Each of the libraries in the GAC has an assembly version number that consists of 4 parts (major, minor, build and revision number). When loading libraries from the GAC all 4 parts and the public signing key fingerprint must match.

It is general practice and recommended by Microsoft (http://msdn.microsoft.com/netframework/programming/deployment/default.aspx?pull=/library/en-us/dndotnet/html/dplywithnet.asp#dplywithnet_version) that a library is ABI compatible when only the build and revision number change and the major and minor number stay the same.

The library package name *must* be prefixed with `lib`.

To reflect the ABI stability and prevent breakages when a ABI-incompatible version is released, a similar solution for native library packages (http://www.netfort.gr.jp/~dancer/column/libpkg-guide/libpkg-guide.html#naminglibpkg) is used. The major and minor number *must* mirror the SONAME version and the resulting package name should be `libfooX.Y-cil`, where X is the major and Y the minor number of the assembly version.

One notable exception for this naming are assemblies that end on a number (Mono.C5 for example). In this case the package should be named `libfoo123-X.Y-cil` (i.e. `libmono-c5-0.5-cil`) to improve the readability.

The `-cil` suffix is chosen to prevent confusion with native library package names. Never use "sharp" in the package name as it does not represent the language, and a CLI library can be used with all CLI implemented / enabled languages such as C#, IronPython, IronRuby, Boo, Nemerle, J#, VB.NET (full list (`http://www.mono-project.com/Languages`)).

Unnecessary package renames should be avoided. Existing package names that do not follow this policy should not be renamed until the next incompatible ABI change, at which point the new naming scheme should be used.

If the upstream software does not use major and minor number to reflect ABI stability or breaks ABI with a change in build or revision, the package *must* be renamed to either `libfooA.B.C-cil` or `libfooA.B.C.D-cil` (where A, B, C, D are the complete assembly version numbers), depending at which point (major or minor) the breakage occurred. All Policy Files *must* be dropped at this stage until a new major or minor version is released.

The upstream software may use wildcards in the assembly versions (1.2.* for example) which are filled by the compiler with a random value. You *must* replace these wildcards with 0 (1.2.0.0 in the example) to make it possible to use Policy Files and make predictable version numbers.

More than one library can be installed in one package but it is required that they *must* all have the same assembly version and belong together.

3.2.2 Policy Files

As explained above a exact match of the version number is required to load a library from the GAC. To override this behaviour and make different versions of ABI-compatible library packages really ABI-compatible you have to use Policy Files (`http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconcreatingpublisherpolicyfile.asp`). These files have to be named `policy.X.Y.foo.dll` (where X and Y are the major and minor number of the assembly it should be compatible with), it *must* be signed with the same signing key as the original assembly and it must be installed into the GAC. For information on how to create policy files look at the previous Policy Files link or at the example below.

Overriding the GAC policy should only be done when the different library versions are really ABI-compatible. This can be checked using mono-api-check of the `mono-devel` package. You *must* also raise the version in the clilibs control file to the minimum version when new interfaces/classes/methods were added.

3.2.3 clilibs Control File

The `clilibs` control file *MUST* be present in all GAC library packages. It can be created with the dh_makeclilibs helper script and has a format similar to the `shlibs` file created by `dh_makeshlibs(1)` and also has a similar use: it is used by dh_clideps helper script to find the correct dependencies.

You *should* always set the minimum required version of the library in the `clilibs` file.

3.2.4 pkg-config File

Many libraries deliver a `.pc` file for use by the `pkg-config` helper utility, which aids other libraries and applications to link against libraries.

All GAC library packages should have a pkg-config `.pc` file located in `/usr/lib/pkgconfig`. The filename *must* be identical to that shipped by upstream.

3.2.5 Signing

When installing libraries into the GAC signing is required. The signing key should be supplied by upstream. If upstream is not supplying the key then you *must* use the `mono.snk` key from the `cli-common-dev` package. This key *must* be used for all following versions of the library to maintain ABI compatbility.

Unnecessary ABI breakages should be avoided. Existing keys shipped by the source package should not be replaced (with mono.snk) until the next incompatible ABI change.

3.3 non-GAC Library Packaging

This includes libraries that are not ABI-stable, may be not strong-named and are usually in an early stage of development. They *must* not include a clilibs control file.

3.3.1 Naming

The package should be named `libfoo-cil` (without a version in the package name) and libraries should not be installed into the GAC but only into `/usr/lib/upstream_package_name`.

Applications using non-GAC libraries *must* copy the libraries they need into their own application directory. You can compare this with static linking of native libraries.

Chapter 4

Mono Specific Packaging help

This section offers help with common problems encountered when packaging Mono-specific applications for Debian.

4.1 Naming

The official name of the Mono Project is: Mono, mono:: or mono. To keep this consistent for users, it should *always* be called "Mono" (not MONO, mono, mono:: or mixed with the .NET name). The explanation of what Mono is, should be in the package *long* description.

4.2 DLL Maps

Often times, upstream software developers are not packagers, and vice versa. Developers do not necessarily test their software with packaging issues in mind. The most common problem we see from this are missing DLL exceptions.

4.2.1 Introduction

When Mono code invokes an external library, it usually calls something like [DllImport("foo")] which expands "foo" to a shared library name such as "libfoo.so" which is then searched for in the library search path.

In Debian and some other binary Linux distributions, packages are split into runtime and developer (-dev) packages. Since the versioned library libfoo.so.X is usually used at runtime, and libfoo.so is a symlink only used when building against the library, the libfoo.so symlink is in the libfoo-dev package.

When packaging an application which uses libfoo.so normal users should not need the -dev packages installed just to run the application. However, Mono defaults to looking for the unversioned libfoo.so, which is unavailable in the runtime package.

When the DLL map is missing or upstream forgets to install the DLL map, it will result in a DllNotFoundException (http://www.mono-project.com/DllNotFoundException) which will stop the execution of the program.

4.2.2 Solution: DLL map config file

This can be fixed by creating a DLL map for the application exe or for the library DLL that is trying to invoke libfoo.so. If libfoo.so is invoked by the DLL bar.dll, create an xml file, bar.dll.config to tell Mono which .so should be loaded at runtime. bar.dll.config should be installed to the same directory as bar.dll.

```
<configuration>
<dllmap dll="foo" target="libfoo.so.0"/>
</configuration>
```

A config file can contain as many dllmap directives as are needed. If the upstream developer already ships a config file, but it is incomplete, you should create a patch against it in your package.

Most Mono software developers are very helpful people, and will readily accept patches to solve this type of bug if you bring it to their attention. Please be sure to inform them of all these changes.

4.3 MONO_DISABLE_SHM

The Mono runtime uses a shared directory, by default `~/.wapi`. This directory will be created/used when any CLI application is executed (like the C# compiler mcs).

There are 2 problems with this:

- In an autobuilder environment often the running user has no home directory.
- Mono uses the wrong home directory when running within fakeroot (it tries `/root/.wapi` instead of `$HOME/.wapi`).

In these cases, the package building will fail, applications will hang, die with strange Mono runtime errors or segfault. This includes dh_clideps or dh_makeclilibs, since they run monodis.

The solution is to include `cli.make` from `cli-common-dev` in `debian/rules` or to manually set the MONO_DISABLE_SHM environment variable.

```
export MONO_DISABLE_SHM = 1
```

Chapter 5

DotGNU Portable.NET Packaging help

This section offers help to common problems encountered when packaging DotGNU Portable.NET-specific applications for Debian.

5.1 Naming

The official name of the DotGNU Portable.NET project is exactly that. To keep this consistent for users, it should be *always* called "DotGNU Portable.NET" (not pnet or Portable.NET). The explanation of what DotGNU Portable.NET is, should be in the package *long* description.

Chapter 6

Appendix

6.1 Helper Scripts: cli-common-dev

When using cli-common-dev and the included dh_* scripts packages *must* Build-Depends on `cli-common-dev` (>= 0.7) (this version may change later, when cli-common-dev has changes which are required to be used by all CLI packages, the CLI Policy version will represent such changes).

6.1.1 dh_makeclilibs

dh_makeclilibs is used to create the clilibs control files which are used later by `dh_clideps` for this or other packages. It *must* only be used when your package contains libraries that other packages may link against.

It has the same use (and very similar parameters) to `dh_makeshlibs`. You should always use the most minimal version necessary.

This program must be called before `dh_clideps`.

See `dh_makeclilibs(1)` for details.

6.1.2 dh_clideps

`dh_clideps` is used to discover the native and managed dependencies of the packages. It uses the clilibs control files, the `.config` of assemblies and the `shlibs` files created by `dh_makeshlibs`. The discovered dependencies are written into the ${cli:Depends} variable.

`dh_shlibdeps` must be run before `dh_clideps`. `dh_makeshlibs` and `dh_makeclilibs` must be run before `dh_clideps`. If not, when two binary packages from the same source package depend on one another, `dh_clideps` will not be able to determine the dependencies.

`dh_clideps` can remove duplicate dependencies created by running `dh_clideps` and `dh_shlibsdeps` when run given the -d parameter.

See `dh_clideps(1)` for details.

6.1.3 dh_installcligac

`dh_installcligac` is used to facilitate the installation of strong-named assemblies into the various caches installed on the user's machine. Its primary purpose is to install the assemblies at the point of installation instead of pre-packing them inside the Debian package; this is also known as late-GAC install.

To identify which assemblies need to be installed into the GAC, `dh_installcligac` uses the `debian/installcligac` or the `debian/packagename.installcligac` to list the assemblies to install or uninstall at installation or removal respectivly.

The file format of the `installcligac` is simple: the full installed path of every assembly to install into the GAC. For example, the liblog4net1.2-cil package would have this in the `debian/installcligac` file:

```
/usr/lib/cli/log4net-1.2/log4net.dll
```

`dh_installcligac` needs to be called after `dh_install` and before `dh_clideps`. See `dh_installcligac(1)` for details.

6.2 Examples

6.2.1 debhelper 5/6 Example

For binary-arch packages:

```
binary-arch: build install
 ...
 dh_shlibdeps -a
 dh_makeclilibs -a -V
 dh_installcligac -a
 dh_clideps -a
 ...
```

For binary-indep packages:

```
binary-indep: build install
 ...
 dh_makeclilibs -i -V
 dh_installcligac -i
 dh_clideps -i
 ...
```

6.2.2 debhelper 7 Example

With debhelper's 7 `/usr/bin/dh` you don't need to add any extra commands to `debian/rules` yourself as debhelper has an API that allows to extend it. `cli-common-dev` as of version 0.5.7 can extend debhelper 7 with all commands that are needed. You can enable this by passing "cli" to dh in `debian/rules` like this:

```
%:
    dh $@ --with cli
```

That's it, you are done! :-)

6.2.3 cdbs Example

```
common-binary-predeb-arch common-binary-predeb-indep::
 dh_shlibdeps
 dh_makeclilibs -V
 dh_installcligac
 dh_clideps
```

6.2.4 Executable Wrapper Script Example

```
#!/bin/sh
exec /usr/bin/cli /usr/lib/package/package.exe "$@"
```

6.2.5 API Compatibility Check Example

You need to install following packages for this example: mono-devel libmono-sharpzip0.6-cil libmono-sharpzip0.84-cil

```
mono-api-check /usr/lib/mono/gac/ICSharpCode.SharpZipLib/0.6.0.0__1b03e6acf1164f73/ICSharpCode.SharpZipLib.dll \
   /usr/lib/mono/gac/ICSharpCode.SharpZipLib/0.84.0.0__1b03e6acf1164f73/ICSharpCode.SharpZipLib.dll
CLI API Check
Assembly Name:          ICSharpCode.SharpZipLib
Missing Interfaces:     44
Additional Interfaces:  79

The two assemblies you compared are not API compatible!
You must use a new package name!

The new assembly has additional interfaces. You must raise
the minimal version in clilibs!
```

The mono-api-check wrapper script checks whether there are new public/protected interfaces (where interface in this context means namespace, class, method, interface, delegate, etc) or any missing ones. When an interface is changed it will show up as missing and additional. You should follow the instructions, in this case you must create a new versioned package for the library and raise the minimal version number for the dh_makeclilibs call.

6.2.6 GAC Policy File Example

```
<configuration>
   <runtime>
      <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
       <dependentAssembly>
         <assemblyIdentity name="foo" publicKeyToken="35e10195dab3c99f" />
         <bindingRedirect oldVersion="1.2.0.0-1.2.10.0" newVersion="1.3.0.0"/>
       </dependentAssembly>
      </assemblyBinding>
   </runtime>
</configuration>
```

The above example would be used for a policy file for the "foo" assembly and would tell the GAC that version 1.3.0.0 is compatible with versions 1.2.0.0 to 1.2.10.0. You have to compile and install it with

```
al -link:policy.1.2.foo.config -out:policy.1.2.foo.dll -keyfile:path/to/keyfile
gacutil /i policy.1.2.foo.dll
```

Keep in mind that the filenames *must* be policy.X.Y.foo.config and policy.X.Y.foo.dll where foo is the assembly name and X.Y is the major and minor version number you want to be compatible with.

6.3 Migrating Existing Packages

Many CLI packages already exist in Debian, or are in ITP, and conform to the deprecated Mono Conventions (`http://wiki.debian.org/?MonoConventions`).

Any `debian/rules` hacks or patches that exist to redirect files to `/usr/share/dotnet` should be removed, and adjusted according to upstream file locations (`/usr/lib`). See Mono Debian Plan (`http://wiki.debian.org/?MonoDebianPlan`) for the rationale behind this change.

Also, be sure to replace references to dh_netdepends, dh_makenetlibs, and ${net:Depends} with the newer names described in the policy above.

Please remove any build-deps on `mono-jit`, `mono-mint`, `mono-utils` (this one had the dh_* helper scripts which are now in `cli-common-dev`) and `libmono-dev` (use this one only if the package really links against `mono` or requires the mono.pc file).