GStreamer FAQ (0.8.5)

This is the FAQ for GStreamer, a multimedia framework. Questions and answers range from general information to deep-down-and-dirty compilation issues.

1. Getting Started

So you're eager to get started learning about GStreamer. There's a few ways you can get started.

- If you want to learn by reading about it, start with *General*
- If you'd rather learn by trying it out, start with *Getting GStreamer*
- If you want to live on the bleeding edge and develop and use CVS, see *Building GStreamer from CVS*

2. General

1. Is GStreamer a media player ?

No, GStreamer is a development framework for creating applications like media players, video editors, streaming media broadcasters and so on. That said, very good media players can easily be built on top of GStreamer and we even include a simple yet functional media player with GStreamer, called gst-player.

2. Why is GStreamer written in C ? Why not C++/Objective-C/... ?

We like C. Aside from "personal preference", there are a number of technical reasons why C is nice in this project:

- C is extremely portable.
- C is fast.
- It is easy to make language bindings for libraries written in C.
- The GObject object system provided by GLib implements objects in C, in a portable, powerful way. This library provides for introspection and runtime dynamic typing. It is a full OO system, but without the syntactic sugar. If you want sugar, take a look at GOB (http://www.5z.com/jirka/gob.html).
- Use of C integrates nicely with Gtk+ and GNOME. Some people like this a lot, but neither Gtk+ nor GNOME are required by GStreamer.

So, in closing, we like C. If you don't, that's fine; if you still want to help out on GStreamer, we always need more language binding people. And if not, don't bother us; we're working :-)

3. What applications are available for GStreamer ?

GStreamer is still very early in its development, but already we see some really nice applications being developed in parallel with GStreamer. Both gst-player and gst-editor are very closely linked to GStreamer itself for obvious reasons.

4. Does GStreamer support the format of my media files?

GStreamer aims to support every format imaginable, but that doesn't mean the developers have managed to achieve that aim yet. If a GStreamer enabled application doesn't play back your files, you can help us solve that problem by filing an enhancement request bug (htpp://bugzilla.gnome.org) for that format. If you have it, please provide:

- links to other players, preferrably Open Source and working on Unix
- links to explanations of the format.
- ways to obtain mediafiles in that format to test.

5. What are the exact licensing terms for GStreamer and its plugins ?

All of GStreamer, including our own plugin code, is licensed under the GNU LGPL (http://www.gnu.org/licenses/lgpl.html) license. Some of the libraries we use for some of the plugins are however under the GPL, which means that those plugins can not be used by a non-GPL-compatible application.

As part of the GStreamer source download you find a file called LICENSE_readme in gst-plugins package. That file contains information in the exact licensing terms of the libraries we use. As a general rule, GStreamer aims at using only LGPL or BSD licensed libraries if available and only use GPL or proprietary libraries where no good LGPL or BSD alternatives are available.

From GStreamer 0.4.2 on, we implemented a license field for all of the plugins, and in the future we might have the application enforce a stricter policy (much like tainting in the kernel).

6. Is GStreamer a sound server ?

No, GStreamer is not a soundserver. GStreamer does however have plugins supporting most of the major soundservers available today, including ESD, aRTSd, and to some extent Jack. Support for MAS is also planned.

7. Will GStreamer be available for platforms other than Unix ?

Depends. Our main target is the Unix platform. That said, interest has been expressed in porting GStreamer to other platforms and the GStreamer core team will gladly accept patches to accomplish this.

8. What is GStreamer's relationship with the GNOME community ?

While GStreamer is operated as an independent project, we do have a close relationship with the GNOME community. Many of our hackers consider themselves also to be members of the GNOME

community. There are plans to make (some part of) GStreamer an official part of the development framework of GNOME. This does not exclude use of GStreamer by other communities at all, of course.

9. What is GStreamer's relationship with the KDE community ?

The GStreamer community wants to have as good a relationship as possible with KDE, and we hope that someday KDE decides to adopt GStreamer as their multimedia API, just like the GNOME community plans on doing. There have been contacts from time to time between the GStreamer community and KDE and we do already have support for the aRTSd sound server used by KDE. Also, some of the KDE hackers have created Qt bindings of GStreamer and made a simple video player.

10. I'm considering adding GStreamer output to my application...

That doesn't really make sense. GStreamer is not a sound server, so you don't output directly to GStreamer, and it's not an intermediate API between audio data and different kinds of audio sinks. It is a fundamental design decision to use GStreamer in your app; there are no easy ways of somehow 'transfering' data from your app to GStreamer. Instead, your app would have to use or implement a number of GStreamer elements, string them together, and tell them to run. In that manner the data would all be internal to the GStreamer pipeline.

That said, it is possible to write a plugin specific to your app that can get at the audio data.

3. Dependencies

1. Why are there so many dependencies ?

Making a full-featured media framework is a huge undertaking in itself. By using the work done by others, we both reduce the amount of redundant work being done and leave ourselves free to work on the architecture itself instead of working on the low-level stuff. We would be stupid not to reuse the code others have written.

However, do realize that in no way you are forced to have all dependencies installed. None of the core developers has all of them installed. GStreamer has only a few obligate dependencies : GLib 2.0, popt >= 1.6.0, and very common stuff like glibc, a C compiler, and so on. All of the other dependencies are optional.

So, in closing, let's rephrase the question to "Why are you giving me so many choices and such a rich environment ? "

2. Does GStreamer use GTK+ 1.2/GLib 1.2 or GLib 2.0 ?

Since the 0.3.3 release of GStreamer, we use GLib 2.0 as the core library for GStreamer, which features a move of GObject from GTK+ 2.0 to GLib 2.0. If you want to compile using GTK+ 1.2/GLib 1.2, you need to get the 0.3.1 or earlier release. It is of course not supported.

3. Does GStreamer offer support for DVD decoder cards like dxr2/3 ?

We do have support for the dxr3, although dxr2 support is unknown. GStreamer can easily accomodate hardware acceleration by writing new device-specific elements.

4. Is GStreamer X independent ?

Yes, we have no X dependency in any of our core modules. There are GStreamer applications that run fine without any need for X. However, until our Linux Framebuffer or libsvga plugin is ready, you will not be able to play videos without X. In the future, there will probably be lots of different output plugins for video available.

5. What is GStreamer's position on efforts such as LADSPA ?

GStreamer actively supports such efforts, and in the case of *LADPSA* (http://ladspa.org/), we already have a wrapper plugin. This wrapper plug-in detects the LADSPA plugins present on your system at register time.

6. Does GStreamer support MIDI ?

Not yet. The GStreamer architecture should be able to support the needs of MIDI applications very well however. If you are a developer interested in adding MIDI support to GStreamer we are very interested in getting in touch with you.

7. Does GStreamer depend on GNOME ?

No. But many of the applications developed for GStreamer do, including our sample applications. There is nothing hindering people from developing applications using other toolkits however and we would happily help promote such efforts. A good example of an application using GStreamer, but which is not using GNOME is the *Mozstreamer* (http://mozstreamer.mozdev.org) which uses Mozilla XUL.

4. Getting GStreamer

1. How do I get GStreamer ?

Generally speaking, you have three options, ranging from easy to hard :

- distribution-specific packages
- source tarballs
- CVS

2. How can I install GStreamer from source ?

We provide tarballs of our releases on our own site, at http://gstreamer.freedesktop.org/src/ (http://gstreamer.freedesktop.org/src/)

When compiling from source, make sure you specify PKG_CONFIG_PATH correctly when building against GStreamer. For example, if you configured GStreamer with the default prefix (which is /usr/local), then you need to

```
export   PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
```

before building gst-plugins.

3. Are there premade binaries available ?

Yes, we currently provide precompiled packages for Red Hat, Debian and Linux Mandrake.

We provide RPMS for Red Hat and Fedora Core through our Apt for rpm page. (http://gstreamer.freedesktop.org/pkg/) We usually support the last 2-3 releases of Red Hat. (RH9 and FC1)

GStreamer is already in Debian experimental, so if you are a debian user you should be able to get the Debian packages from there.

GStreamer is also in Mandrake Cooker so if you are using a recent release of Linux Mandrake you should be able to get GStreamer from there. To learn howto get packages from Mandrake cooker the Mandrake cooker page has more info (http://www.mandrakelinux.com/en/cookerdevel.php3)

For other RPM based distributions we recommend getting the source tarball and doing 'rpm -ta gstreamer-0.X.X' to create rpms. We keep our SPEC file constantly up to date so you should always be able to build GStreamer rpms from a tarball. The SPEC file is aimed at Red Hat however, so there might be some need for you to edit the Requirements list if your distribution name these packages differently.

4. Why don't you provide premade binaries for distribution XY ?

GStreamer is run on a volunteer basis. The package that are provided are made by non-paid people who do this on their own time. The distributions we support with binaries are the distributions that we have people who have volunteered to make binaries for. If you are interested in maintaining GStreamer binaries for other distributions or Unices we would be happy to hear from you. Contact us through the GStreamer-devel mailing list.

5. I am having trouble compiling GStreamer on my LFS installation, why ?

If you are running LFS our basic opinion is that you should be knowledgeable enough to solve any build issues you get on your own. Being volunteered based we can't promise support to anyone of course, but are you using LFS consider yourself extra unsupported. We neither can or want to know enough, about how your unique system is configured, to be able to help you. That said, if you come to the #gstreamer channel on irc.openprojects.net we might of course be able to give you some general hints and pointers.

6. How do I get GStreamer through CVS ?

see this page : http://gstreamer.freedesktop.org/dev/ for CVS access. (anonymous and developer)

5. Using GStreamer

1. Ok, I've installed GStreamer. What can I do next ?

First of all, verify that you have a working registry and that you can inspect them by typing

```
$ gst-inspect    fakesrc
```

This should print out a bunch of information about this particular element. If this tells you that there is "no such element or plugin", you haven't installed GStreamer correctly. Please check how to get GStreamer If this fails with any other message, we would appreciate a bug report.

It's time to try out a few things. Start with gst-launch and two plug-ins that you really should have : fakesrc and fakesink. They do nothing except pass empty buffers. Type this at the command-line :

```
$ gst-launch    fakesrc    num-buffers=3    ! fakesink
```

This will print out output that looks similar to this :

```
RUNNING   pipeline
fakesrc0:    last-message    = "get        *******   (fakesrc0:src)gt;     (0  bytes,   0)  0x8057510"
fakesink0:    last-message    = "chain     *******   (fakesink0:sink)lt;      (0  bytes,   0)  0x8057510"
fakesrc0:    last-message    = "get        *******   (fakesrc0:src)gt;     (0  bytes,   1)  0x8057510"
fakesink0:    last-message    = "chain     *******   (fakesink0:sink)lt;      (0  bytes,   1)  0x8057510"
fakesrc0:    last-message    = "get        *******   (fakesrc0:src)gt;     (0  bytes,   2)  0x8057510"
fakesink0:    last-message    = "chain     *******   (fakesink0:sink)lt;      (0  bytes,   2)  0x8057510"
execution   ended   after   5 iterations   (sum   301479000   ns,   average   60295800   ns,   min   3000   ns,   max
```

(Some parts of output have been removed for clarity) If it looks similar, then GStreamer itself is running correctly.

2. Can my system play sound through GStreamer ?

You can test this by trying to play a sine tone. For this, you need to link the sinesrc plug-in to an output plug-in that matches your hardware. A (non-complete) list of output plug-ins for audio is

- osssink for OSS output
- esdsink for ESound output
- artssink for aRTs output
- alsasink for ALSA output
- jacksink for JACK output

First of all, run gst-inspect on the output plug-in you want to use to make sure you have it installed. For example, if you use OSS, run

```
$ gst-inspect    osssink
```

and see if that prints out a bunch of properties for the plug-in.

Then try to play the sine tone by running

```
$ gst-launch    sinesrc   ! osssink
```

and see if you hear something. Make sure your volume is turned up, but also make sure it is not too loud and you are not wearing your headphones.

3. How can I see what GStreamer plugins I have on my system ?

To do this you use the gst-inspect command-line tool, which comes standard with GStreamer. Invoked without any arguments,

```
$ gst-inspect
```

will print out a listing of installed plugins. To learn more about a particular plugin, pass its name on the command line. For example,

```
$ gst-inspect    volume
```

will give you information about the volume plugin.

Also, if you install the gst-editor package, you will have a graphical plugin browser available, gst-inspect-gui.

4. Where should I report bugs ?

Bug management is now hosted on GNOME's Bugzilla at http://bugzilla.gnome.org, under the product GStreamer. Using bugzilla you can view past bug history, report new bugs, etc. Bugzilla requires you to make an account here, which might seem cumbersome, but allows us to at least have a chance at contacting you for further information, as we will most likely have to.

5. How should I report bugs ?

When doing a bug report, you should at least describe

- your distribution
- how you installed GStreamer (from cvs, source, packages, which ?)
- if you installed GStreamer before

It also is useful for us if you attach output of the gst-feedback command to your bug report. If you're having problem with a specific application (either one of ours, somebody else's, or your own), please also provide a log of gst-mask by running

```
myapp   --gst-mask=-1    > mask.log   2>&1
gzip  mask.log
```

(interrupting the program if it doesn't stop by itself) and attach mask.log.gz to your bug report.

If the application you are having problems with is segfaulting, then provide us with the necessary gdb output. See *The GStreamer application I used stops with a segmentation fault. What can I do ?*

6. How do I use the GStreamer command line interface ?

You access the GStreamer command line interface using the command gst-launch. To decode an mp3 and play it through OSS, you could use

```
gst-launch  filesrc  location=thesong.mp3  ! mad  ! osssink
```

. More examples can be found in the gst-launch man page.

To automatically detect the right codec in a pipeline, try

```
gst-launch  filesrc  location=my-random-media-file.mpeg  ! spider  ! osssink
```

. Try replacing osssink with sdlvideosink and see what happens.

We also have a simple tool called gst-launch-ext used for debugging, which has predefined pipelines for you. This means you can just write

```
gst-launch-ext  (filename)
```

and it will play the file if the extension is supported. Note that no effort has been made for uninterrupted synchronized playback using this tool.

6. Troubleshooting GStreamer

1. My GStreamer-based application crashes on startup with errors about unfound schedulers on the command-line. I get undefined behaviour as soon as any GStreamer element is being initialized.

Your registry is probably missing, or it is outdated (i.e. not updated after a recent upgrade). Fix this by running gst-register yourself:

```
gst-register
```

In the worst case, you might have to run it both as user and as root.

Note that package managers are suggested to run this automatically during the post-installation. Our RPMs and Debian packages do just that.

2. Some application is telling me that I am missing a plug-in. What do I do ?

Well, start by checking if you really are missing the plug-in.

```
gst-inspect  (plug-in)
```

and replace (plug-in) with the plug-in you think is missing. If this doesn't return any result, then you either don't have it or your registry cannot find it.

If you're not sure either way, then chances are good that you don't have it. You should get the plug-in and run gst-register to register it. How to get the plug-in depends on your distribution.

- if you run GStreamer using packages for your distribution, you should check what packages are available for your distribution and see if any of the available packages contains the plug-in.
- if you run GStreamer from a source install, there's a good chance the plug-in didn't get built because you are missing an external library. When you ran configure, you should have gotten output of what plug-ins are going to be built. You can re-run configure to see if it's there. If it isn't, there is a good reason why it is not getting built. The most likely is that you're missing the library you need for it. Check the README file in gst-plugins to see what library you need. Make sure to remember to re-run configure after installing the supporting library !
- if you run GStreamer from CVS, the same logic applies as for a source install. Go over the reasons why the plug-in didn't get configured for build. Check output of config.log for a clue as to why it doesn't get built if you're sure you have the library needed installed in a sane place.

3. I get an error that says something like (process:26626): GLib-GObject-WARNING **: specified instance size for type 'DVDReadSrc' is smaller than the parent type's 'GstElement' instance size What's wrong ?

If you run GStreamer CVS uninstalled, it means that something changed in the core that requires a recompilation in the plugins. Recompile the plugins by doing "make clean && make".

If you run GStreamer installed, it probably means that you run the plugins against a different (incompatible) version than they were compiled against, which ususally means that you run multiple installations of GStreamer. Remove the old ones and - if needed - recompile again to ensure that it is using the right version.

Note that we strongly recommend using Debian or RPM packages, since you will not get such issues if you use provided packages.

4. The GStreamer application I used stops with a segmentation fault. What can I do ?

There are two things you can do. If you compiled GStreamer with specific optimization compilation flags, you should try recompiling GStreamer, the application and the plug-ins without any optimization flags. This allows you to verify if the problem is due to optimization or due to bad code. Second, it will also allow you to provide a reasonable backtrace in case the segmentation fault still occurs.

The second thing you can do is look at the backtrace to get an idea of where things are going wrong, or give us an idea of what is going wrong. To provide a backtrace, you should

1. run the application in gdb by starting it with

```
gdb (gst-application)
```

(If the application is in a source tree instead of installed on the system, you might want to put "libtool" before "gdb")

2. Pass on the command line arguments to the application by typing

```
set args (the arguments to the application)
```

at the (gdb) prompt

3. Type "run" at the (gdb) prompt and wait for the application to segfault. The application will run a lot slower, however.
4. After the segfault, type "bt" to get a backtrace. This is a stack of function calls detailing the path from main () to where the code is currently at.
5. If the application you're trying to debug contains threads, it is also useful to do

```
info threads
```

and get backtraces of all of the threads involved, by switching to a different thread using "thread (number)" and then again requesting a backtrace using "bt".

6. If you can't or don't want to work out the problem yourself, a copy and paste of all this information should be included in your bug report.

7. Building GStreamer from CVS

1. How do I check out GStreamer from CVS ?

GStreamer is hosted on Freedesktop.org. GStreamer consists of various parts. In the beginning, you will be interested in the "gstreamer" module, containing the core, and "gst-plugins", containing the basic set of plugins.

To check out the HEAD version of the core, use

```
cvs -d:pserver:anoncvs@cvs.freedesktop.org:/cvs/gstreamer co gstreamer
```

This will create a directory "gstreamer" in your current directory. If you want to get another module, replace the last "gstreamer" with the name of the module.

2. How do I get developer access to GStreamer CVS ?

If you want to gain developer access to GStreamer CVS, you should ask for it on the development lists, or ask one of the maintainers directly. If you are not already a registered developer with a user account on Freedesktop.org, You will then have to provide them with:

1. your desired unix username
2. your full name
3. your e-mail address
4. a copy of your public sshv2 identity. If you do not have this yet, you can generate it by running "ssh-keygen -t dsa". The resulting public key will be in .ssh/id_dsa.pub

3. I ran autogen.sh, but it fails with something like this:

```
+ running  aclocal  -I  m4  -I  common/m4  ...
aclocal:  configure.ac:  8:  macro  'AM_DISABLE_STATIC'  not  found  in  library
aclocal:  configure.ac:  17:  macro  'AM_PROG_LIBTOOL'  not  found  in  library
aclocal  failed
```

What's wrong ?

aclocal is unable to find two macros installed by libtool in a file called libtool.m4. Normally this would indicate that you don't have libtool, but that would mean autogen.sh would have failed on not finding libtool.

It is more likely that you installed automake (which provides aclocal) in a different prefix than libtool. You can check this by examining in what prefix both aclocal and libtool are installed.

You can do three things to fix this :

1. install automake in the same prefix as libtool
2. force use of the automake installed in the same prefix as libtool by using the --with-automake option
3. figure out what prefix libtool has been installed to and point aclocal to the right location by running

   ```
   export  ACLOCAL_FLAGS="-I  $(prefix)/share/aclocal"
   ```

   where you replace prefix with the prefix where libtool was installed.

8. Developing applications with GStreamer

1. How do I compile programs that use GStreamer ?

GStreamer uses pkg-config to assist applications with compilationa and linking flags. pkg-config is already used by GTK+, GNOME, SDL, and others; so if you are familiar with using it for any of those, you're set.

If you're not familiar with pkg-config to compile and link a small one-file program, pass the --cflags and --libs arguments to pkg-config. For example:

```
$ libtool --mode=link gcc `pkg-config --cflags --libs gstreamer-0.8` -o myprog myprog.c
```

would be sufficient for a gstreamer-only program. If (for example) your app also used GTK+ 2.0, you could use

```
$ libtool --mode=link gcc `pkg-config --cflags --libs gstreamer-0.8 gtk+-2.0` -o myprog myprc
```

Those are back-ticks (on the same key with the tilde on US keyboards), not single quotes.

For bigger projects, you should integrate pkg-config use in your Makefile, or integrate with autoconf using the pkg.m4 macro.

2. How do I develop against an uninstalled GStreamer copy ?

It is possible to develop and compile against an uninstalled copy of gstreamer and gst-plugins (for example, against CVS copies). The easiest way to do this is to use a script like this (for bash):

```
#!/bin/bash -i
#
# this script is in CVS as gstreamer/docs/faq/gst-uninstalled
#
# set up environment to use and develop gstreamer and friends uninstalled
#
# set up PATH, LD_LIBRARY_PATH, PKG_CONFIG_PATH, GST_PLUGIN_PATH, MANPATH,
# PYTHONPATH
#
# prefer uninstalled versions, but also put installed ones on the path
#
# this script assumes that the relevant modules are checked out one by one
# under a given tree specified below in MYGST
#
# symlink this script in a directory in your path (for example $HOME/bin)
# to a name that reflects the version of your checkout
#
# e.g.:
# - mkdir $HOME/gst/head
# - ln -sf gst-uninstalled $HOME/bin/gst-head
# - checkout copies of gstreamer modules in $HOME/gst/head
# - gst-head

# this script is run -i so that PS1 doesn't get cleared

# change this variable to a different location depending on where you
# store your cvs checkouts
MYGST=$HOME/gst

# extract version from $0
# if this script is called "gst-head" then version will be "head"
```

```
VERSION=‘echo    $0 | sed s/.*gst-//g‘

# base path under which dirs are installed
GST=$MYGST/$VERSION
if test ! -e $GST; then
  echo "$GST does not exist !"
  exit
fi

# set up a bunch of paths
PATH=$GST/gstreamer/tools:$GST/gst-plugin          s/tool s:$GST /gst-p layer/ src:$ GST/gs t-edit or/src :$GST
# /some/path:    makes the dynamic linker look in . too, so avoid this
export LD_LIBRARY_PATH=$GST/prefix/lib${LD_LIBR          ARY_PA TH:+:$ LD_LIB RARY_ PATH}
export PKG_CONFIG_PATH=$GST/prefix/lib/pkgconfi          g:$GST /gstre amer/p kgcon fig:$G ST/gst -plugi ns/pk
export GST_PLUGIN_PATH=$GST/gstreamer:$GST/gst-          plugin s:$GST /gst-f fmpeg :$GST/ gst-mo nkeysa udio
export MANPATH=$GST/gstreamer/tools:$GST/prefix          /share /man:$ MANPAT H
pythonver=‘python     -c "import sys; print sys.version[:3]"‘
export PYTHONPATH=$GST/gst-python:$GST/prefix/l          ib/pyt hon$py thonve r/sit e-pack ages:$ PYTHON PATH

# if we got a command, run it, else start a shell
if test ! -z "$1";
then
  $@
  exit $?
fi

# set up prompt to help us remember we’re in a subshell, cd to
# the gstreamer base dir and start $SHELL
cd $GST
PS1="[gst-$VERSION]     $PS1"  $SHELL
```

If you put this script in your path, and symlink it to gst-cvs (if you want to develop against cvs HEAD) or to gst-0.6 (if you want to develop against the 0.6 branch), it will automatically use the uninstalled version from that directory.

This requires you to have put your checkouts of gstreamer and gst-plugins under ~/gst/cvs (for the HEAD version). The program is easily modifiable if this isn’t the case.

After running this script, you’ll be in an environment where you can use the uninstalled tools, and where gst-register registers the uninstalled plugins by default. Also, pkg-config wil detect the uninstalled copies before any installed copies.

3. How can I use GConf to get the system-wide defaults ?

It’s a good idea to use GConf to use default ways of outputting audio and video. Since GStreamer’s GConf keys can be more than just one element, but a whole pipeline, it would be a good idea to use the gstgconf library. It provides functions to parse the GConf key to a usable pipeline.

To link against gstgconf, use pkg-config to query the gstreamer-libs-0.8.pc file for link flags, and add -lgstgconf to the link flags. This fragment of configure.ac shows how to use pkg-config to get the LIBS:

```
dnl check for GStreamer helper libs
PKG_CHECK_MODULES(GST_HELPLIBS, gstreamer-libs-0.8 >= $GSTREAMER_REQ,,exit)
AC_SUBST(GST_HELPLIBS_LIBS)
AC_SUBST(GST_HELPLIBS_CFLAGS)
```

This fragment of a Makefile.am file shows how to make your application link to it:

```
bin_PROGRAMS = application

application_LDADD = $(GST_LIBS) $(GST_HELPLIBS_LIBS) -lgstgconf
application_CFLAGS = $(GST_CFLAGS) $(GST_HELPLIBS_CFLAGS)
```

4. How do I debug these funny shell scripts that libtool makes ?

When you link a program against uninstalled GStreamer using libtool, funny shell scripts are made to modify your shared object search path and then run your program. For instance, to debug gst-launch, try

```
libtool gdb /path/to/gstreamer-launch
```

. If this does not work, you’re probably using a broken version of libtool.

5. Why is mail traffic so low on gstreamer-devel ?

Our main arena for coordination and discussion is IRC, not email. Join us in #gstreamer on irc.freenode.net For larger picture questions or getting more input from more persons, a mail to gstreamer-devel is never a bad idea. However, we do archive our IRC discussions, which you may find in the gstreamer-daily mailing list archives.

6. What kind of versioning scheme does GStreamer use ?

For public releases, GStreamer uses a standard MAJOR.MINOR.MICRO version scheme. If the release consists of mostly bug fixes or incremental changes, the MICRO version is incremented. If the release contains big changes, the MINOR version is incremented. If we’re particularly giddy, we might even increase the MAJOR number. Don’t hold your breath for that though.

During the development cycle, GStreamer also uses a fourth or NANO number. If this number is 1, then it’s a CVS version. Any tarball or package that has a nano number of 1 is made from CVS and thus not supported. Additionally, if you didn’t get this package or tarball from the GStreamer team, don’t have high hopes on it doing whatever you want it to do.

If the number is 2 or higher, it’s an official pre-release in preparation of an actual complete release. Your help in testing these tarballs and packages is very much appreciated.

7. What is the coding style for GStreamer core ?

The core is basically coded in K&R with 2-space indenting. Just follow what's already there and you'll be fine. The core could use a code cleanup though at this point.

Individual plugins in gst-plugins or plugins that you want considered for addition to the gst-plugins module should be coded in the same style. It's easier if everything is consistent. Consistency is, of course, the goal.

If you use emacs, try these lines:

```
(defun gstreamer-c-mode ()
  "C mode with adjusted defaults for use with GStreamer."
  (interactive)
  (c-mode)
  (c-set-style "K&R")
  (setq c-basic-offset 2))

(setq auto-mode-alist (cons '("gst. */.*\\.[ch]$" . gstreamer-c-mode)
                            auto-mode-alist))
```

Or, run your code through

```
indent -br -bad -cbi0 -cli2 -bls -l100 -ut -ce
```

before submitting a patch (FIXME: check if these are indeed the proper options).

As for the code itself, the GNOME coding guidelines (http://developer.gnome.org/doc/guides/programming-guidelines/book1.html) is a good read. Where possible, we try to adhere to the spirit of GObject and use similar coding idioms.