GNU Bayonne Installation Guide

David Sugar
GNU Telephony.
sugar@gnu.org, http://www.gnutelephony.org

2005-06-03

Contents

1 Introduction

This guide was created to assist those trying to build and install the second edition of GNU Bayonne (to be referred to as GNU Bayonne 2) for the first time. GNU Bayonne 2 is a large and complex package, which can involve both hardware and software, as well as a number of additional software packages, each of which my need to be configured to successfully deploy a running server.

GNU Bayonne 2 itself is licensed as free software under the GNU General Public License. This means the source for this complete package is made available to the user, and the user is given specific rights to use and modify the sources provided which are often denied in proprietary packages. GNU Bayonne itself depends on other parts of the GNU system, such as GNU Common C++ and GNU osip, which are also licensed as free software. This guide does not try to explain how to fully exercise these rights, but simply how to configure and install GNU Bayonne 2 as distributed through the GNU project.

GNU Bayonne 2 is a fully portable server and may be used under a large number of different 32 and 64 bit operating systems, and is not bound to any machine architecture. Hence, GNU Bayonne 2 may be used with machines that have Sparc,

PowerPC, S/370 chips, or other architectures equally as well as on Intel based commodity PC hardware. While GNU Bayonne 2 is very portable, this manual only tries to cover installation on "posix" operating system, such as GNU/Linux, xBSD, Solaris, or MAC OS/X. The information here is not directly relevant to other platforms which GNU Bayonne 2 may also support, such as Microsoft Windows, QNX, or specialized realtime and embedded systems. Installation instructions specific to Microsoft Windows will be covered in a special guide which will accompany the GNU Telephony "cape" installer.

2 Building GNU Bayonne

While GNU Bayonne 2 is a very large and complex service, it is built and installed in a manner that is no different from other GNU packages. GNU Bayonne 2 uses a "configure" script that is generated from autoconf and uses automake to generate Makefiles. This section will cover the basics to get you started, while the remaing sections will help you in building GNU Bayonne 2 based on specific configurations and platforms.

2.1 Before you begin

GNU Bayonne 2 differs from most software packages and services one may deploy on GNU/Linux systems in one key respect; to be useful, it may require computer telephony hardware. This hardware can take many forms, and comes in different telephone voice line capacities, as well as support for many different types and forms of telephone circuit interconnection. Bayonne also may be used in a "testing" for using just a soundcard, and can be used without physical hardware as a IP telephony application server. Most commonly, we assume GNU Bayonne 2 will be used as a voice application server that sits behind an existing telephone switch such as an enterprise PBX system or IP voice gateway. However, GNU Bayonne 2 can also be connected and used with the public telephone network directly.

With these wide ranges of uses, all of which can depending on different hardware, with different characteristics and capabilities, the first choice one has to make is in understanding how or for what purpose one will use GNU Bayonne

2 for. The second choice is if one is building a traditional wired telephony application server. If so then one must also choose which vendor's hardware one will use with it, based on the intended deployment and capability or suitability of different vendor's computer telephony hardware which GNU Bayonne 2 supports.

If you already understand how you intend to use GNU Bayonne 2 and already have the optional computer telephony hardware that you plan to use, it is strongly suggested that you skip ahead to the subsection of this manual that describes your hardware before going through the next few sections. In the hardware sections of this manual you will find information on what steps must be done to configure and install the drivers and/or libraries that come with your given hardware. These steps must often be done BEFORE you begin installation of GNU Bayonne 2 itself.

If you have not selected your computer telephony hardware, you can get a good idea of what hardware is supported from each vendor under GNU Bayonne 2 that might meet your needs by reviewing the different hardware sections. However, we do not make any recommendations for use of any given comperable hardware solution from any specific vendor as being preferred over any other vendor. There is neither product reviews or comparisons in the hardware sections, just information on known issues of compatability, configuration, installation.

Once you have installed your computer telephony hardware and or support for IP voice protocol stacks, and have verified that it is functionally working, as outlined in the relevant sections, you should proceed to the following section on building the server.

2.2 Bayonne User and Group

One of the first things you should do is create a "bayonne" user and "group" account. The "bayonne" user account is the user account that the Bayonne server will run under when started by "root". This can be created as a system account using something like "adduser -r bayonne". The "bayonne" group is used so that you can assign the device nodes of your computer telephony hardware under group permission with rw access under "bayonne", and then assign individual user accounts you may wish to access your computer telephony cards under to the "bayonne" group.

You do not need to use a user or group named "bayonne" for this purpose; you could run bayonne under the same group and user id that apache uses, for example. The group and user that bayonne runs under is set in /etc/bayonne/server.conf, and may be changed from it's default of "bayonne", or left blank, in which case the server, when started under root, will remain running under root. However, its generally advised to remove root permission from Bayonne in case of possible security issues in the code.

The other reason to use a group assignment of "bayonne" and to associate user accounts with the group is that Bayonne creates control and data files that may be accessed under group "bayonne". Hence, if you want to administer your running Bayonne server that is started by root, you can selectivily enable other user accounts to do so by adding them to the "bayonne" group, rather than having to do a su or login to the "bayonne" user.

2.3 Building from Source

Before you can build and use GNU Bayonne 2, you must build and install GNU Common C++ 2, GNU ccAudio 2, and GNU ccScript 3. These packages are also part of the GNU system and may be downloaded from `ftp://ftp.gnu.org/` or from any GNU mirror site. If you are using the Bayonne SIP driver, you will also need GNU ccrtp, osip, and exosip. However the first package you should download and build is GNU Common C++, and you should download the latest release version, which will be found in `ftp://ftp.gnu.org/gnu/commoncpp`.

After downloading GNU Common C++, you can unpack it with:

```
tar -zxvf commoncpp2-{VERSION}.tar.gz
```

Substitute VERSION with the version of your package.

To install GNU Common C++, enter the commoncpp2 directory you have unpacked to and run the "configure" script. For GNU/Linux systems, this script

can be ran simply with ./configure, and GNU Common C++ will be installed to your /usr/local directory. If you wish to install GNU Common C++ to /usr, then enter ./configure −−prefix=/usr. Once configure completes, simply perform a "make install".

Next, you should download GNU current versions of GNU ccScript 3 (ccscript3) from `ftp://ftp.gnu.org/gnu/ccscript` and the GNU ccAudio 2 (ccaudio2) package from `ftp://ftp.gnu.org/gnu/ccaudio`. These packages depend on GNU Common C++ 2 being installed before they can be made. After downloading these packages, unpack each one similar to above. You will then run the "configure" script from each of these packages using the same options you choose to use when building GNU Common C++ 2 (if any). Then simply perform a "make install".

If you are going to use SIP, then you should download GNU current version of GNU ccrtp from `ftp://ftp.gnu.org/gnu/cccrtp`. This package depends on GNU Common C++ being installed before it can be made. After downloading this packages, unpack similar to above. You will then run the "configure" script from ccrtp using the same options you choose to use when running "configure" for GNU Common C++ (if any). Then simply perform a "make install".

Depending on your platform and needs, it will be useful to pre-install a number of other packages. For gsm audio processing in ccaudio2 and Bayonne, you will want to install libgsm and it's dev package (typically "apt-get install libgsm1-dev" on Debian based systems). For ODBC database connectivity in ccscript3 and Bayonne, you will want to install unixodbc (typically "apt-get install unixodbc-dev" on Debian systems). To generate class documentation, you will want doxygen installed. For creating Bayonne's pdf manuals you will need pdflatex, as found in the tetex extras package.

At this point if you plan to use specific Computer Telephony hardware, then you should also have installed any device drivers, header files, and link libraries that were part of or required by the computer telephony hardware you will be using. If you are using a CAPI based computer telephony card under GNU/Linux, then your distribution may already include the capi library needed for building GNU Bayonne and you need not have anything else installed. Many non-capi cards include link libraries and drivers that must be present for the card to work and to be controlled by external programs. Please follow the notes in the different hardware sections based on your hardware and make sure your hardware is work-

ing correctly before you download and install Bayonne.

You may now download GNU Bayonne 2 (bayonne2) from `ftp://ftp.gnu.org/gnu/bayonne`. Once you download the GNU Bayonne 2 package, you will unpack it as before, and you may then enter the package directory and run the "configure" script. The Bayonne configure can be passed a number of options that control what features will be enabled or disabled in GNU Bayonne 2. These can be used to tailor the server image for specific applications where only a select set of GNU Bayonne features and capabilities are required. A description of how to tailore GNU Bayonne 2 is described in the next section.

If you do not wish to tailor GNU Bayonne 2, and in most cases, it is unessisary to do so, then you can simply run the "configure" script as is with default options. The configure script will test to see what computer telephony support exists on your machine, and will then build GNU Bayonne with drivers for whatever computer telephony support it has found. You may then use "make" to compile GNU Bayonne 2. Before you perform "make install", review the section on installing your new server.

It is important to make sure the libraries required for your computer telephony hardware are present first because Bayonne's configure script automatically detects which computer telephony card libraries are present and selects which computer telephony drivers GNU Bayonne 2 will be built with. If Bayonne is installed before your computer telephony hardware and libraries, then it will likely fail to use your hardware since it did not detect it at configure time. However, you can always re-run the configure script in the package to enable GNU Bayonne 2 to build drivers for your card if you change hardware or forget to install your hardware first.

2.4 Compile-time configure options

There are a number of compile time options that can be selected by the "configure" script that are part of both GNU Bayonne 2 and GNU Common C++ 2. These options are specified through "configure" because they are options that must be selected before the package has been compiled, and cannot be altered except by re-compiling Bayonne. Many compile-time options relate to disabling specific features that you may choose not to use and thereby results in a smaller

executable image. Some compile time options enable specific features or support depending on additional optional packages being installed.

A number of options are offered in GNU Common C++ 2 to reduce the memory footprint it uses. If you only intend to use GNU Common C++ 2 with Bayonne, then these options disable GNU Common C++ features which Bayonne does not use though other applications might. The most useful of these include the "−−without-compression" option which removes libz support, the "−−without-ipv6" option which removes IPV6 socket support, the "−−without-libxml2" option which removes GNOME libxml2, and the "−−without-exceptions" option which removes C++ exception checking overhead.

GNU Bayonne 2 only uses libccgnu2 from GNU Common C++ 2. With an optimized build of GNU Common C++ and by selectivily installing modules, it should be possible to deploy Bayonne on GNU/Debian systems with 8 megs of ram or less. Future versions of GNU Common C++ may also offer an alternate "embedded" compile-time configure profile which will further reduce memory requirements by eliminating dependencies on libstdc++.

Currently Bayonne 2 is packaged with a special module which can be loaded to enable sunrpc support. As I found rpcgen is broken on some platforms, this is not enabled by defualt. Currently you can access it with the "−−enable-sunrpc" option in the Bayonne configure script. As the service is a runtime loadable plugin, and comes with a number of stand-alone binaries, in the future this may be made available as a seperate package that will be named bayonnerpc.

GNU Bayonne 2 also offers the "−−disable-libexec" option for its "configure" script. This option, when used, removes support from Bayonne to run external applications. This option is only recommended for systems with tight resource constraints, or where building servers for pre-bundled and dedicated applications.

Another option that is important is −−with-voices=xxx. This can be used to select libraries from which languages will be installed. By default, a generic english language voice language system is installed. To get russian and french added, for example, one would use −−with-voices="ru fr". Alternately, all of the languages found in a given distribution can be installed with −−enable-voices.

An importion configure and compile-time option is "−−disable-testing". This is used to disable support for build directory execution testing and other command line features and options that would not be used in a dedicated production environment. The result is a smaller, simpler runtime production server.

Other compile-time options may be offered in the future. Most options are now configured at runtime by selecting which modules will be loaded into your Bayonne server.

2.5 Using pre-built packages

Many GNU/Linux distributions include pre-compiled and pre-packaged versions of GNU Bayonne. GNU Bayonne 2 may also be found in the Debian unstable tree. On these systems, all you may need to do is select the Bayonne package and install it. From that point, all dependencies will often be installed automatically for you. For example, in a debian system, it may be possible to do an "apt-get install bayonne2". I have considered making seperate repositories supporting some distros I commonly use, but none have been setup as yet.

When installing pre-compiled copies of GNU Bayonne 2, these pre-built packages will only have support for whatever telephony hardware was available and detected on the machine that the package was originally built on. This may mean a pre-compiled Bayonne package will not work with your computer telephony hardware even if the hardware is listed as fully supported under Bayonne simply because the machine used to build the pre-packaged distribution of GNU Bayonne 2 did not have the link libraries for your hardware installed on it.

A simple solution is to rebuild GNU Bayonne 2 on your local machine from the source package supplied by your GNU/Linux vendor after installing your computer telephony hardware. By building a new binary package from the source package, you will have a new binary package which should include support for your hardware. You can then install your newly generated binary package and have it maintained by your package management system.

GNU Bayonne 2 and each of it's dependent packages also include their own spec files, and you can use these to locally create "rpm" packages for your own produc-

tion environment. These can be done by using the -tb option to have rpmbuild create a new binary package from a .tar.gz. When you do this, or rebuild the GNU Bayonne 2 package as noted above, however, you are restricted to the compile time configuration options that the packager originally defined. You can still modify these compile time options in the Bayonne .spec file and build installable binary packages tailored for your environment.

2.6 Building from cvs

If you want to retrieve Bayonne from CVS, you should start with the GNU Common C++ package. This can be retrieved from cvs by a network connected machine as follows:

```
cvs -d :pserver:anonymous@cvs.gnutelephony.org:/cvsroot/gnutelephony
login
```

```
cvs -z3 -d :pserver:anonymous@cvs.gnutelephony.org:/cvsroot/gnutelephony
co commoncpp2
```

For ccaudio2, you can use:

```
cvs -z3 -d :pserver:anonymous@cvs.gnutelephony.org:/cvsroot/gnutelephony
co ccaudio2
```

and for ccscript3,

```
cvs -z3 -d :pserver:anonymous@cvs.gnutelephony.org:/cvsroot/gnutelephony
co ccscript3
```

You will then enter your new module directory (commoncpp2) and run the special script command, "./reconfig". This will generate a new "configure" script to

use in configuring the package and do a "make install" as noted under building from source. You can then do the same for ccaudio2 and ccscript3.

Once GNU Common C++ is compiled and installed, you can use the following to retrieve ccrtp, osip2, and exosip from cvs if you plan to use SIP:

```
cvs -d :pserver:anonymous@subversions.gnu.org:/cvsroot/ccrtp login

cvs -z3 -d :pserver:anoncvs@subversions.gnu.org:/cvsroot/ccrtp co ccrtp

cvs -d :pserver:anonymous@subversions.gnu.org:/cvsroot/osip login

cvs -z3 -d :pserver:anoncvs@subversions.gnu.org:/cvsroot/osip co osip

cvs -d :pserver:anonymous@subversions.gnu.org:/cvsroot/exosip login

cvs -z3 -d :pserver:anoncvs@subversions.gnu.org:/cvsroot/exosip co exosip
```

You should then enter the ccrtp directory and perform a "./reconfig", followed by "./configure" and "make install". This will build and install ccrtp from cvs. For osip, you will cd into the osip directory and run "autogen.sh", which that package uses to generate a configure script, followed by running "configure". You can then do a "make install" for osip. For exosip, you will need to cd to exosip/exosip and then do the same as for osip.

Finally, once this is installed, you can do the same for GNU Bayonne 2. You can retrieve Bayonne from CVS from

```
cvs -z3 -d :pserver:anonymous@cvs.gnutelephony.org:/cvsroot/gnutelephony
co bayonne2
```

You will then run "./reconfig" in the bayonne2 directory to build your "configure" script.

With a cvs checkout of GNU Bayonne 2, you can perform updates to your installation through cvs itself. By entering each working cvs package directory and selecting "cvs update -Pd", the latest changes in cvs will be downloaded. You should then run "./reconfig", "./configure", and "make install" as you did originally, and starting from the commoncpp2 directory first.

You can also use the cvs to create patches for GNU Bayonne 2 or any of it's dependent packages. you can use "cvs diff" to create patches if you want to submit patches to the Bayonne mailing list (bayonne-devel@gnu.org). Don't forget to run "cvs update" before you start each work session!

3 Installing your new server

When building Bayonne from source, somewhere between performing "make" to compile Bayonne and "make install" there are a number of things you can do to determine that your newly compiled server is functional.

3.1 Testing your Bayonne Drivers

After compiling your new server, you can immediately test it to see if it is functional without having to do any further configuration. Bayonne can execute directly from the working directory you are building from using a special test mode. This test mode can also allow you to determine if your hardware or IP voice driver configuration is actually working and operates correctly with Bayonne even before Bayonne itself has been installed for production use.

GNU Bayonne test mode is activated by passing the "−−test" option when running the server. You can run the server and test applications delivered with GNU Bayonne without installing the server using "−−test". There is also a server test directory (called tests) which contains test scripts you can run to test various

driver features. You should cd to this directory and look at the various script files present here. They each contain comments which explain what they test and what you can expect when you run one.

To run a given test script with a driver, you can execute the shell script testit from this directory. To run the dtmf test script with the soundcard driver, for example, you would execute "./testit dtmf.scr". This basically runs "../server/bayonne −−test soundcard dtmf.scr". With the soundcard driver, the server will startup in test mode, and "wait" for an incoming call. This can be simulated by pressing the space bar. The soundcard driver is useful because you can use the keyboard to simulate dtmf input, and get a basic idea of server functionality very easily.

You can use test scripts to test any driver from this directory before installation. For example, to run the same test application for the dialogic driver, you can enter from the tests directory "../server/bayonne −−test dialogic test.scr". The server will start up in test mode and offer test.scr to the next incoming call received on your dialogic card.

Another useful option to use when testing the server from the build directory is −−debug. You would normally use this in place of the −−test option, as in, for example, "../server/bayonne −−debug dialogic test.scr". When used in debug mode, the server will fall into the gdb runtime debugger if it crashes. This can then be used to examine and debug the server.

3.2 Editing config files

Once Bayonne is installed, you will have to at minimum edit config files to indicate which driver you want the server to run when it is started. The simplest way to do this is to edit the /etc/sysconfig/bayonne file and set the DRIVER= line to specify the driver you are using. You may wish to set your default language and voice library settings in this file as well.

The actual location of the "bayonne" driver and base configuration file will vary based on your operating system and distribution. For example, /etc/sysconfig/bayonne is found on typical GNU/Linux systems that are based on the RedHat distribution. This same file may be /etc/conf.d/bayonne on a typical Debian or Gentoo

GNU/Linux system. On FreeBSD, this will be found as /etc/defaults/bayonne.conf. In addition, there is a directory, /etc/bayonne, which will hold all the configuration files used by the server once started.

The Bayonne server itself is normally ran as a daemon process, and may be started from your system initialization scripts using the "bayonne.init" script. This script is supported under chkconfig, and you can use "chkconfig bayonne on" to enable Bayonne services to run the next time your system is booted. You may also invoke the startup script to start the service now, in "/etc/rc.d/init.d/bayonne.init start".

Once you have the server installed and running, you will want to look at the administration guide. The admin guide includes information on how to schedule inbound calls with the scheduler file, and all of the many things found in the /etc/bayonne directory. That information is beyond the scope of this manual.

3.3 Installing on GNU/Linux

Generally Bayonne is most widely tested and deployed on GNU/Linux. If the target machine can compile GNU Bayonne 2, it will generally run fine. GNU Bayonne 2 is not dependent on kernels, glibc versions, or gcc. GNU Bayonne 2 uses threading extensivily, but supports both older Linux pthread support and the new NPTL threading found in 2.6 kernels (and early Fedora releases) just as well. Any modern GNU/Linux distribution should work fine, as well as most older distributions, including Debian Woody, RedHat 6.x and later, etc.

What you will find if you choose to use H323 support, is that recent OpenH323 releases may only compile or work correctly on certain GNU/Linux distributions. Also, the distributions of OpenH323 included in many GNU/Linux distributions predate 12.1 and will often not build correctly with GNU Bayonne. When this happens, it is generally recommended that you do NOT install the pre-packaged version of OpenH323 that came with your GNU/Linux distribution, but rather build and install OpenH323 from a current release tarball found on www.openh323.org.

Similarly, many of the computer telephony cards supported under GNU/Linux

either come with binary drivers, or drivers that only compile with specific kernel or glibc releases. These may also restrict your choice of distributions further.

GNU/Bayonne attempts to be compliant with FHS in the layout and use of system directories. This means if you know FHS or are using a LSB compliant GNU/Linux distribution, then GNU Bayonne will by default install things consistent in locations generally consistent with lsb & fhs as other packages do.

GNU Bayonne is also aware of the split between Debian based GNU/Linux distributions and so called "RedHat" based ones. On a "Redhat" based distribution, GNU Bayonne installs it's master system config file as /etc/sysconfig/bayonne, and it's init script under /etc/rc.d/init.d. On a Debian based system, the master system config file is found under /etc/conf.d/bayonne, and the init script is under /etc/init.d.

Generally I have tested GNU Bayonne 2 GNU/Linux support on non-x86 hardware. GNU Bayonne 2 audio processing is endian aware and it does compile on other common GNU/Linux target architectures such as ppc, s/390, etc. Availablility of computer telephony hardware usable on some of these targets may limit the ways GNU Bayonne 2 may be used on them, but for a pure IP voice server, any of these alternative platforms should prove workable and may offer some distinct advantages.

3.4 Installing under Mac OS/X

GNU Bayonne 2 works fine under recent versions of Mac OS/X. Many of the issues identified for GNU/Linux and BSD systems apply to Bayonne under OS/X as well. However, the only driver fully usable at this time is the SIP driver. If more recent versions of openh323 can successfully compile under OS/X, it may be possible to use the Bayonne H323 driver as well. The soundcard driver will not be usable until OSX audio support is fully completed in ccaudio2.

3.5 Installing under Free/Net/OpenBSD

Building and using GNU Bayonne 2 under BSD is no different than using a GNU/Linux distribution. GNU Bayonne 2 supports FreeBSD threads and reentrant libraries and generally behaves best under newer versions of BSD distributions (FreeBSD 5.3, OpenBSD 3.6, NetBSD 2.0, or later) which fully support native threads and SMP. However, there are issues; first, there is a lack of supported computer telephony hardware available on FreeBSD. This limitation will largely restrict one to using SIP or H323. The oss testing sound driver plugin does also work on FreeBSD, although there are some quirks in it's behavior on some BSD systems.

When building GNU Bayonne 2 on FreeBSD, many GNU/Linux specific characteristics are still retained, including the use of fhs file system layout. However, the GNU Bayonne 2 master config file is stored under /etc/defaults/bayonne.conf where it is expected for FreeBSD, and in the future we will offer a bayonnerc init script.

3.6 Installing under Solaris, AIX, HP/UX, etc..

In theory, at least, GNU Bayonne 2 should be able to compile and install under Sun Solaris, and make use of Solaris threading. In fact, any Unix target supported by posix threads (pthread, with or without sunos extensions) should be able to be used, including such systems as IBM AIX, HP/UX, etc. In the case of HP/UX, GNU Bayonne 2 will likely be compiled by the HP C++ compiler. In practice, none of these platforms have been tested by me directly, although there have been reports of people successfully building GNU Bayonne 2 under Solaris for use with the Solaris port of the quicknet ltapi driver. If you have success with these platforms, or have specific patches to offer to make them work, I would be interesting in knowing about this. It may also be true that GNU Bayonne 2 might happen to compile and/or run under Unixware, although no effort has been made to establish this and no effort will be made to support that platform.

3.7 Installing under Microsoft Windows

This would be covered in a seperate manual to be included in the future with the CAPE installer.

3.8 Notes on SMP and multi-core systems

To support very large port densities under GNU Bayonne 2, it is generally recommended to use SMP systems. GNU Bayonne 2 is already fully threaded and optimized for SMP use, although it can be tuned further to optmize SMP performance. In particular, some drivers offer the option of setting the number of parellel dispatch threads, with the default being just "1". These values should be increased in the appropriate config files for true SMP systems.

On many SMP "aware" operating systems, memory allocation is still often singular, and this can lead to unessisary lock delays. On AIX, this can be tuned by selecting multiple heaps, and this is the preferred method to use for GNU Bayonne 2 on that platform. Using a special SMP optimized memory allocation library, such as "libhoard" is also recommended.

4 Hardware support

When a GNU Bayonne 2 server is started, it will install support for a particular family of computer telephony hardware or IP telephony stack. This family is installed at runtime through a DSO plugin. When a driven for given family of hardware is selected, this generally means that one or more computer telephony boards of that product family may be used.

Some GNU Bayonne 2 framework support multiple driver plugin and the use of multiple family of computer telephony cards in one running instance. This means you can, for example, install both SIP and Voicetronix hardware in the same machine, and have both running together in the same instance of the GNU Bayonne 2 framework. You may also be able to have different vendor cards on the same server if you start multiple instances of GNU Bayonne, with each in-

stance started with a seperate driver plugin. The default Bayonne application server only supports loading a single driver type into a running instance. Some alternative servers built on the framework will offer loading of multiple drivers to meet specific needs.

Many of the GNU bayonne 2 drivers can not only support multiple boards of a given hardware family, but may also support mixing different types of boards from a given family of hardware together in a single running image. For example, one can use any combination of one or more supported analog and digital telephony cards from Intel/Dialogic with the Bayonne Intel/dialogic driver in GNU Bayonne 2.

Some GNU Bayonne 2 driver plugins may only support one type of card installed in a given running instance of the server. The current GNU Bayonne 2 Voicetronix driver, for example, supports either the use of Voicetronix OpenSwitch12 or OpenLine4 cards, but not the use of both of these cards together in the same server. The exact limitations and range of hardware support is detailed in the subsection on each hardware family.

4.1 Using OpenH323 Driver

You will need to start by installing a modern distribution of OpenH323. This may be found at www.openh323.org. A version of openh323 ¿ 12.0 and pwlib ¿= 0.5 should work. Earlier versions will not. You can compile and install each of these packages with make install to the default directory, and Bayonne will be able to find and use your openh323 configuration. It may be able to use other kinds of installation configurations as well.

Because there is no reliable means to auto-detect which version of openh323 has been is installed, and the entire Bayonne build process will halt if an older and unsupported version is found, openh323 support and auto-detection is now disabled by default in the Bayonne configure script. You will need to use “configure −−enable-openh323” to explicitly activate it. Since various versions of openh323, source builds, and pre-packaged distros of openh323 all use different methods for locationg and configuring installation paths for openh323 libraries and includes, the auto-detect code found in configure may not be sufficient, two extra configure options also exist; “−−with-pwlibdir=path” and “−−with-openh323dir=path”.

If you do all these things, and all the pre-requisets are met, GNU Bayonne should compile. During the "configure" process, and using the "−−enable-openh323" option, you will want to look for the line: "checking for openh323 libraries... found". If this appears in configure, then probably everything is fine.

While current openh323 support is built around the standard openh323 distribution, Bayonne 2 will soon migrate to use Opal. From Opal we will also adapt iax2 and perhaps an alternate sip driver.

4.2 Using SIP Driver

You need to use the CVS version of osip2 and exosip as found on Savannah. First, you need to perform a cvs checkout of the sources, as follows:

```
cvs -d :pserver:anonymous@subversions.gnu.org:/cvsroot/osip login
```

```
cvs -z3 -d :pserver:anoncvs@subversions.gnu.org:/cvsroot/osip co osip
```

```
cvs -d :pserver:anonymous@subversions.gnu.org:/cvsroot/exosip login
```

```
cvs -z3 -d :pserver:anoncvs@subversions.gnu.org:/cvsroot/exosip co exosip
```

Next, go into the osip directory, find, and run the "./autogen.sh" command. This will build the configure script you need for osip. Now you can run "./configure" followed by "make install".

Once osip2 is installed, go into the exosip directory, and find autogen.sh there. When you run configure do so as "./configure –disable-josua". Now you should be able to do a "make install". Finally, go (back) to Bayonne, and perform the configure and build steps there.

Current support is for the exosip “1” library. In the future we will support exosip “2”.

4.3 Using Intel/Dialogic Hardware

GNU Bayonne 2 has been tested successfully with both Analog and Digital span voice processing cards from Intel. One or more Intel/Dialogic cards may be placed in a given server and used directly. Intel/Dialogic Cards of different types may also be mixed together.

When multiple cards are used in a GNU Bayonne 2 server, it is nessisary to interconnect the sc-bus cables accross all of your cards. GNU Bayonne also requires sufficient voice processing resources to be available for all the ports you wish to accept calls on concurrently. Voice processing can be shared between different cards over the sc-bus directly.

GNU Bayonne 2 supports a single driver for Intel/Dialogic hardware based on Intel/Dialogic globalcall services. This driver supports both analog cards and single or multi-span T1/E1 cards used on PRI circuits and provides support for DM3 voice cards as well as older lines of Intel/Dialogic hardware. This driver can work with older releases of Intel/Dialogic runtime systems, but it is recommended that the latest, 5,1 (sp1) be applied and used under GNU/Linux with GNU Bayonne 2.

The driver supports automatic detection of your Intel/Dialogic hardware and require minimal configuration options in GNU Bayonne 2 using the same [dialogic] section in /etc/bayonne/drivers.conf. You are required to install a dialogic driver/sdk kit.

4.3.1 Before You Begin; know your hardware

The next section is meant to simplify the installation process for System Release 5.1 with Service Pack 1 (SP1) for GNU/Linux, with a separate section on DM3-based hardware. DM3-based hardware is newly supported under GNU/Linux and represents a whole new series of Dialogic brand telephony boards. DM3-based

cards can be used with the GNU Bayonne 2 Intel/Dialogic driver.

Before beginning the installation, it's crucially important to know what hardware you have and to verify that it's supported by this release. The list of supported hardware is in the Release Guide. Read it; know it. Furthermore, GNU Bayonne 2 requires that, whatever combination of hardware you use, that sufficient voice resources are present. This means that if you use non-DSP span termination cards, you may need to also have sc-bus dsp resource cards in your system for a working configuration.

Basically, there are 2 different hardware families of Intel/Dialogic hardware - Springware and DM3. The Springware family has been around for a long time and includes everything from 2-port analog cards up through dual-span cards with 48 ports. The DM3 family begins at 48 ports per board and goes up from there.

The easiest way to tell the difference is to look at the model name on the card bracket sticker. All DM3 hardware will have a model name that starts with: DM/ while Springware model names will start with plain D/ (or MSI/ for the MSI cards). For instance, a Springware card might have a model name of D/480JCT-2T1 while a DM3 card might have a model name like DM/V480A-2T1. You get the drift.

There is an excellant faq which describes how to install the Intel/Dialogic runtime for GNU/Linux already, and I will not repeat this here. The key point, however, is that the current release of their driver requires the LiS streams package, and Intel/Dialogic recommends using LiS 2.13.16. The driver and sdk is also only certified for use with either RedHat 7.2 GNU/Linux with the 2.4.9-13 (or greater) kernel update or RedHat 7.3 GNU/Linux with the 2.4.18-5 (or greater) kernel update RPM.

My own experiance with the Intel Dialogic sdk's have suggested that it is far better to actually use LiS 2.14.6 or later, which are more stable than the depreciated and unsupported 2.13 beta series for LiS. Also, it has been said that any GNU/Linux distribution that is somewhat similar to these RedHat releases may also work fine with a little extra work, including Debian (woody?) and several Mandrake releases. However, you may try those at your own risk of a failed sdk install.

If you are installing DM3 hardware, please be aware that the configuration and installation of the sdk for this hardware requires some extra steps. In any case, if the installation is successful, you will be able to start your dialogic runtime environment by running /etc/rc.d/init.d/dialogic start. If you try this, you have the hardware installed, and it initially fails, sometimes it helps to simply reboot the machine after installing their sdk.

Assuming the startup completes normally, your next step is to test your Intel/Dialogic hardware and sdk configuration. This is important to do if you are inexperianced with Intel/Dialogic hardware before you try configuring and using GNU Bayonne 2. Very often we find many questions come up on the mailing list that are not Bayonne issues at all and which could have been resolved simply by doing these tests first before trying to get Bayonne to work with your configuration.

4.3.2 Testing your system

The best/only way to test DM3 hardware is via the few demo programs included that are DM3-capable. That is, essentially, /usr/dialogic/demos/gc_demos/gc_basic_call_model. To run that, run:

```
# cd /usr/dialogic/demos/gc_demos

#  vi  gc_basic_call_model.cfg       #  Edit  this file to reflect the
boards/channels/protocols  to test. If you don't get this setup right,
it won't work!!

# ./gc_basic_call_model
```

If you get an error from gc_Start(), reboot the system. There are some library dependencies that get resolved at boot time.

The software furnished for testing Springware hardware is much broader and is really dependent on your hardware. Specifically, analog vs digital; ISDN vs robbed-

bit, etc. However, you *can* (and should, actually) use the afore-mentioned gc_basic_call_model program which has the advantages of working with all hardware and all protocols.

However there is also available:

pansr - which works with analog and robbed-bit T1 with immediate start protocol. It's located in /usr/dialogic/demos/dx_demos. The source code is there and you should take a look at the command-line options as they need to be set right or, surprise!, it won't work.

isdiag - a great PRI ISDN testing program in /usr/dialogic/bin

sampletest - a stripped-down isdiag, but with source code in /usr/dialogic/demos/cc_demos

If you cannot get Bayonne to run with your springware hardware, please, first test that your system is really working, that your carrier spans are really configured correctly, etc, and try these programs before requesting help on the Bayonne mailing list.

4.3.3 Driver Options

In order to fully utilize all of GC, you'll need to download the latest protocol package from http://support.dialogic.com/releases/protocols/GCProtocols30/index.htm so if you are using the GlobalCall driver, you may as well get that now.

We assume that at the time of GNU Bayonne 2 release, the Bayonne globalcall driver will support full auto-detection of Intel/Dialogic cards. However, you still need to specify the protocols that are being used by digital span cards as part of the [dialogic] section of /etc/bayonne/drivers.conf. To understand how this works, we should review how Intel/Dialogic names line protocols.

4.4 Using Voicetronix Hardware

The Voictronix driver supports two types of Analog DSP computer telephony cards made by Voicetronix (http://www.voicetronix.com.au). These two cards are the OpenLine4 and OpenSwitch12. Both these cards use the same the driver, which may be downloaded directly from the Voicetronix site.

Download the latest driver. This will be named something like vpb-driver-2.2.24.tar.gz. Untar it to a directory, such as say /usr/src/vpb. Enter the directory you have unpacked the driver in and perform a "make OSTYPE=linux". This will build the Voicetronix card driver for your machine.

The first problem you will find is that the Voicetronix driver requires the Linux kernel sources to be installed on your machine. You must install these, and then perform a "make menuconfig" in your Linux source directory to assure there is at least a default build configuration.

Assuming you do this, and then return to compile the Voicetronix driver again, you will find that on at least some GNU/Linux distributions, it will still not compile. This is because most current 2.4 based systems assume that the kernel sources are referenced through /usr/src/linux-2.4, and the Voictronix driver simply assumes /usr/src/linux. What I typically do is modify Voicetronix Makefile so that the kernel include path matches up correctly for current distributions. You can do this, or just create an extra sym-link under /usr/src/linux.

Assuming you are able to compile the source, you should then perform a "make OSTYPE=linux install". This will install the kernel driver modules, an include file, and a link library. Unfortunately, the vpb package chooses to install the vpb link library (libvpb.a) under /usr/local/lib, and the include file (vpbapi.h) under /usr/include. I suggest moving both either to /usr or to /usr/local, as Bayonne may fail to find or properly build Voicetronix support depending on what prefix it is configured for if these are mixed up this way.

Next you will need to install the Voicetronix kernel modules. This can be done with the insmod command. If you have OpenLine4 cards, then you need only perform an "insmod vpb". If you have the OpenSwitch12 card, you will also need to perform an "insmod vpbhp". These drivers will be shown in the kernel,

and you will need to find out what major device numbers they are using from /proc/devices.

Once you determine the major device numbers, you must create device nodes. If the vpb device is "254", for example, you will create 4 device nodes; "mknod /dev/vpb0-3 c 254 0-3". You should set the file permissions for these devices to enable the userid that you will run the Bayonne server under to access them. Ordinary user accounts can access and use Voicetronix hardware if you choose appropriate permissions to enable this. What I typically do is create a seperate "bayonne" entry in /etc/groups and selectivily add specific user accounts to that group. You can then create your device nodes under the bayonne group and have that group have rw priviledes to the device. If you have an OpenSwitch12 card, you will also need to create a device node for /dev/vpbhp0 using that devices major number.

If you are using the VoiceTronix OpenSwitch series of cards, you will need to specify VPB_MODEL=V12PCI in your environment. The current driver will autodetect the number of ports and cards you have in your system, but it will not allow you to mix OpenSwitch and OpenLine cards in the same machine.

4.4.1 About the OpenSwitch12 Card

The OpenSwitch12 card is a complete PBX on a single board. This allows you to directly plug analog telephone stations into the card and use them as internal extensions, as well as plug your incoming analog phone lines to it. The OpenSwitch12 is a full length PCI card. Some tower cases now being made choose to extend drive bays down the entire length of the case. These cases will not accept a full length PCI card and you will not be able to fit an OpenSwitch12 card in them. As with all PBX cards, you need to follow the vendors instructions to avoid damaging your phone service and your card. Specifically, we would not want to plug a trunk line into a station port.

Another issue you will find with the OpenSwitch12 card is that the card must operate on a seperate and unshared IRQ vector. If you install your card and insert the vpbhb driver module, you will see in /proc/interrupts what irq your card is using. If it is using an irq that is not used by any other device, then, after inserting the driver, you will see the number of interrupts incrementing in

/proc/interrupts each time you check it. If this is not happening, but the device is listed, along with any other device that also is using the same irq, then your card will simply not function.

The simplest way I found to get around this problem was to disable the serial ports, and then use the plug and play bios to force the PCI slots to use the irq range of the serial devices (irq 3 and 4). Most motherboard based resources tend to start from irq 10, although other pci cards will also do this to share irq's. Unfortunately, the OpenSwitch12 card does not share.

The card can be configured to operate either as a 12 port analog dsp telephony card, as a card with 4 telephone extensions and 8 phone lines, as a card with 8 telephone extensions and 4 phone lines, or as a card which supports plugging in analog telephones on all 12 ports. This use is configured by a set of onboard jumpers which are labeled on the card. Some of these configurations allow for failover, where, if the card is inactive or the server dies, the first 4 trunk ports are connected to the first 4 analog telephone stations. Here is the common jumper settings:

Standard Setup:

JS1,2,5,6 and H1-H16 are jumpered so that Bank A, 0..7 are set as station:

Fail-Over is enabled: H27-34 are jumpered

Next you will need to make cables for your card. Each of the 3 RJ-45 ports on the back of the Voicetronix card supports 4 telephone lines.Take a look at the connectors on the back of the card. Since you probably have your card in your PC, here is some info on the connectors on the back (labeled on the circuit board J1, J2, and J3):

J1 is furthest from the PCI slot
J2 is in the middle
J3 is closest to the PCI slot

The Readme.OpenSwitch12 document describes the OS12 pinout in the "Port Wiring" section. There it is explained that physical connectors J1 and J2 are Bank A. Connector J3 is bank B. Bank A is always ports 0 through 7, and Bank B is always ports 8 through 11.

J1 = ports 0..3 of Bank A
J2 = ports 4..7 of Bank A
J3 = ports 8..11 of Bank B

Harken back to the jumpering section, and we can safely say that we have jumpered the four ports closest to the PCI slot to be trunk ports (the phone company lines get connected here.)

If the tab of an RJ45 is pointing down (you are looking at the conductor side), and the head is flush against your palm, pin 1 is biologically proximal to your wrist, and pin 8 is distal. That is, pin 8 is away from you.

4.5 CAPI4Linux and GNU Bayonne

4.6 Using Soundcards

While soundcards are not actually "computer telephony" hardware, they can also be used with GNU Bayonne 2. This can be very useful since one can develop, debug, test, or even to some extent demonstrate common GNU Bayonne 2 applications without having computer telephony hardware physically installed on the machine you are using. This is particulaly useful in larger development environments where not every hacker has a production server with what might be expensive computer telephony hardware sitting on their desk or available for their exclusive use. Support for using soundcards is built by default, and Bayonne can be started with the soundcard driver by passing the −−test soundcard option to the bayonne command.

When GNU Bayonne 2 is used with a soundcard, it will always execute in the foreground rather than as a background daemon. The keyboard will act as a telephone keypad and can be used to simulate dtmf key interactions. The space-

bar can be used to start the application you are demoing under the soundcard driver, and the 'H' key can be used to simulate a caller hanging up. You can use CONTROL-C or 'D' to shutdown Bayonne.

GNU Bayonne 2 uses "/dev/dsp" to operate the soundcard, and may not work correctly with some broken soundcards that only support fixed (44.1khz) sample rates. Otherwise, it should work on any GNU/Linux machine that one is able to get sound working on. Support exists for soundcard audio on Microsoft Windows as well. The OS/X soundcard support is currently broken but is an issue isolated to the ccaudio2 library.

4.7 Hardware Quick Reference

The following table summerizes the computer telephony hardware supported by GNU Bayonne, and the name of drivers used. These names can be used during server startup when passed to the Bayonne server as "−−test name" "−−load name", or that may be entered in the /etc/sysconfig/bayonne file.

Name	**Driver**	**Summery**
capi20	Capi Driver	Driver for CAPI 2.0 compliant BRI & PRI cards
globalcall	Intel/Dialogic analog	span cards under GlobalCall driver
soundcard	OSS Sound Driver	Soundcard driver for testing
voicetronix	Voicetronix Driver	OL4 and OS12 telephony cards

5 Copyright