GNU Bayonne Installation Guide

David Sugar
Open Source Telecom.
sugar@gnu.org, http://www.gnu.org/software/bayonne

2003-01-03

Contents

1 Introduction

This guide was created to assist those trying to build and install GNU Bayonne for the first time. GNU Bayonne is a large and complex package, and involves both hardware and software, as well as other additional software packages, that may be needed to configure and successfully deploy a running server.

GNU Bayonne 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++, 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 as it is distributed through the GNU project and used as a standard part of your GNU/Linux systems.

This document is intended to cover installation of the 1.2 release of GNU Bayonne. Much of what may be found here should be relevant even to much older

releases, and certainly will remain relevant to future releases of GNU Bayonne until this documentation is updated.

2 Building GNU Bayonne

While GNU Bayonne 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 uses a "configure" script that is generated from autoconf and uses automake to generate Makefiles. The following subsections will help you in building GNU Bayonne for the first time.

2.1 Before you begin

GNU Bayonne differs from most software packages and services one may deploy on GNU/Linux systems in one key respect; to be useful, it depends on 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.

Most commonly, we assume GNU Bayonne will be used as a voice application server that sits behind an existing telephone switch such as an enterprise PBX system. However, GNU Bayonne can also be connected and used directly with the public telephone network, and, or even, with the right hardware, as a complete programmable telephone system entirely on it's own.

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 for. The second choice is then 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. This installation guide is not intended to explain how GNU may be used to create applications or what features it has, but only to cover issues relevant to software installation. If these questions are unknown to you, then it is strongly suggested that you carefully review the other documentation

that is part of GNU Bayonne 1.2 and then make some decision as to how you wish to use and deploy GNU Bayonne before continuing any further with this manual!

If you already understand how you intend to use GNU Bayonne and already have the 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 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 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 have verified that it is functionally working, as outlined in the hardware sections, you should proceed to the following section.

2.2 Building from Source

Before you can build and use GNU Bayonne, you must build and install GNU Common C++, GNU ccAudio, and GNU ccScript. These packages are all part of the GNU system and may be downloaded from `ftp://ftp.gnu.org/` or from any GNU mirror site. 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 commoncpp-{VERSION}.tar.gz
```

Substitute VERSION with the version of your package.

Before you install GNU Common C++, you may wish to consider if you will make use of XML and SQL support in GNU Bayonne. If you intend to use these features, then you need to have the Gnome XML library (libxml2). This library is already installed as part of a normal Gnome desktop system, and very often used by many other applications, so it is likely you will already have it on your system. If you are on an RPM based system, and intend to use XML support, libxml2 is often split between a main and "devel" package. You will need to make sure your "libxml2-devel" package is installed, or that a "xml2-config" command exists in your bin path for GNU Common C++ and Bayonne to detect and include XML support.

To install GNU Common C++, enter the commoncpp 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 from `ftp://ftp.gnu.org/gnu/cc` and the GNU ccAudio package from `ftp://ftp.gnu.org/gnu/ccaudio`. These packages depend on GNU Common C++ 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++ (if any). Then simply perform a "make install".

At this point you should also have installed any device drivers 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, then many GNU/Linux distributions 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

working correctly before you download and install Bayonne.

You may now download GNU Bayonne from `ftp://ftp.gnu.org/gnu/bayonne`. Once you download the GNU Bayonne 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. 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 is described in the next section.

If you do not wish to tailor GNU Bayonne, 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. 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 GNU Bayonne's configure script automatically detects which computer telephony card libraries are present and selects which computer telephony drivers GNU Bayonne 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 to build drivers for your card if you change hardware or forget to install your hardware first.

2.3 Compile-time configure options

There are a number of compile time options that can be selected by the "configure" script that is part of GNU Bayonne. 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 op-

tions enable specific features or support depending on additional optional packages being installed.

A number of compile-time options relate to disabling checks for additional telephony drivers. This may be useful if you are preparing a machine for use with a specific or known computer telephony hardware, and you do not wish to have GNU Bayonne build or even test for drivers and cards you will not be using. Each driver GNU Bayonne supports can be fully disabled with a simple "–without-xxx" option passed to "./configure". For example, to disable Aculab support, you can use "–without-aculab".

When using "–without-xxx" options to disable computer telephony card drivers, keep in mind that the name Bayonne uses for drivers is not always the same as what you might assume. For example, cards that use the Linux Kernel /dev/phone interface are known in Bayonne collectivily as "phonedev". Hence, to disable Quicknet and any other /dev/phone based telephony support, you would use "–without-phonedev". There is no seperate "–without-quicknet".

Disabling unused drivers may save on compile-time and some disk space for your system, but offers no other immediate benefit. Since GNU Bayonne selects which driver it will use at runtime and dynamically loads it, unused drivers are not present in memory when the server is running. However, there are a number of other features that may be disabled if you do not plan to use them which do consume considerable system resources.

Perhaps the single most useful compile time option in terms of system memory use is the "–without-xml" configure option. If you plan to create and deploy applications that are written in Bayonne scripting alone, or that use scripting and external scripting languages like perl or python for database connectivity, and plan to make no use of GNU Bayonne's ability to parse and transcode XML documents or access local databases, this option will strip out a large range of functionality and create a very optimized server image.

One immediate benefit from the "–without-xml" option is that the GNU Bayonne server will be linked against libccgnu2 alone, rather than including ccgnuext2, libxml2, and other miscellanous libraries. The one downside to using this option is that there are a number of additional server features and plugins which are associated with or that depend on server XML support. These include the direc-

tory scan and mailbox plugins used for building voice mail systems in ccScript alone, and SQL support for Bayonne's ccScript, which could eliminate the need for using external perl applications to bind database access. However, if you do not plan to make use of these extended features, then certainly it is useful to disable XML support.

Using Bayonne's tgi service and perl scripts to connect telephony applications with databases had one problem; tgi is transactional, and so each session would create and tear down a database connection. This reduced database performance. The need to externally script databases connectivity and maintain it seperate from call flow scripts meant two or more seperate applications written in different languages had to be used to create telephony applications that interact with databases.

With the 1.2 release of GNU Bayonne, we have added native scripting support for voice messaging and sql database connectivity. This support allows one to script these services entirely in Bayonne, without ever needing to use an external scripting language. As a result, one can now disable support for tgi and external scripting languages by using the new "–without-tgi" compile time option. This will mean no gateway processes will be created and no code to support TGI will be present in your server. However, because sql and msgbox support are tied to XML support, this option should be considered as exclusive with the "–without-xml" option as one would normally not use both.

Another curious feature is the ability to host 'user' applications, These operate in a manner somewhat anagolous to how a web server hosts special xxx user paths and allows individual login accounts on a machine running a Bayonne server to host and privately maintain complete Bayonne applications. The sandboxing of individual user logins is not complete, and this feature is only really useful in rare hosting situations where an ASP is providing third party telephony services hosting. Most deployed uses of Bayonne will not make use of this capability, and it may be safely disabled using "–without-users" from "configure".

Finally, GNU Bayonne has a network protocol module that allows multiple servers to directly communicate with each other. This protocol support allows each server to form a global network call state for an entire site of Bayonne servers. This protocol is also used to support the site monitoring utility (sitemon) and other specialized network services. If you have only one stand-alone server you are deploying, and you do not wish to take advantage of any of these network services,

you can use the “–without-nodes” configure option.

2.4 Enabling special features

Not all compile-time options are used to disable features in GNU Bayonne. There are a couple of special “configure” options which can be used to add new and specific capabilities to your GNU Bayonne server.

Asychronous I/O is still an experimental kernel feature in the Linux kernel. However, there are a number of benefits one can derive from using ‘aio’. These include the fact that ‘aio’ services can be timed, and hence if one is reading voice data from the disk subsystem, the process or thread doing this will not block indefinately on bad data or when failing to seek the next disk block of voice data. Support for asynchronous i/o can be enabled using “–enable-aio”. This also requires libaio to be present on your system, and for the computer telephony driver to also optionally support aio. Most drivers at this time do not support or make use of aio when enabled, although this may change in the future.

Perhaps more useful is the “–with-flite” option. This option can enable and install support for Carnegie Mellon’s Flite (Festival Light) text to speech system into GNU Bayonne. To use this option, you must also specify the directory where your flite package is unpacked. This is typically done by using “–with-flite=dirpath”, such as “–with-flite=/home/myhome/src/flite-xxx”, where the path is where flite is installed, for example.

When “–with-flite” is used, the say command adds support for both caching text to speech translations, and for using flite internally linked with the bayonne server. When flite is intalled, this also disables the altplay keyword which otherwise supports non-tts alternate speech. This results in a high performance text to speech system that you can use with synthesised voices. If you have the flite derived “theta” tts system, you can instead use “–with-theta=/opt/theta” to enable that instead of flite (assuming theta is installed under /opt/theta).

2.5 Using pre-built packages

Many GNU/Linux distributions include pre-compiled and pre-packaged versions of GNU Bayonne. GNU Bayonne 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 is possible to do an "apt-get install bayonne".

However, when installing pre-compiled copies of GNU Bayonne, 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 did not have the link libraries for your hardware installed on it.

A simple solution is to rebuild GNU Bayonne 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 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 production 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 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, however.

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 -z3 -d :pserver:anoncvs@subversions.gnu.org:/cvsroot/commoncpp co -rRELEASE1 co
```

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. The -rRELEASE1 option is used to retrieve the "RELEASE1" branch of GNU Common C++, which is currently the stable branch.

Once GNU Common C++ is compiled and installed, you can use the following script to retrieve the remaining packages you will need from cvs:

```
for i in ccaudio ccscript bayonne;
  do cvs -z3 -d :pserver:anoncvs@subversions.gnu.org:/cvsroot/\$i co \$i;
done
```

You should then enter the ccaudio and ccscript directories in turn, and perform a "./reconfig" in each one, followed by "./configure" and "make install". This will build and install ccscript and ccaudio from cvs. Finally, once these are installed, you can do the same for GNU Bayonne. After running "./reconfig" in the bayonne directory, you may wish to consider reviewing the section on compile time options to determine if there are any you wish to change through the "configure" script before configuring and installing Bayonne.

With a cvs checkout of GNU Bayonne, 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 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@lists.sourceforge.net). 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 hardware

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 is working and operates correctly with Bayonne even before Bayonne itself has been installed.

GNU Bayonne test mode is activated by passing the "–test" option when running the server. If you are in the compile directory, you can quickly test Bayonne with it's internal server regression test by entering the following command: "server/bayonne –test –soundcard". This will start GNU Bayonne with the soundcard driver, as explained in the hardware section later on, and then will perform each of the internal server regression tests, and then exit when they are complete, while reporting the results to you on the console screen.

You can also run and test applications delivered with GNU Bayonne without isntalling the server. For example, there is a simple application called "playrec" which allows you to play and record prompts. You can see that your server is running, again using the soundcard driver, by entering "server/bayonne –test –soundcard playrec". This time, rather than performing the regression tests and exiting, the server will startup and wait for an 'incoming' call. With the soundcard driver, you can simulate an incoming call by pressing the space bar.

Once you are sure the server is operating correctly, you can begin to test it with your hardware even before you install it. Testing it this way is useful because you will receive all kinds of debugging information in the console window. This can also be useful for testing new releases before they are installed. To perform the basic regression test using your telephony hardware, you can pass the name

of the driver to load for your current system. For example, if you have Voictronix hardware, you would use "server/bayonne –voicetronix –test" and Bayonne will run it's normal regression test after loading the Voicetronix driver. This will tell you right away if Bayonne is able to function correctly with your hardware.

You can also use test mode to run and test applications just like with the soundcard driver. For example, if you do "server/bayonne –dialogic playrec", and hook your Dialogic card to a live telephone circuit, it will answer and run the playrec script over the phone line. Once you are done with any testing you wish to do, you may then perform a "make install" to install the server and it's various support files.

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 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 /etc/bayonne.conf. That information is beyond the scope of this manual.

4 Hardware support

When GNU Bayonne is started, it will install hardware support for a particular family of computer telephony hardware. 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.

GNU Bayonne only supports a single hardware plugin and the use of a single family of computer telephony cards at any one running instance. This means you cannot, for example, install both Voicetronix and Aculab hardware in the same machine, and have both running together in the same instance of GNU Bayonne. You may, however, be able to have different vendor cards on the same server if you start multiple instances of GNU Bayonne, with each instance started with a seperate driver plugin.

Many of the GNU bayonne 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.

Some GNU Bayonne driver plugins may only support one type of card installed in a given running instance of the server. The current GNU Bayonne 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 Aculab Hardware

GNU Bayonne supports Aculab span cards that include voice resources under GNU/Linux. I am not immediately familiar with the aculab product lines or which cards are specifically supported, however.

Aculab chooses to make the source for their kernel drivers and sdk libraries fully available to their customers. I do not believe they are available on a true "Open Source" license, but one is free to at least compile and use the sources provided to build a system. In fact, since no binaries are included, it is nessisary to do this before you can install and use Bayonne with Aculab hardware.

Since the aculab sdk package offers no default instructions on where to install their drivers by default, you need to specify the location where your aculab sdk is being installed. This can be done with the –with-aculab= directive in your configure script. The default location I assume is /usr/src/aculab/dtk111, and if you build your sdk there, then Bayonne will automatically detect it without the need of special configure options.

Once you unpack your aculab dtk, you will need to go through the call, speech, switch, and tools directory, and run the "make" command in each. I believe there are some additional steps required to compile and activate their kernel module, although I have never done this, as I have never had the oppertunity to use their hardware. What I do recall of their kernel module is that they have a header file that has to be hand modified based on the card model and port capacity.

Assuming the kernel module compiles and activates successfully, and the full dtk is built, it should be possible to install Bayonne and then compile it for aculab driver support.

4.2 Using Intel/Dialogic Hardware

GNU Bayonne 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 be mixed together freely.

When multiple cards are used in a GNU Bayonne 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 supports two seperate drivers for Intel/Dialogic hardware. The first driver is based on the older Dialogic "runtime" services and supports both analog cards and single or dual span T1/E1 cards used on PRI circuits. 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.

GNU Bayonne also supports the Intel/Dialogic GlobalCall api with an alternate "globalcall" driver. This driver primarly supports digital span cards, and provides support for DM3 voice cards as well as older lines of Intel/Dialogic hardware. The Globalcall driver also supports raw t1/e1 channels as well as PRI. Eventually, the older runtime driver will be depreciated in favor of globalcall going forward.

Both the globalcall and runtime drivers support automatic detection of your Intel/Dialogic hardware and require minimal configuration options in GNU Bayonne. Both also use the same [dialogic] section in /etc/bayonne.conf. Both require you install a dialogic driver/sdk kit.

4.2.1 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 only be used with the GNU Bayonne GlobalCall 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 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.

Our 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. 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.

4.2.2 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.

4.2.3 Runtime Driver Options

4.2.4 GlobalCall 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 1.2 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.conf. To understand how this works, we should review how Intel/Dialogic names line protocols.

To be added by Mark from here!

4.3 Using Quicknet Hardware

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.

Finally, you need to edit the [vpb] section of your /etc/bayonne file for the card that you have installed. If you have one or more OpenLine4 cards, you would set the model= entry to "V4PCI", which happens to be the default. There is also an entry in [vpb] for the total number of cards you have installed. Each OpenLine4 card supports 4 analog telephony ports, and you can have up to 4 cards in a given system, for a total capacity of 16 ports.

If you are using one or two OpenSwitch12 cards, then you would set the model= entry of the [vpb] section to "V12PCI". Since you can only specify one card model at a time, you can only use either OpenLine4 or OpenSwitch12 cards in a single server; you cannot mix different cards together.

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.

4.4.2 Other Voicetronix configuration settings

There are a large number of miscellaneous configuration options available under the **[vpb] settings in /etc/bayonne.conf. Some are related to hardware configuration, tone detection, and to various timing needs for using the card. These are as follows:**

cards= is used to specify the total number of Voicetronix cards that you have installed in your system.

first= is used with the older ISA based Voicetronix cards to indicate the starting address, in hex, of your first voicetronix card. The old ISA card would normally start at 0x300 and use 0x10 bytes per card. Bayonne assumed any additional cards would immediately follow the first in order, so if you had multiple cards, they would occupy sequential i/o port addresses starting from where the first card was jumpered.

model= specifies the card model you have installed. This currently supports V4ISA, V4PCI, and V12PCI.

firmware= is used to specify the dsp firmware load file. This is only nessisary for the OpenLine4 and older isa cards. You would specify the full path of the firmware file you are using, as found in /etc/vpb.

stations= is used to specify the bank arrangement of OpenSwitch cards, as defined previously.

hooktime= the number of milliseconds required for the hook state of the OpenLine card or OpenSwitch trunk port to settle after an onhook or offhook request.

pickup= & hooktime= the delay in milliseconds required for a line pickup operation to complete before call processing can continue for the port when a line is being picked up.

reorder=f1,f2 is used to specify the frequency components of the reorder tone your PBX will generate to indicate the caller is in the reorder state.

4.5 CAPI4Linux and GNU Bayonne

4.6 Using “oss” Soundcards

While “oss” soundcards are not actually “computer telephony” hardware, they can also be used with GNU Bayonne. This can be very useful since one can develop, debug, test, or even to some extent demonstrate common GNU Bayonne 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 –soundcard option to the bayonne command.

When GNU Bayonne 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 spacebar 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 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.

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 “–name” or may be entered in the /etc/sysconfig/bayonne file or under driver= in /etc/bayonne.conf.

Name	Driver	Summery
aculab	Aculab Driver	Aculab BRI and PRI voice resource cards
capi20	Capi Driver	Driver for CAPI 2.0 compliant BRI & PRI cards
dialogic	Intel/Dialogic	Runtime analog, digital, ISDN cards
globalcall	Intel/Dialogic	span cards under GlobalCall driver
phonedev	Kernel Driver	/dev/phone based telephony devices
soundcard	OSS Sound Driver	Soundcard driver for testing
voicetronix	Voicetronix Driver	OL4 and OS12 telephony cards

5 Using non GNU/Linux systems

While GNU Bayonne is itself built on a highly portable code base, we primarily speak of it's use with GNU/Linux and will do so for the current release of this manual. This is principally because GNU Bayonne requires computer telephony hardware to do anything useful, and while most computer telephony hardware vendors directly support GNU/Linux systems, virtually none support other excellent freely licensed systems, such as FreeBSD, or even other GNU platforms, such as GNU/Hurd.

Where at least some computer telephony vendors do make their software and driver kits available under fully free licenses, it is at least potentially possible for these drivers to become available under or ported to other operating systems, such as FreeBSD. If and when drivers supported by GNU Bayonne become widely available on such platforms, additional information on how to use GNU Bayonne on these platforms will become a standard part of this installation guide. To help bring GNU Bayonne to these other platforms requires your help in leading these often blind vendors to other freely licensed platforms, and I encourage anyone reading this who uses other freely licensed platforms to do so if they can.

GNU Bayonne can and in fact has been successfully used on some proprietary Unix platforms. In particular, there have been people who have compiled and successfully used GNU Bayonne under Sun Solaris.

Such use has no immediate interest to me, although, as freely licensed software we do not make any statement of what platforms one can or cannot be permitted to run GNU Bayonne under other than to say one is free to make use of the source in any manner desired so long as the conditions of the GNU GPL are fully met should any derived versions of GNU Bayonne be distributed for use on other platforms.

If one has successfully ported GNU Bayonne to other platforms, and wishes to contribute help or documentation on how to install the system, I would be happy to consider adding such contributions to this document. We are also happy to take in patches for the distribution to support building on other platforms as it would be a futile exercise to limit where people can run this software and wasteful to have distributions fork over such an issue.

6 Copyright