FreeTDM

Introduction

FreeTDM is a library implementing unified high level API for both signaling and I/O for multiple telephony boards (Digium and Sangoma are most popular). FreeSWITCH uses a module called "mod_freetdm". If you come from Asterisk world, think of FreeTDM as you think of chan_dahdi, with the major difference that FreeTDM is nicely structured as a library so it can be used in other applications.

FreeTDM uses a modular architecture to plug different signaling stacks below. For example, FreeTDM can use libpri for PRI support. However also can use other PRI stacks, like the telco-grade stack from Trillium that Sangoma offers (free of charge).

FreeTDM has been designed to work with multiple boards from multiple vendors. The most well known and used vendors are Digium and Sangoma. In order to expose a common API for different hardware I/O modules are needed. There are three I/O modules in FreeTDM as of this writing, one for every hardware layer supported.

I/O modules

- Sangoma (ftmod_wanpipe): This module uses native Sangoma API (libsangoma) which is installed when you install the Wanpipe drivers. More information on the Wanpipe drivers can be found at Sangoma's wiki: http://wiki.sangoma.com/, this module can only be used with Sangoma cards and is the recommended option for anyone with a Sangoma card.
- Zaptel/DAHDI I/O module (ftmod_zt): This module supports the DAHDI interface. It also supports the old Zaptel interface but its usage is discouraged and you are not likely to get support from developers if you use Zaptel. The DAHDI interface is used by several hardware vendors, including Digium, Redfone, Sangoma and Xorcom (although Sangoma can work in DAHDI mode, it's recommended to use the native Sangoma mode via ftmod_wanpipe).
- PIKA (ftmod_pika): This module is intended for PIKA boards.

I/O modules take care of writing and reading raw data and executing commands in the boards, but do not do anything else. All high level intelligent telephony signaling is handled by the FreeTDM signaling modules.

Signaling modules

FreeTDM supports PRI, BRI, SS7, MFC-R2 and Analog (and some others) signaling. One thing to keep in mind is, there is no single module per signaling. You may have different options within the same signaling. So for example, for PRI you can choose between 3 different stacks. Hopefully the following description will help you decide.

ISDN Modules

- Sangoma ISDN module (**ftmod_sangoma_isdn**): Offers telco-grade Trillium stack to **provide both PRI and BRI signalling support** (only supported with Sangoma cards). In order to install this you must have libsng_isdn already installed. For more information about installation and configuration for this module you can go here: http://wiki.sangoma.com/Freeswitch-FreeTDM-Sangoma-ISDN-Library
- LibPRI module (**ftmod_libpri**): Offers support for PRI lines using the open source libpri stack for any type of card supported by FreeTDM (i.e. Sangoma and Digium). In order to install this module you must have libpri already installed. **Now supports both PRI and BRI signalling** (since 3d5ccf05 on 06th Nov). You must be using a version of libpri that supports BRI to get that support (libpri-1.4.12_beta1 or newer). As features are added to libpri, this module must be updated to support them.
- Native ISDN module (ftmod_isdn): First stack ever supported by FreeTDM for PRI lines. This module is **still under development and is not considered stable**.

SS7 Modules

- Sangoma SS7 module (**ftmod_sangoma_ss7**): Offers telco-grade Trillium stack to provide SS7 signaling support (only supported with Sangoma cards). In order to install this module you need libsng_isdn library from Sangoma and you also need to get a commercial license key. **Contact Sangoma for more information**.

Analog

- Analog module (**ftmod_analog**): Offers generic FXO/FXS support for any type of card supported supported by FreeTDM (i.e. Digium, Sangoma and Xorcom).
- Analog EM module (**ftmod_analog_em**): Offers generic E&M signaling for any type of card supported by FreeTDM (i.e. Digium and Sangoma).

MFC-R2

- OpenR2 module (**ftmod_r2**): Offers MFC-R2 support in E1 lines for any type of card supported by FreeTDM (i.e. Digium and Sangoma). In order to compile this module you need to have the openr2 library installed (latest SVN trunk is required). http://www.libopenr2.org/

For more information about how to configure MFC-R2 support go here: FreeTDM_OpenR2

Custom protocols

- Boost (**ftmod_sangoma_boost**): **Deprecated module** offering support for the boost protocol to access PRI, BRI and SS7 signaling daemons for Sangoma cards. In order to compile this module you need to have SCTP libraries and development headers. **Again, this module is DEPRECATED, DO NOT USE IT**, the best way to use a Sangoma board is using ftmod_sangoma_isdn for BRI and PRI, ftmod_sangoma_ss7 for SS7 signaling and ftmod_analog and ftmod_analog_em for Analog protocols.

Tapping

- PRI tapping (ftmod_pritap): This module is used to tap (passive monitoring) PRI lines (you cannot place calls, just monitor calls from other T1/E1 links). It's been tested only on Sangoma cards, but it may work with other vendors. In order to compile this module you must have libpri installed and run the FreeTDM ./configure script with the option "--with-pritap"

Installation

FreeTDM is a library and as such can be installed alone, no need of FreeSWITCH. However, to do anything useful with it, you either need to write your own application, or use FreeSWITCH. As FreeTDM was born as a part of FreeSWITCH, the source code is still part of the FreeSWITCH main code repository. If you just want FreeTDM as an API there is no need to install FreeSWITCH.

Dependencies

Some FreeTDM modules depend on other libraries, and those libraries are detected during the ./configure step, therefore you must install the dependencies beforehand otherwise the modules you want won't be built.

- ftmod_wanpipe depends on the Wanpipe headers and libsangoma to be installed. You must install the Wanpipe drivers before configuring the FreeTDM build if you want that module.
- ftmod_libpri and ftmod_pritap depends on libpri (http://downloads.asterisk.org/pub/telephony/libpri/releases/).
    - if the mod/ftmod_libpri.* files are not installed, you may need to specifically compile them... libpri now depends on dahdi. edit Makefile and look for the line that starts with 'UTILITIES='. Remove everything on the line except 'UTILITIES='. The test scripts are not needed if you are not using dahdi.
        - cd libs/freetdm
        - ./configure --with-libpri --prefix=/opt/freeswitch
        - make
        - make install
- ftmod_sangoma_isdn depends on libsng_isdn (http://wiki.sangoma.com/FreeTDM-Sangoma-ISDN-Library and http://wiki.sangoma.com/wanpipe-freeswitch-ftdm-libsngisdn#libsng_isdn).
- ftmod_r2 depends on openr2 (http://www.libopenr2.org/)
- ftmod_sangoma_boost depends on SCTP library and headers.

If the dependencies are not met, simply those modules will not be built, if you do not need them then you don't need to install their dependencies.

The following script is a convenient way of doing all the stuff mentioned here in the Sangoma wiki:

```
#!/bin/sh
#
# Simple script to install FreeTDM/Wanpipe pre-reqs
# NOTE: Assumes you want to put stuff in /usr/src
#
cd /usr/src
echo Fetching wanpipe drivers...
wget ftp://ftp.sangoma.com/linux/current_wanpipe/wanpipe-current.tgz
machine=`uname -m`
echo Fetching libsangoma for ${machine}
wget ftp://ftp.sangoma.com/linux/libsng_isdn/libsng_isdn-current.${machine}.tgz
echo Unpacking wapipe drivers...
tar zxvf wanpipe-current.tgz
echo Unpacking libsangoma...
tar zxvf libsng_isdn-current.${machine}.tgz
echo Building wanpipe...
wanpipedir=`tar tzvf wanpipe-current.tgz | head -n 1 | cut -c49-`
cd ${wanpipedir}
make freetdm
make install
echo Building libsng...
cd ..
libsngdir=`tar tzvf libsng_isdn-current.${machine}.tgz | head -n 1 | cut -c49-`
cd ${libsngdir}
make install
cd ..
echo Doing wanrouter hwprobe...
wanrouter hwprobe
```

Low level drivers

In case of DAHDI, low level drivers have to be installed. Luckily there is a ready-to-be-built git tree to install the Digium DAHDI svn latest trunk + zaphfc driver + OSLEC echo canceller. However, DAHDI devices are owned by asterisk:asterisk by default. They must be changed to freeswitch:freeswitch before installing.

```
git clone --depth=0 git://dahdi-zaphfc.git.sourceforge.net/gitroot/dahdi-zaphfc/dahdi-zaphfc
cd dahdi-zaphfc
make
find . -name genudevrules -exec sed -i 's|asterisk|freeswitch|g' {} \;
make install
```

Debian provides a tidy way to install DAHDI via module assistant.

FreeTDM with FreeSWITCH

If you want to use FreeTDM along with FreeSWITCH, the installation is pretty easy. The FreeSWITCH ./bootstrap and ./configure scripts take care of configuring FreeTDM build too. The only additional step is open modules.conf and uncomment the line: ../../libs/freetdm/mod_freetdm, this will tell the build system you want mod_freetdm to be compiled. This module is necessary to glue FreeSWITCH with FreeTDM.

Having freeswitch working with freetdm and ftmod_sangoma_isdn you have to do this:

1. Download libsng_isdn version 1.2.0+
    a. As root
    b. Unpack
    c. make install
2. Download wanpipe version 3.5.17+
    a. Unpack
    b. make freetdm
    c. make install
3. Download Freeswitch (e.g. (git-d872408 2010-11-09 19-29-19 -0500))
    a. bootstrap.sh
    b. uncomment mod_freetdm in modules.conf
    c. configure
    d. make all; make install
4. Configure Wanpipe and FreeSWITCH for sangoma_isdn
    a. As root
    b. wancfg_fs

    c. Maybe you want to check file permissions of conf/freetdm.conf, conf/wanpipe.conf and conf/autoload-configs/freetdm.conf.xml if you run FS as non-root user
    d. Add d-channel in conf/freetdm.conf (e.g. d-channel => 1:16)
5. Start FS or load mod_freetdm

FreeTDM API

FILL ME!!

Configuration

Main Config files

- /etc/freeswitch/freetdm.conf - sets up the spans from the driver and configures the library (the freetdm library)
- /etc/freeswitch/autoload_configs/freetdm.conf.xml Sets context and several parameters
- /etc/freeswitch/tones.conf Needed for dialout to work and more.

Each config is explained further below in it's own section

FreeTDM Configuration

The FreeTDM library takes care of opening the span devices and configuring the I/O options. No high level signaling configuration at all is done by the library at startup. This configuration file is called **freetdm.conf and is a simple text file** (the few people using the C API of FreeTDM have the option of configuring via API and not via configuration file if they want to). The exact path to the file depends on the --prefix= option provided to the FreeTDM ./configure script. If you used FreeSWITCH build system then it'll be the same --prefix option you used for FreeSWITCH, which by default is /usr/local/freeswitch, and the full path to the config is **/usr/local/freeswitch/conf/freetdm.conf**

freetdm.conf

The freetdm.conf file has a simple format (not XML) and in that file you configure which spans and channels to use, the trunk type, groups, audio gains.

Some rules for the syntax of the file are:

- There is an optional [general] section that can be used for global configurations.
- FreeTDM spans are channel containers and each span definition starts with the word span, its type and name within brackets (i.e. [span wanpipe mySpanName]).
- Any channel must be placed within a single span configuration. The channels definition should be after the parameters the channel intents to use (order matters, just like Asterisk's chan_dahdi.conf).
- If you start any line with either ';' or '#', the line will be ignored.
- The special string "__END__" can be used at the beginning of a line to stop parsing the file (useful if you want to ignore the rest of the file at a given point).
- The ';' character can be used for comments at any place within a line.
- Both, '=' and '=>' can be used as the character to separate parameter name from parameter value.

How you configure this file will depend on which hardware layer you use (i.e. Wanpipe or DAHDI) and your particular scenario. The general syntax for the file is.

```
[general]
global_parameter => value

# This span is awesome
[span <span I/O type> <span name>]
; other comment here
parameter1_name => value ; mid-line comment
parameter2_name => value
parameterN_name => value
<sig>-channel => <channel-range-format>
parameterX_name => value
<sig>-channel => <channel-range-format>

# This span is not so awesome
[span <span I/O type> <span name>]
; other comment here
parameter1_name => value ; mid-line comment
parameter2_name => value
parameterN_name => value
<sig>-channel => <channel-range-format>
```

For the [general] section, the valid parameters are:

- cpu_monitor - 'yes' or 'no'. This determines whether FreeTDM launches a thread to monitor the CPU load and stop accepting calls when a given threshold is reached.
- cpu_monitoring_interval - How often in milliseconds to monitor the CPU usage. Defaults to 1000ms (1 second).
- cpu_set_alarm_threshold - Percentage of CPU at which an alarm will be triggered (defaults to 80).
- cpu_reset_alarm_threshold - Percentage of CPU at which an alarm will be cleared (defaults to 70).
- cpu_alarm_action - What to do when the CPU alarm is triggered. Valid values are 'reject' to stop accepting calls and and 'warn' to just print warnings. The values may be combined separated by comma.

This is an example of a general configuration section:

```
[general]
cpu_monitor => yes
cpu_monitoring_interval => 2000 ; monitor usage every 2 seconds
cpu_set_alarm_threshold => 90 ; whenever 90% of global CPU usage is reached, trigger the alarm.
cpu_reset_alarm_threshold => 80 ; when the CPU usage decreases at 80%, clear the alarm.
cpu_alarm_action => reject,warn ; Start rejecting calls when the CPU alarm is triggered and also print warnings.
```

For every span section, the valid I/O types are: wanpipe, zt and pika. So you can start your span sections either as:

```
[span wanpipe myWanpipeSpan]
```

or for Zaptel or DAHDI spans as:

```
[span zt myDAHDISpan]
```

finally, for PIKA spans that would be:

```
[span pika myPIKASpan]
```

The valid parameters in the configuration for each span are:

- trunk_type - Determines the type of trunk for this span. Valid values are: E1, T1, J1, BRI, BRI_PTMP, FXO, FXS, EM, this value is global for the whole span.
- analog-start-type - Either "kewl", "loop", "ground" or "wink".
- group - The group name for outbound dialing. You can put here any string, but it must start with a letter to avoid confusion with span numbers. Any channels below this group parameter will be added to the group for channel hunting during outbound dialing.
- txgain - The audio gain for transmission. Any float value. Very big float values will result on clipping of the audio though. Typical values range from -5.0 to 5.0.
- rxgain - The audio gain for reception. Any float value. Very big float values will result on clipping of the audio though. Typical values range from -5.0 to 5.0.
- number - This is only used for FXO to set an incoming DNIS to a fixed number.

In addtion to the previous parameters, you must also define which type of channels to add to the span. Any parameters before the channel parameter will affect the channel settings (again, like Asterisk chan_dahdi.conf).

- b-channel - specify one or more b-channels (ISDN and SS7).
- d-channel - specify one or more D channels for ISDN.
- fxo-channel - specify one or more FXO signaling channels (analog, RBS).
- fxs-channel - specify one or more FXS signaling channels (analog, RBS).
- em-channel - specify E & M signaling channels.

The valid value format for xx-channel parameters depends on the **I/O type (hardware layer)** (either wanpipe, zt or pika). Details are included in the next sections.

In addition to freetdm.conf, a few other files are optional configuration.

The following files are used by the I/O modules as global configuration:

- zt.conf
- wanpipe.conf
- pika.conf

The settings in those files are global in nature and specific to the I/O module in question. All parameters must be below a [defaults] section.

zt.conf accepted parameters:

- codec_ms - Number of milliseconds to configure the read/write rate for the channels. (defaults to 20)
- wink_ms - How many milliseconds to wait to detect a wink. (defaults to 150)
- flash_ms - Duration in millisecond of the flash (on-hook / off-hook).
- echo_cancel_level - Echo cancellation level in the DAHDI driver.

- rxgain - Tx gain in the DAHDI driver.
- txgain - Rx gain in the DAHDI driver.

wanpipe.conf accepted parameters:

- codec_ms - Number of milliseconds to configure the read/write rate for the channels. (defaults to 20)
- wink_ms - How many milliseconds to wait to detect a wink. (defaults to 150)
- flash_ms - Duration in millisecond of the flash (on-hook / off-hook).

Wanpipe mode

In this mode you do not need to have Zaptel nor DAHDI installed. You will need recent version of the Wanpipe drivers (using latest always recommended) and compile them in TDM Voice API mode, you can follow the ./Setup script instructions. You can also just "make freetdm && make install" in the Wanpipe drivers directory for a quick installation of the drivers in the proper mode.

For wanpipe spans the xx-channel (ie b-channel) parameter value syntax is:

```
xx-channel => <wanpipe-span-number>:<channel number> xx-channel => <wanpipe-span-number>:<low-channel-number>-
<higher-channel-number>
```

So you can add either a single channel or a range of channels within a span. The wanpipe devices are populated in the Linux file system by "wanrouter start" Wanpipe command at the /dev/ directory. For Windows you can look at them using the Windows device manager.

As an example, for a range of channels from 1 to 23 in span 4, devices ranging from /dev/wanpipe4_if1 to /dev/wanpipe4_if23 will be created in Linux file system and you can add them to freetdm.conf like this:

```
[span wanpipe trunk1]
trunk_type => T1
b-channel => 4:1-23
```

The key here is "wanpipe" is set as the I/O type in the span definition and the xx-channel (in this case a b-channel) syntax is <wanpipe span number>:<range of channels>. Because in Wanpipe the devices are created per span, the channel count is not global for all spans, but rather per span (note the difference with DAHDI where the channel numbers keep growing across spans).

This is an example for 2 spans (span 7 and 10) that will be used as a T1 PRI link.

```
[span wanpipe PRI_1]
trunk_type => T1
b-channel => 7:1-23
d-channel => 7:24

[span wanpipe PRI_2]
trunk_type => T1
b-channel => 10:1-23
d-channel => 10:24
```

Converting that configuration to E1 is simple:

```
[span wanpipe PRI_1]
trunk_type => E1
b-channel => 7:1-15
d-channel => 7:16
b-channel => 7:17-31

[span wanpipe PRI_1]
trunk_type => E1
b-channel => 10:1-15
d-channel => 10:16
b-channel => 10:17-31
```

An FXO analog span with two PSTN interfaces (2 jacks) where the wanpipe span is number 1 (in Linux, think /dev/wanpipe1_if1 and /dev/wanpipe1_if2) would look like this:

```
[span wanpipe FXO]
trunk_type => FXO

txgain => 3.5
rxgain => 7.5
fxo-channel => 1:1

txgain => 0.5
rxgain => 8.5
fxo-channel => 1:2
```

An FXS analog span with two analog phone interfaces would look like this:

```
[span wanpipe FXS]
trunk_type => FXS
txgain => 3.5
rxgain => 7.5
fxs-channel => 1:3

txgain => 0.5
rxgain => 8.5
fxs-channel => 1:4
```

The wanpipe spans must have been previously configured in /etc/wanpipe/ by Sangoma configuration tools (check http://wiki.sangoma.com/ for details). Keep in mind only spans configured as *TDM_VOICE_API* in *wanpipe*.conf* can be used together with spans configured with I/O type *wanpipe* in *freetdm.conf*

You can get full examples here: Freetdm.conf_Examples

DAHDI mode

DAHDI mode can be used by any hardware that implements the DAHDI interface, including Sangoma and Digium cards (yes Sangoma supports both Wanpipe and DAHDI modes in FreeTDM) and Redfone foneBRIDGE2 T1/E1 to Ethernet bridges. Sangoma cards are better used in Wanpipe mode. The only scenario where you may need to run a Sangoma card using DAHDI interface is when your particular card does not have HDLC engine (like Sangoma B601) and you need HDLC performed in the software drivers. Because the wanpipe drivers still do not perform the HDLC in software, if you have a Sangoma B601 you must use DAHDI mode, otherwise please use Wanpipe mode for better support with Sangoma cards.

Digium cards on the other hand should always be used in DAHDI mode.

DAHDI mode is most likely compatible with Zaptel mode, but Zaptel is deprecated. This mode depends on "ftmod_zt.so", which is always compiled, regardless of whether you have DAHDI or Zaptel headers. On runtime, FreeTDM will try to determine (checking the existence of /dev/zap/ctl or /dev/dahdi/ctl) which one are you using.

In a typical DAHDI installation, the freetdm.conf file will be a basic mirror of the contents /etc/dahdi/system.conf (or /etc/zaptel.conf if using zaptel), but you still need them because they are used by dahdi_cfg or ztcfg to configure the hardware.

As an example, for a range of channels from 1 to 23 in span 1, devices ranging from /dev/zap/1 to /dev/dahdi/23 will be created in Linux file system and you can add them to freetdm.conf like this:

```
[span zt trunk1]
trunk_type => T1
b-channel => 1-23
```

The key here is "zt" is set as the I/O type in the span definition and the xx-channel (in this case a b-channel) value is just a range of channels. Because in DAHDI the channel count is global for all spans, so the next span will start at channel 25 (assuming span 1 has 24 channels).

This is an example for 2 spans (span 1 and 2) that will be used as a T1 PRI link.

```
[span zt PRI_1]
trunk_type => T1
b-channel => 1-23
d-channel => 24

[span wanpipe PRI_2]
trunk_type => T1
b-channel => 25-47
d-channel => 48
```

As you see, in DAHDI the syntax for the xx-channel parameters is just a range of channels which is global for the whole system, no span is specified (as opposed to wanpipe mode where the span is specified).

You can get full examples here: Freetdm.conf_Examples

'**Note:**' If your DAHDI system.conf file has lines like these ....

```
bchan=1-15,17-31
dchan=16
```

Then setup your freetdm.conf file to have lines like these ...

```
b-channel => 1-15
d-channel => 16
b-channel => 17-21
```

You can setup freetdm.conf to have channels like the system.conf, however in at least one instance this caused problems with calls over channel 16. Splitting the channels out over a few lines solved that issue.

freetdm.conf.xml Configuration

Once the library is configured, if you are using FreeSWITCH (other users may use only the library to build custom TDM applications), you have to tell FreeSWITCH which signaling settings to use and where to send the incoming calls to. This is done through /usr/local/freeswitch/conf/autoload_configs /freetdm.conf.xml (assuming you have default installation path for FreeSWITCH).

The autoload_configs/freetdm.conf.xml will then look like this:

```
<configuration name="freetdm.conf" description="FreeTDM Configuration">
  <settings>
    <param name="debug" value="0"/>
    <param name="sip_headers" value="true"/>
    <!--<param name="hold-music" value="$${moh_uri}"/>-->
    <!--<param name="enable-analog-option" value="call-swap"/>-->
    <!--<param name="enable-analog-option" value="3-way"/>-->
  </settings>
   <pri_spans>
     <span name="myPRI">
       <!-- Log Levels: none, alert, crit, err, warning, notice, info, debug -->
       <param name="q921loglevel" value="alert"/>
       <param name="q931loglevel" value="alert"/>
       <param name="mode" value="user"/>
       <param name="dialect" value="q931"/>
       <param name="dialplan" value="XML"/>
       <param name="context" value="pstn"/>
     </span>
   </pri_spans>

   <analog_spans>
     <span name="myFXO">
       <!--<param name="hold-music" value="$${moh_uri}"/>-->
       <param name="dialplan" value="XML"/>
       <param name="context" value="Incoming-FXO"/>
       <param name="tonegroup" value="fr"/>
       <param name="enable_callerid" value="true"/>
       <param name="enable-analog-option" value="call-swap"/>
       <!--<param name="enable-analog-option" value="3-way"/>-->
     </span>

     <span name="myFXS">
       <!--<param name="hold-music" value="$${moh_uri}"/>-->
       <param name="dialplan" value="XML"/>
       <param name="context" value="default"/>
       <param name="tonegroup" value="fr"/>
       <param name="enable_callerid" value="true"/>
       <param name="enable-analog-option" value="call-swap"/>
       <param name="dialplan" value="XML"/>
       <param name="context" value="Incoming-FXS"/>
     </span>
```

```
   </analog_spans>

   <config_profiles>
     <profile name="my_pri_te_e1_1">
       <param name="switchtype" value="euroisdn" />
       <param name="interface" value="cpe"/>
     </profile>
   </config_profiles>

   <sangoma_pri_spans>
     <span name="wp1" cfgprofile="my_pri_te_e1_1">
       <param name="dialplan" value="XML"/>
       <param name="context" value="pstn"/>
       <param name="tei" value="<*0..127>"/>
       <param name="overlap" value="<yes|*no>"/>
       <param name="min_digits" value="4"/>
       <param name="setup arbitration" value="<yes|*no>"/>
       <param name="facility" value="<yes|*no>"/>
       <param name="facility-timeout" value="0"/>
       <param name="outbound-called-ton" value="<*unknown|international|national|network-specific|subscriber-
number|abbreviated-number|reserved>"/>
       <param name="outbound-called-npi" value="<*ISDN|data|telex|national|private|reserved|unknown>"/>
       <param name="outbound-calling-ton" value="<*unknown|international|national|network-specific|subscriber-
number|abbreviated-number|reserved>"/>
       <param name="outbound-calling-npi" value="<*ISDN|data|telex|national|private|reserved|unknown>"/>
       <param name="outbound-rdnis-ton" value="<*unknown|international|national|network-specific|subscriber-
number|abbreviated-number|reserved>"/>
       <param name="outbound-rdnis-npi" value="<*ISDN|data|telex|national|private|reserved|unknown>"/>
       <param name="outbound-bearer_cap" value="<*speech|unrestricted-digital|3.1Khz>"/>
       <param name="outbound-bearer_layer1" value="<v110|ulaw|alaw>"/>
       <param name="local-number" value=""/>
     </span>
   </sangoma_pri_spans>

</configuration>
```

" * " defaults

The general syntax for spans inside freetdm.conf.xml is:

```
<span name="spanName">
    <param name="dialplan" value="XML"/>
    <param name="context" value="default"/>
    <param name="xxx" value="xxx"/>
</span>
```

The name attribute of the tag *MUST* match a span name in freetdm.conf, this is how FreeSWITCH knows which span you want to configure with that signaling.

Many parameters on the section depend of the signaling module section where they live. Every span must be inside a signaling module tag. Each signaling module has its own unique tag identifier. The dialplan and context parameter are accepted by all signaling modules. But other than that every span must be configured with parameters supported by its signaling module. These are the signaling module tags that you can use in freetdm.conf.xml

- <analog_spans> </analog_spans> - Spans will be configured by the signaling module ftmod_analog.
- <analog_em_spans> </analog_em_spans> - Spans will be configured by the signaling module ftmod_analog_em.
- <pri_spans> </pri_spans> - Spans will be configured by the native FreeTDM ISDN stack ftmod_isdn.
- <libpri_spans> </libpri_spans> - Spans will be configured by the libpri module ftmod_libpri.
- <pritap_spans> </pritap_spans> - Spans will be configured by the PRI tapping module ftmod_pritap.
- <sangoma_pri_spans> </sangoma_pri_spans> - Spans will be configured by the Sangoma ISDN signaling module ftmod_sangoma_isdn.
- <sangoma_bri_spans> </sangoma_bri_spans> - Spans will be configured by the Sangoma ISDN signaling module ftmod_sagoma_isdn.
- <sangoma_ss7_spans> </sangoma_ss7_spans> - Spans will be configured by the Sangoma SS7 signaling module ftmod_sangoma_ss7.
- <r2_spans> </r2_spans> - Spans will be configured by the OpenR2 MFC-R2 signaling module ftmod_r2.
- <boost_spans> </boost_spans> - Spans will be configured by the custom protocol module ftmod_sangoma_boost to access SS7, PRI and BRI lines.

Refer to the samples included in the freetdm sources inside the conf/ directory for more information on the parameters for the spans on each signaling section.

There are a lot of fields/parameters to configure - for a full list look at the source code of appropriate module. For example, for ftmod_sangoma_isdn - look at [FSW_dir/libs/freetdm/src/ftmod/ftmod_sangoma_isdn/ftmod_sangoma_isdn_cfg.c :: ftmod_isdn_parse_cfg(2)]. For example, to send CONNECT ACK for outgoing calls (which is optional according to ITU-T Q.931, Figure A.2/Q.931 – Overview protocol control) use "send-connect-ack" parameter.

Here is one working example for HFC-S BRI PCI cards. There are still problems with other devices on the S0-bus, but it works as a single device on the S0 so far.

```
<configuration name="freetdm.conf" description="FreeTDM Configuration">
  <settings>
    <param name="debug" value="0"/>
    <!--<param name="hold-music" value="$${moh_uri}"/>-->
    <!--<param name="enable-analog-option" value="call-swap"/>-->
    <!--<param name="enable-analog-option" value="3-way"/>-->
  </settings>

   <libpri_spans>
     <span name="ZTHFC1">
        <param name="node" value="cpe"/>
        <param name="switch"   value="euroisdn"/>
        <param name="opts" value="omit_redirecting_number"/>
        <param name="dp" value="unknown"/>
        <param name="debug" value="none"/>
        <!-- <param name="debug"    value="q931_all"/> -->

        <param name="dialplan" value="XML"/>
        <param name="context" value="public"/>
      </span>
   </libpri_spans>
</configuration>
```

tones.conf

- see tones.conf_Examples

Some components in freetdm (mainly the ftmod_analog module) uses tones.conf for tone generation and detection. The dial tone is particularly crucial for dialing out. If you use analog interfaces and you want to dial out you must make sure the configuration in tones.conf is correct for your country, otherwise FreeTDM won't dial in lines without dial tone (or a dial tone that cannot detect).

Dial Plan

When using mod_freetdm to use FreeTDM with FreeSWITCH. You need to add dial plan rules to use your FreeTDM lines.

```
<extension name="outgoing-pri">
  <condition field="destination_number" expression="^.+$">
    <action application="bridge" data="freetdm/1/a/${destination_number}"/>
  </condition>
</extension>

 <extension name="outgoing-FXO">
  <condition field="destination_number" expression="^.+$">
    <action application="bridge" data="freetdm/1/1/${destination_number}|freetdm/1/2/${destination_number}"/>
  </condition>
</extension>

Then to bridge an FXS channel to FXO channel
 <extension name="bridge-FXS-to-FXO">
  <condition field="destination_number" expression="^.+$" />
  <condition field="chan_name" expression="FreeTDM/1/[34]" >
    <action application="bridge" data="freetdm/1/1/${destination_number}|freetdm/1/2/${destination_number}"/>
  </condition>
</extension>
```

Ascending vs. Descending Channel Selection

FreeTDM supports automatic selection of the channel within the span. Use "a" for ascending (lowest-numbered channel available) and "A" for descending (highest-numbered channel available).

Examples:

```
<!-- Ascending, i.e. start at channel 1 and work up -->
<action application="bridge" data="freetdm/1/a/${destination_number}>

<!-- Descending, i.e. start at "the top" (e.g. 23 in T1) and work down -->
<action application="bridge" data="freetdm/1/A/${destination_number}>
```

Sangoma A200 with Software Echo Cancellation

If you have a non-D series A200 card, it will be unusable without software echo cancellation. As of 03/11 the native wanpipe docs on the sangoma wiki do not account for this. It is not possible to use a Sangoma A200 series with software echo cancellation in "Wanpipe-native" mode. Instead, use wanpipe with dahdi. (Also, hangup detection may not work without software echo cancellation. It may work better once mg2 cancellation is working, and it may just work right with oslec. This has been reported with Gentoo-64.)

HINT: Build in wanpipe-native mode, then return here and try this:

STEPs 1: Compile and install dahdi, dahdi-tools, and libpri

STEPs 2: Compile and install the latest wanpipe driver with "./Setup dahdi" in the wanpipe driver source dir. Ignore the parts about "Asterisk"

... same thing with Oslec instead of mg2 software echo cancellation

Follow the steps above, but first patch dahdi as per oslec docs. Hint: don't download dahdi-complete. Download dahdi and dahdi-tool separately.

STEP 3: Set "echocanceller=oslec,X", where X is the span number of your A200, in /etc/dahdi/system.conf.

Sangoma G.722 ISDN calls

As of May 19th 2011, Sangoma cards may be used to make and receive TDM calls using G.722 codec (or any other codec that can fit in the TDM 64kbps data stream).

The way it works is based on the "unrestricted digital" bearer capability available in ISDN circuits. A typical voice call will be sent to the telco (or the telco will send it to us) using the default bearer capability "speech". However, using "unrestricted digital" is understood that the data in the B channel could be just about anything. You must configure FreeSWITCH to decide the codec to use, in this case G.722. In your Sangoma span XML configuration (for example, just below <param name="context" value="xx" />) you set this parameter:

```
<param name="unrestricted-digital-codec" value="G722@8000h" />
```

That instructs FreeTDM to setup G.722 codec for all unrestricted digital calls.

In order to originate calls with bearer capability "unrestricted digital", you must call the bridge application like this:

```
<action application="bridge" data="{freetdm_bearer_capability=1}freetdm/xxx" />
```

The key point is the {freetdm_bearer_capability=1}, where 0 is speech (default) and 1 is unrestricted digital.

FreeSWITCH/FreeTDM CLI commands

Delivered by mod_freetdm

If mod_freetdm is loaded inside FreeSWITCH, you have many FreeTDM commands that you can use.

To see a list of your configured spans, at the CLI type

```
ftdm list
```

To see detailed information about a specific span

```
ftdm dump <span_id>
```

To see detailed information about a specific channel on span

```
ftdm dump <span_id> <chan_id>
```

To start a span (this terminates FS with sangoma_isdn when you stopped the span before)

```
ftdm start <span_id>
```

To stop a span

```
ftdm stop <span_id>
```

To get a pcap trace (I guess it doesn't work currently)

```
ftdm q931_pcap <span_id> on|off [pcapfilename without suffix]
```

(this command shouldn't be defined in mod_freetdm, but in ftmod_isdn I guess)

To enable/disable DTMF detection on whole span or certain channel

```
ftdm dtmf on|off <span_id> [<chan_id>]
```

To enable a trace on whole span or certain channels

```
ftdm trace <path where to save files per channel> <span_id> [<chan_id>]
```

To disable a trace on whole span or certain channels

```
ftdm notrace <span_id> [<chan_id>]
```

To adjust gains on whole span or certain channels

```
ftdm gains <rxgain> <txgain> <span_id> [<chan_id>]
```

All other commands are send currently to ftmod's api!

Delivered by ftmod_sangoma_isdn API

ftmod_sangoma_isdn is basing on Sangoma's libsng_isdn library. You have to download and install it befor you run FS's configure script.

To enable partial runtime decoding of q921 or q931 messages on a span

```
ftdm sangoma_isdn trace <q921|q931> <span_name>
```

Call this for both q921 and q931 to have both decoded

To disable decoding of q921 and q931 messages on a span

```
ftdm sangoma_isdn trace disable <span_name>
```

To get layer 1 statistics from span (derived from wanpipe I guess)

```
ftdm sangoma_isdn l1_stats <span_name>
```

To get a short status report of each span

```
ftdm sangoma_isdn show_spans
```

SIP-Headers

If parameter "sip_headers" is enabled in freetdm.conf.xml the you can control mod_freetdm with the following headers:

- X-FreeTDM-CallerName
- X-FreeTDM-CallerNumber
- X-FreeTDM-ANI
- X-FreeTDM-ANI-TON
- X-FreeTDM-ANI-Plan
- X-FreeTDM-ANI2
- X-FreeTDM-DNIS
- X-FreeTDM-DNIS-TON
- X-FreeTDM-DNIS-Plan
- X-FreeTDM-RDNIS
- X-FreeTDM-RDNIS-TON
- X-FreeTDM-RDNIS-Plan
- X-FreeTDM-Screen
- X-FreeTDM-Presentation

Channel Variables

Info

- freetdm_span_name
- freetdm_span_number

FreeTDM span number for an incoming call.

- freetdm_chan_number

FreeTDM channel number for an incoming call.

- ftdm_usage 1 1

Checks if span 1 channel 1 has 0 calls so we know if the line is free to make a call. Note there is an inherent small race between the time you check and the time you try to use the channel, but likely good enough for the real world. Could be applied in dialplan as:

```
<condition field="${ftdm_usage 1 1}" expression="^0$">
</condition>
```

Control

- freetdm_bearer_capability
- freetdm_bearer_layer1
- freetdm_outbound_ton
- freetdm_custom_call_data

Applications

- disable_ec

Disable echo cancelation on this call.

- disable_dtmf

Disable DTMF detection on this call.

- enable_dtmf

Enable DTMF detection on this call.

Mailing List FAQ

This is a collection of questions and answers from the FreeSWITCH mailing list.

FreeTDM partial spans, group dialing and trunks

http://lists.freeswitch.org/pipermail/freeswitch-users/2010-October/063519.html

The question:

```
I have oodles of FXO ports configured on Xorcom Astribanks. On the Astribank
each set of 8 FXO ports forms a zaptel/openzap/freetdm span.

Now if a trunk consists of an integral multiple of 8 FXO ports it is easy to
just dial out on the first available channel:

<action application="bridge" data="freetdm/1/a/$1|freetdm/2/a/$1"/>

This grabs the first available channel out of 16 FXO ports.

But what is the "right" way to do it when I need to use 1.5 spans (i.e. 12
FXO ports)?

Is there such a thing as a virtual span that can be built out of individual
FXO ports?
```

The answer:

```
You can create spans with channels from any other span (as long as the
signaling is the same).

[span zt xorcomSpan]
; channels from 1.5 trunks
fxo-channel => 1-12

Then dial:

<action application="bridge" data="freetdm/xorcomSpan/a/$1/>

You can also just define a group for those channels and use the group
for dialing.

[span zt xorcomTrunk1]
trunk_type => FXO
group => xorcomg1
fxo-channels => 1-8

[span zt xorcomTrunk2]
trunk_type => FXO
group => xorcomg1
fxo-channels => 9-12

Then dial using the group name.

<action application="bridge" data="freetdm/xorcomg1/a/$1/>

This means spans and groups are strings within the same name space (as
far as hunting is concerned), and you should not make them conflict.
```

The question in this thread refers to "virtual spans". In FreeTDM, all spans declared in freetdm.conf are virtual or logical, however you want to call them. Meaning the FreeTDM span structure it's just a channel container. The actual channels may belong to different physical spans. You can do things like:

freetdm.conf

```
[span wanpipe superWanpipeSpan]
b-channel => 1:1-15
b-channel => 1:17-31
b-channel => 2:1-15
b-channel => 2:17-31
```

This freetdm.conf file creates a span containing B channels from 2 physical E1's. A span of type "wanpipe" was used for this example because is clearer that channels from 2 spans are being put in a single span. For zt I/O spans this looks like:

freetdm.conf

```
[span zt superZtSpan]
b-channel => 1-15
b-channel => 17-31
b-channel => 32-46
b-channel => 48-62
```

This super spans, or logical spans that are composed of channels across physical spans are actually used for some signaling types like "boost", which requires all b-channels in a single span. That module however is deprecated, and is not recommended to group channels like that (because some other signaling modules assume all channels in a span come from a single physical span).

Note that, although for analog modues it's supported to add channels from any physical span to any FreeTDM span, it's not recommended for other signaling modules (like PRI, BRI). Group dialing was introduced to stop using FreeTDM logical spans as dialing groups. If you need to group channels for dialing purposes, use the "group" parameter to add the channels to a given group.

FreeTDM Screening Indicator

http://lists.freeswitch.org/pipermail/freeswitch-users/2010-October/064539.html

The question:

```
Hi,
I have a server with FreeSWITCH (latest GIT revision) + FreeTDM with some
Sangoma A108 boards.
I have the necessity to set, for outgoing calls, the ISDN information
element "Screening Indicator" of calling party number.
I have see that some information element can be set in the
"freetdm.conf.xml" file (i.e. calling and called typer of number or
numbering plan) but I have not found any reference to the screening
indicator.
If there is no way to specify this information element for each trunk, it's
possible to change its default value (that is 01=user provided, verified and
passed)?
Thanks.
```

The answer:

```
To check the value on incoming call, try the screen_bit variable from the dial plan:

${screen_bit} which will be either true or false.

To set it for outgoing calls, try using the privacy application that FreeSWITCH has.

<action application="privacy" data="on" />
```

FreeTDM Sangoma PRI rx errors

http://lists.freeswitch.org/pipermail/freeswitch-users/2010-October/064569.html

The question:

```
After installing FreeSWITCH and Sangoma, I am getting following error in fs_cli

2010-10-27 17:57:03.228457 [ERR] ftmod_sangoma_isdn_stack_rcv.c:883 sng_isdn->s1:L1 Rx Error
2010-10-27 17:57:03.228457 [DEBUG] ftmod_sangoma_isdn_stack_rcv.c:871 sng_isdn->s1: Resetting L1 link

What does these mean?
```

These errors mean we're seeing errors on L1 (the T1/E1 link).

To troubleshoot, it is advised:

```
Can you check what type of L1 errors you are seeing:

from FS cli:
ftdm sangoma_isdn l1_stats wp1

Can you also check if wanpipemon is reporting errors:

wanpipemon -i w1g1 -c Ta

(repeat a couple times to see if any of the alarms are continuously
toggling or the number of errors at the bottom keep incrementing)
```

How to identify whether a port in a card is a problem

The question:

```
How to check whether there is a problem with the ports in the Sangoma card that you have?
```

The Answer:

```
You need to check by using loopback.

Single port loopback:

            +-------+
            |       |
        1   2   4   5
        |       |
        |       |
        +-------+
You need to configure the port as MASTER to get the clock signal. Otherwise tx will expect clock
from rx and rx will expect clock from tx, and port won't show "Connected" status.

If it shows "Connected" then most probably there is no problem in the port
```

```
If you are using a CSU with the sangoma card, you can simply put the CSU in loopback mode and start
sending out the test patterns
```

What is YELLOW alarm

The question:

```
What is meant by YELLOW alarm? Will I be able to simulate to get better understanding?
```

The Answer

Of-course yes. You can generate the YELLOW alarm as follows.
Make a Loopback cable as follows:

Configure the clock of PORT A as "NORMAL" and PORT B as "MASTER".
Remove the 1 and 2 pins, in PORT B. You will get "YELLOW" alarm on PORT B.
In PORT A, it will have "Remote Alarm Indication" = "ON". (wanpipemon -i w1g1 -c Ta)
PORT A will be in "Connected" state (since it gets the packet sent by PORT B.
PORT B will be in "Disconnected" state.

Here is the explanation:

Once PORT B is unable to receive any message from PORT A, it will send a indication saying that "I'm unable to receive you" in the Tx (pins 4,5), and it will set the "YELLOW" alarm on. Then PORT A finds that, PORT B is not receiving packets which is sent by PORT A, and it will enable the "RAI". (Red Alarm Indicator). Alarm recovery time can vary from equipment to equipment. Yellow alarm is also known as AIS on some equipment.