revision 0.2_pre - August/12/2006

Moisés Humberto Silva Salmerón <moy@ivsol.net>

This documents is released under the most recent version of GNU documentation license (GNU FDL). More information about the license can be found at http://www.gnu.org/licenses/fdl.txt

NOTES ABOUT MFCR2 AND UNICALL WITH ASTERISK.

About this document.

This document is based in my personal experience using MFCR2 signaling to connect Asterisk with Avantel telco. Im not a signaling expert and the document is prone to errors. The signaling table provided is based in the analysis of a signaling trace sent to me my Avantel when we were trying to detect the errors we had in the call setup. Any contribution to the document will be appreciated.

¿What is MFCR2 anyway ?

Multi Frequency Compelled Region 2 signaling. That is a signaling that is based on audible tones transmision. R2 signaling is of CAS (Channel Associated Signaling) type and was developed in the 60's, but still used in Europ, Asia, Latin America and Australia. It is pseudo-defined by the ITU (International Telecommunications Union). By "pseudo-defined" I mean that is defined but each country makes a variant of the signaling to fit their needs. That is, R2 does not work exactly the same way in all countries. A simple definition of R2 is, a signaling used to the setup of calls. This is shown in the following diagram:

Telecommunication Center1 <========= SignalingMFCR2 ==========> Telecommunication Center 2

The are 3 main groups of R2 signaling, those are:

Digital R2, Usually used in E1 trunks for PCM (Pulse Coded Modulation) systems, in Mexico, we use ALAW as codification method.
Analog R2.
Pulse R2, used in slow transmissions, like satelital connections.

¿So, how does it works?

MFCR2 is a "peer to peer" signaling. Wich it means that only 2 peers are involved and both sides of the connection are considered equal (no client -> server model). Here both sides of the link operate the same way. R2 use the digital facilities of the E1 lines to setup the calls. Is beyond the scope of this document (and the author skills) to fully explain how E1 works, however we can say that is a transmission method that segments a single physical line into 32 channels (time slots), referred as TS0 - TS31. From this 32 channels, the channel 0 is used for information like the framing, synchrony, alarms and specific country stuff. Channels from 1-15 and 17-31 are used for voice or data. Channel 16 is for signaling , dont confuse the E1 signaling with R2 signaling. R2 is a higher level signaling that will travel using the E1 facilities. R2 signaling will travel within each of the voice time slots; ie; You can configure only

10 of the 31 E1 available channels to be a voice channels using R2 signaling, and the rest for data or other purposes. R2 use 2 types of signals:

Supervisory Signals
This signals take care of the line state (ie; IDLE, BLOCKED). Use 4 bits code, commonly known as ABCD bits. Despite that are 4 bits, usually only 2 of them (AB) are used.

Interregister Signals.
This signals are transmitted using audible multi frequency tones (MF tones). The ANI and DNIS digits are transmitted at the start of a call using this signals.

Lets take a look to supervisory signals first. The supervisory signals are very simple. The next table describes the 2 types of supervisory signals (Forwards and Backwards). Forwards means that is the signaling as seen from the caller to the callee. Backwards means the oposite. The following table corresponds to the signaling beetween Ericsson Avantel central and my GNU/Linux server using libmfcr2 (software R2 C library) wrote by Steve Underwood.

Forwards ABCD	Nombre	Backwards ABCD	Nombre
1 0 0 1	Idle	1 0 0 1	Idle
1 0 0 0	Seizure	1 1 0 1	Seizure Acknowledgment
1 0 0 1	Clear Back	1 0 0 1	Clear Back
1 0 0 1	Clear Forward	1 0 0 1	Clear Forward
0 1 0 1	Answer	0 1 0 1	Answer
1 1 0 1	Blocking	1 1 0 1	Blocking

Seizure
This signal is sent to request a new call.

Seizure Ack
Ack sent to confirm the reception of the Seizure signal, this Seizure ACK means that the other end can start with the multifrequency signaling (MF interregister signals)

Answer
Tell the other end that the call has been answered.

Clear Backwards
Calle has hanged up.

Clear Forwards
Caller has hanged up

Blocking
Not accepting calls at this moment.

Interregister signals are used to setup the call with more call details, like the ANI and DNIS. The ANI (Automatic Number Identification) is the service to identify the caller, better known as "CallerID". The DNIS (Dialed Number Identification Service) is the service provided to identify the requested DID (Direct Inward Dial), that is, the number that was dialed by the caller.

There are 3 types of MF signals:

R2-Compelled: This means that when a tone pair is sent from the switch (Forward), the signal will stay there until it receives other tone pair as ACK. When the switch receives the ACK, quits sending the tone pair. The other end must detect this silence condition and quit their own tone pair used as ACK. The switch will detect the silence condition generated by the other end and will send the next tone pair signal, and so on.

R2-Non-Compelled: The same as in R2 compelled, but the tone pairs are sent as pulses.

R2-Semi-Compelled: A mix between compelled and non compelled. Forward signals are sent as compelled (continuous tone) and Backwards signals are sent as Non-compelled (pulses).

Any further reference to R2 in this document it means "R2-Compelled".

The interregister signals are "inband", what it means that the signals are sent in the same channel as voice/data. Forward interregister signals are divided in Group I and Group II. Backward interregister signals are divided in Group A and Group B. The next table shows this signals. At the end of the document we include a signaling trace between the Avantel Ericcson central and my GNU/Linux server with libmfcr2 installed.

Señales Forward	Señales Backward
Group I Used for ANI and DNIS transmission. Digits are represented from 1 - 10, 10 means 0, 15 means end of digits.	Group A Used for confirm digits reception and signal group change. A-1 Next Digit A-3 End of digits (Address Complete) and change to signals of group B. A-4 Congestion A-5 Send caller category A-6 End of digits.
Group II Used to transmit the caller category I-1 Suscriptor without priority. I-2 a I-9 Suscriptor with priority. I-11 a I-15 National use.	Group B Used to confirm Group II signals and to provide destiny information. This signals are always used after A-3 signal (See Group A signals) B-1 Open Audio Path B-3 Busy Line B-4 Congestion B-5 Unknown Number B-6 Free charge line

How MFC/R2 fits into Asterisk?

In the best of the cases, put MFCR2 to work with Asterisk is simple. There are information in voip-info.org about it. But sometimes things just "dont work", and we have to stop and make some analysis to determine why is not working. In my case, I decided to write this document because I spent about 2 weeks having problems because of a small programming error in the software library that interprets R2 signaling (libmfcr2). More information in spanish about this error and how I solved it can be found at http://moy.ivsol.net/, but some of you may have enough knowing that Steve Underwood has released some time ago a fixed version.

In the following schema we can see how MFCR2 fits in GNU/Linux and how talks to Asterisk and Zaptel drivers.

[Low Level] [High Level]

spandsp / libsupertone

PSTN <----> zaptel card <-----> zaptel driver <----> kernel linux <-----> libmfcr2 <------> libunicall <-----> chan_unicall <----> Asterisk

Zaptel drivers are "character devices" registered in the linux kernel. The driver registers some IOCTLs that the kernel exposes to user space applications can communicate with the Zaptel cards. In this case libmfcr2 talks to the kernel using the driver IOCTLs to be notified about the ABCD bits, information reception, etc in the zaptel spans. In the same way, libmfcr2 use system calls like write() to send information (voice/data) using the zaptel file descriptors (/dev/zap/[channel-number]). When some change occurs in the ABCD bits so they represent a "Seizure" state (See Supervisory signals), libmfcr2 communicates with libunicall (Telephony abstraction library) to start a new call. The libunicall library exposes a well defined API to handle calls in a signaling independent way, MFCR2 is just one signaling for calls, but others can be used with Unicall as well. Unicall has the responsibility of communicate to a higher level application all the low level events reported by libmfcr2. libmfcr2 use other library called "spandsp" to identify and respond to the MF tones of the R2 signaling. The higher level application wich communicates with Unicall can be "chan_unicall" that becomes a part of Asterisk PBX. But other higher application that we will see is "testcall". Common telephony tones can be used by libmfcr2 using the "libsupertone" library, to generate tones like busy, disconnected etc. This tones can be generated too by Asterisk, see /etc/asterisk/indications.conf for more information.

From this short intercommunication description we can make 2 important conclusions:

1. libunicall is not limited to R2 signaling neither to Asterisk. In this case can be put to work in team with libmfcr2 in the lower level, and Asterisk (chan_unicall) in the higher level. Below libunicall we have libmfcr2 (using spandsp, libsupertone) used to talk R2 signaling through zaptel devices. And up to libunicall we have Asterisk/chan_unicall, a higher level application that will "make usefull things" with the call. But is evident that is also possible to use other PCI cards and telephony signaling without changes to Asterisk, since Asterisk will continue to be happily talking with the same interfaces of libunicall, never realizing that libmfcr2 and zaptel are no longer used. However, this is partially truth, because some small parts of chan_unicall make use of IOCTLs of zaptel cards. For a clean hardware abstraction, these should be removed. Regarding signaling, libunicall seems to be a very clean abstraction.

2. The second conclusion is that libunicall can be used by other higher level applications. That means Asterisk is only a simple user of libunicall. Other libunicall user is "testcall" a small application included with Unicall distribution. This simple application only generates and receives calls, just like Asterisk do through the use of chan_unicall. The difference is that Asterisk can bridge the calls for usefull purposes with SIP/IAX/H323/technologyX channels and other more complex services. Testcall application is very usefull to isolate tests on mfcr2 signaling. All this means that we is possible to use libunicall, libmfcr2, spandsp, libsupertone with other higher level applications like YATE (yet another telephony engine, http://yate.null.ro/pmwiki/) or FreeSwitch (http://www.freeswitch.org).

Configuration And Troubleshooting.

Before doing anything, remember that any configuration or installation stuff must be done with root privileges. For troubleshooting and testing we will be using 2 important tools: "testcall" and "zttool". The first can be found in Unicall sources, and can be compiled executing "make testcall" in the Unicall sources directory. The second tool can be found in zaptel drivers sources and build it executing "make zttool". As we saw earlier, testcall is used for debugging unicall and mfcr2 signaling, and zttool is usefull to be the status of the zaptel cards and their ABCD bits. Now, with these tools ready to be used, we are going now to describe how all the components should be configured to make and receive calls through E1 zaptel cards using CAS and MFCR2 as signaling.

zaptel.conf

zaptel.conf is the zaptel drivers configuration file. Is read to configure the hardware with the proper signaling for many purposes (remember, zaptel cards are used even for non-telephony purposes). The file is read with the command 'ztcfg', it has some optional arguments like "ztcfg -vv" to increment verbosity about what is going on. Here is a simple configuration:

```
span=1,1,0,cas,hdb3
cas=1-10:1101
dchan=16
defaultzone=us
loadzone=us
```

This configuration is used by one of my servers with a 4 port digium card. I only use one of the ports. In E1 cards, a span can be seen as 1 port. This is not true for FXO cards, in wich the card can have 4 ports, but the zaptel driver reports only 1 span. Thus, you can see a span as a driver defined group of channels. For E1 cards, each span has 31 channels. You can check how many spans the zaptel drivers are creating by checking files in the system proc directory, like this:

```
ibbmexico ~ ls /proc/zaptel/
1  2  3  4
ibbmexico ~
```

That will show several files named with numbers. Each of these numbers represents 1 zaptel span. As I said, each zaptel span represents a group of channels that the card driver register in the kernel as character device. However, information in /proc/zaptel is only usefull for informational purposes. Real programming with zaptel devices use the files in /dev/zap, that are the actual file descriptors that can be read(), write() etc. The library libmfcr2 uses these file descriptors to interpret the MFCR2 signaling protocol. To read information about the spans you can execute this:

cat /proc/zaptel/x

Where "x" is the Span number. That will list usefull information about the span. Like this:

```
ibbmexico # cat /proc/zaptel/1
Span 1: TE4/0/1 "T4XXP (PCI) Card 0 Span 1" HDB3/ ClockSource

           1 TE4/0/1/1 CAS (In use)
           2 TE4/0/1/2 CAS (In use)
           3 TE4/0/1/3 CAS (In use)
           4 TE4/0/1/4 CAS (In use)
           5 TE4/0/1/5 CAS (In use)
           6 TE4/0/1/6 CAS (In use)
           7 TE4/0/1/7 CAS (In use)
           8 TE4/0/1/8 CAS (In use)
           9 TE4/0/1/9 CAS (In use)
          10 TE4/0/1/10 CAS (In use)
          11 TE4/0/1/11
          12 TE4/0/1/12
          13 TE4/0/1/13
          14 TE4/0/1/14
          15 TE4/0/1/15
          16 TE4/0/1/16 HDLCFCS
          17 TE4/0/1/17
          18 TE4/0/1/18
          19 TE4/0/1/19
          20 TE4/0/1/20
          21 TE4/0/1/21
          22 TE4/0/1/22
          23 TE4/0/1/23
          24 TE4/0/1/24
          25 TE4/0/1/25
          26 TE4/0/1/26
          27 TE4/0/1/27
          28 TE4/0/1/28
          29 TE4/0/1/29
          30 TE4/0/1/30
          31 TE4/0/1/31
ibbmexico #
```

That will show you all the channels of the span and some other information about the signaling and status of the channels. The image above, is a healthy zaptel span running CAS signaling in their first 10 channels, and currently "In Use", that means some application is using those channels, in this case, the application is Asterisk. If you pay attention, the channel number 16, shows HDLCFCS signaling, thats because channel 16 cannot be used for voice, just for E1 signaling. If I stop asterisk, the (In Use) messages would disappear. Lets read the first line of the file:

"Span 1: TE4/0/1 "T4XXP (PCI) Card 0 Span 1" HDB3/ ClockSource"

This tell us that the span 1 belongs to the card 0 model TE4 PCI has HDB3 coding and is used as clock source. Now lets read a body line:

"5 TE4/0/1/5 CAS (In use)"

This means, channel 5 belongs to card 0 model TE4, span1 and is currrently being used, but not necessarily that is in a call, just means some application (in this case Asterisk) has opened the channel file descriptor. Remember that Zaptel cards are not only used for telephony purposes.

Lets see how zaptel.conf file works, and how should be configured to have MFCR2 working. This is the zaptel.conf file of one working system:

```
span=1,1,0,cas,hdb3
cas=1-10:1101
dchan=16
defaultzone=us
loadzone=us
```

Lets check each parameter. The "span" parameter specifies the span signaling to be used. For E1 cards, each span should be configured with a timing priority, LBO, framing, coding, and optionally, some flags. Each of these values is comma separated. So, the first line of the previous file is read: "Configure span 1 as timing source with priority 1, LBO with 1 value, CAS framing and HDB3 coding". Now, lets explain each of the span parmeters.

span=<span number>,<timing>,<LBO>,<framing>,[<coding>,[<alarm>]]

SpanNo: Number of the span to configure. That is the same file name located at /proc/zaptel

Timing: If you are connecting this span to the telco, is highly probable that the best value here is 1, since you should be using the telco timing as source.

LBO: Line Build Out, is an integer that maps to some of this values:

0: 0 db (CSU) / 0-133 feet (DSX-1)
1: 133-266 feet (DSX-1)
2: 266-399 feet (DSX-1)
3: 399-533 feet (DSX-1)
4: 533-655 feet (DSX-1)
5: -7.5db (CSU)
6: -15db (CSU)
7: -22.5db (CSU)

And it seems are very related with the length of the link. Usually a zero value is correct.

Framing: CAS, Channel Associated Signaling. You can search Wikipedia for what CAS is, but in short, is the E1 signaling where MFCR2 will be embedded in each channel.

Coding: HDB3 (High Density Bipolar-3 Zeros). Is the way in wich zeroes and ones series will be transmitted.

CRC4: The CRC4 is a special and optional parameter. Is used when the other end of the span requires CRC4 checksum. In mexico i dont know any telco using CRC4. If you dont want CRC4 but you want the next sub parameter "Alarm", then just ignore this, Alarm parameter may be 6th or 7th sub parameter.

Alarm: the alarm parameter is a special and optional parameter. I dont know when/how to use it.

The next parameter after "span" is "cas", and means that the specified range of channels (it does not matter wich span belongs) will use the CAS signaling and will have initialized ABCD bits to 1101, wich it means "Blocked" in MFCR2 supervisory signals. The line "dchan=16" is pretty much the same as the line "cas=1-10:1101", but this time says "channel 16 will be used as dchan", dchan are signaling only channels, and dont use ABCD bits, so we dont put ":1101" like in CAS signaling. Actually ABCD bits work only with CAS signaling.

Finally, the parameters defaultzone and loadzone are used for specific country telephony tones. Dont worry about this too much anyway. More information can be found in zaptel.conf.sample in zaptel drivers source directory. Zaptel cards are used for non telephony purposes too, you can even get connected to the Internet using HDLC (High Level Data Link Control).

Once configured zaptel.conf, lets execute the command "ztcfg -vv", this will cause the /etc/zaptel.conf file to be read to configure the zaptel spans and channels. Once configured, check the span in /proc/zaptel to see if the channels have the proper signaling.

ZTTOOL

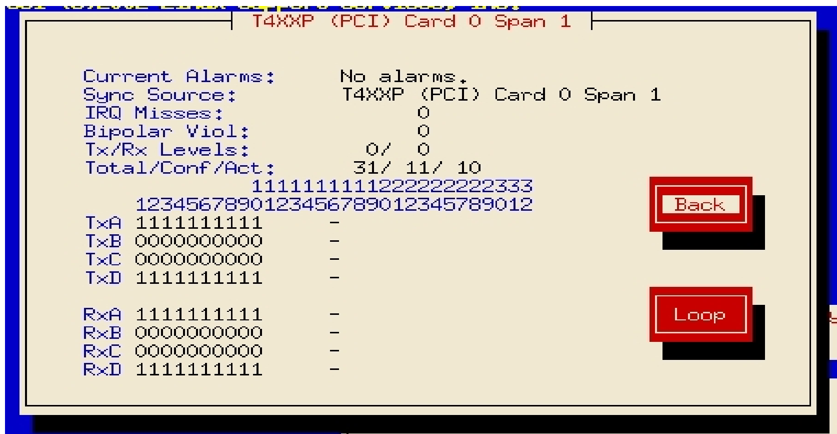
Now we are going to use "zttool" to verify some about the spans. This tool allow to monitor the status of the cards and some other usefull information.

```
# zttool
```

Executing that command should bring an ugly blue screen with a board at the center listing the zaptel spans. Like this:

You can see 2 columns, one called "Alarms" and the other "Span". That way you can check the link status of your spans. If you see any other alarm status than "OK", then something is not ok :p . Red alarm means that no link is detected at all (no cable connected to a valid end point). I cannot certainly say what Yellow alarm means, but i think is just little bit less bad than Red :)

If you hit the "Select" button, the selected span details will be shown.



Current Alarm: This shows the span alarms or "No alarms" if the span is OK

Sync Source: This will tell you what is the syncronization clock used.

IRQ Misses: If you have anything different to zero here, it would be advisable to check the IRQ of the card to make sure that the card is not sharing the same IRQ number that other hardware devices. IRQ misses means missing information, so you can get choppy audio because of it.

Bipolar Viol: Hu??? i really dont know :)

Tx/Rx Levels: Transmission and reception levels

Total/Conf/Act: This is just a guess, but it may mean total channels of the span, configured channels, and used channels.

Then you must see some rows like "TxA, TxB, TxC, TxD" and "RxA, RxB, RxC, RxD", that are nothing but the ABCD bits for CAS signaling transmitted and received. So you can actually monitor the ABCD bits for each single channel of the card. In the image above, we can see that all the bits of the first 10 channels are set to 1001, as being transmitted and received. That means that the other end (far end) of the span is sending 1001 in each channel, and that we are sending 1001 from our end (local end). When the span channels are not being used for any application the Tx bits should be the same as you configured the zaptel.conf file, in my server, when Asterisk is not running, the bits show "1101" (Blocked) just as we configured zaptel.conf for channels 1-10. In the moment Asterisk starts, and loads chan_unicall, the underlying software (unicall, libmfcr2) set the bits to 1001 (Idle) to be able to receive calls in MFCR2. The Rx bits values will depend on what do you have connected at the other end of the span. If you are connected to the telco, then the bits will show that the telco is sending you, if you dont have 1001 in Rx bits, that means the telco is not sending the proper signaling. This could be not enterely true, since usually only the AB bits are used, and the other two (CD) may be different.

unicall.conf

Asterisk configuration is very simple. Usually the default values in unicall.conf that comes with the Unicall sources are enough, just put it under /etc/asterisk or the directory where the Asterisk configuration files are in your system. The more important parameters for unicall.conf are:

protocolclass=mfcr2
protocolvariant=mx,0,4,7
channel=1-10

The parameter "protocolclass" says that we are going to use MFCR2 signaling protocol. Unicall is a separate thing to MFCR2, wich is only one of the protocols unicall can use. So, for unicall to be able to use the MFCR2 protocol, a "protocol handler" should exists in the system (libmfcr2). This protocol handler is installed by libmfcr2 as a shared object that Unicall will attempt to load when needs to interpret a protocolclass mfcr2. The shared object is usually is located in /usr/lib/unicall/protocols/protocol_mfcr2.so, this is the Unicall protocol handler to interpret MFCR2 signaling. The location would depend on how you configured libunicall and libmfcr2. I strongly recommend to use always the same path for libunicall (Telephony Abstraction), libmfcr2 (MFCR2 protocol handler), libsupertone (Supervisory Tones Library) and spandsp (General DSP routines) packages. This path prefix is specified when you install the packages using the command "./configure --prefix=/path/to/install/dir". So be sure you use the same for all these packages.

The next parameter is "protocolvariant", and have several options separated by comma. This parameter works this way:

protocolvariant=<country code>,<expected ANI digits>,<expected DNIS digits>,<options mask>

Lets see the parameter options we used in the configuration.

mx: 2 letter country code protocol variant (in this case, México)

0: This means 0 ANI digits to be received. You need to know how many digits to expect from the other end. If you put more digits than the number of digits the other end is going to send, it wont work, because it will time out waiting for ANI digits. If you are in doubt, put 0 here, but be aware that you wont get callerid.

4: How many DNIS digits to receive

7: Options mask, explained below...

The option mask works this way:

Decimal	Binary	Option
1	00000001	Generate dial tone. This is usually not needed since the telco provides the tone.
2	00000010	Generate Disconnected tone. The

		same as the first option.
4	00000100	Ringback tone
8	00001000	Enable this option to alter the MFCR2 protocol behaviour to ask for the ANI digits before getting the DNIS digits. The normal behaviour is get first the DNIS digits.
16	00010000	Skip B and Group II signals accepting immediatly the calls.

Any combination of options can be used by doing logical OR operation with the binary options (Or a simple sum between the decimal values) and putting the result as decimal number in the last sub parameter of the protocolvariant parameter. In the example, 7 means use Ringback tone, disconnected tone and dial tone.

Once configured, you can start Asterisk. While Asterisk is being started, look for some messages like these:

```
18:33:45 WARNING[849]: chan_unicall.c:2644 handle_uc_event: Unicall/4 event Far end unblocked
Unicall/4 far unblocked
18:33:45 WARNING[849]: chan_unicall.c:2644 handle_uc_event: Unicall/4 event Local end unblocked
Unicall/4 local unblocked
18:33:45 WARNING[849]: chan_unicall.c:2644 handle_uc_event: Unicall/5 event Far end unblocked
Unicall/5 far unblocked
18:33:45 WARNING[849]: chan_unicall.c:2644 handle_uc_event: Unicall/5 event Local end unblocked
Unicall/5 local unblocked
18:33:45 WARNING[849]: chan_unicall.c:2644 handle_uc_event: Unicall/6 event Far end unblocked
Unicall/6 far unblocked
18:33:45 WARNING[849]: chan_unicall.c:2644 handle_uc_event: Unicall/6 event Local end unblocked
Unicall/6 local unblocked
18:33:45 WARNING[849]: chan_unicall.c:2644 handle_uc_event: Unicall/7 event Far end unblocked
Unicall/7 far unblocked
18:33:45 WARNING[849]: chan_unicall.c:2644 handle_uc_event: Unicall/7 event Local end unblocked
Unicall/7 local unblocked
18:33:45 WARNING[849]: chan_unicall.c:2644 handle_uc_event: Unicall/8 event Far end unblocked
Unicall/8 far unblocked
18:33:45 WARNING[849]: chan_unicall.c:2644 handle_uc_event: Unicall/8 event Local end unblocked
Unicall/8 local unblocked
18:33:45 WARNING[849]: chan_unicall.c:2644 handle_uc_event: Unicall/9 event Far end unblocked
Unicall/9 far unblocked
18:33:45 WARNING[849]: chan_unicall.c:2644 handle_uc_event: Unicall/9 event Local end unblocked
Unicall/9 local unblocked
18:33:45 WARNING[849]: chan_unicall.c:2644 handle_uc_event: Unicall/10 event Far end unblocked
Unicall/10 far unblocked
18:33:45 WARNING[849]: chan_unicall.c:2644 handle_uc_event: Unicall/10 event Local end unblocked
Unicall/10 local unblocked
```

Dont pay attention to the fact that says "Warnig", since Steve Underwood used the Warning level for any message in chan_unicall. The "Notice" log level would be more appropiate. I think this issue is fixed in newer versions of chan_unicall. The important thing here is what each of these messages say. Lets take 1 of the messages as example: "Unicall/9 event Far end unblocked" and just after that we see "Unicall/9 event Local end unblocked". This messages are reporting the unicall event that Asterisk has unblocked the local end through setting the ABCD bits of the channel to 1001, and that the far end (my telco) also is detected as "unblocked" because my telco has their ABCD bits set to 1001. If the remote end had the ABCD bits in Blocked (1101), we would have seen a slightly different message saying that the far end is blocked.

Now that Asterisk is started, go ahead and execute the command "UC show channels". If the command is not available, then you dont have ready the chan_unicall.so module, try loading it with "load chan_unicall.so" from Asterisk CLI. If the module is not found, it could be that you installed Unicall in the wrong place. Check google.com for more info about installing Unicall.

With "UC show channels", you should see something like this:

```
ibbmexico*CLI> UC show channels
Channel Extension  Context        Status     Language   MusicOnHold
      1 5860       digital_incomin Idle      en         default
      2 5865       digital_incomin Idle      en         default
      3 5860       digital_incomin Idle      en         default
      4 5860       digital_incomin Idle      en         default
      5            digital_incomin Idle      en         default
      6            digital_incomin Idle      en         default
      7            digital_incomin Idle      en         default
      8            digital_incomin Idle      en         default
      9            digital_incomin Idle      en         default
     10            digital_incomin Idle      en         default
ibbmexico*CLI>
```

The image shows 4 columns.

"Channel" is the zaptel channel.

"Extension" is the last extension that used the channel. In this case my telco sends only 4 numbers from the 8 PSTN numbers. I have the serie from "XXXX5860" to "XXXX5890", that is 30 numbers. Why I have configured only 10 channels?. Because the telco usually sells you more numbers than the actual number you can use at the same time. So the numbers are not tied at all to the channels. If someone dials the 5860 termination, and no other call is active, my telco uses channel 1 to send me that call. While that call is active, the 5860 number is still available, but the telco will send me the call through channel 2, and so on.

"Context" is the Asterisk context used for that channel.

"Status" show the channel status. Idle is the correct status. If you see blocked here, is probable that the telco has not enabled your service.

"Language" is the language to be used for Asterisk sounds in that channel.

"MusicOnHold" the music on hold class to be used for that channel.

TestCall

Once configured zaptel.conf, we dont even need Asterisk to test that our MFCR2 signaling. As I stated earlier, Asterisk is just a user application of Unicall (uses unicall through its channel driver chan_unicall). If you are having problems for making calls through MFCR2, i think is better to try to find where the problem is, Asterisk or Unicall?. Nothing better than use "testcall" application for this. Testcall will allow us to make and accept calls without using Asterisk. Unicall does not install the testcall utility in /usr/bin, so we need to go to the source directory of unicall and see if the binary file is there. If is not there, at least we should see testcall.c, the C source, can be compiled executing "make testcall" in the Unicall source directory. Once installed, we must create a testcall configuration. Before proceeding with configuration, I must say that testcall works in 2 modes: "caller" and "callee". When working in "caller" mode, testcall will attempt to make calls through the specified channels. When workin in "callee" mode, it will wait for calls in the specified channels.

The next is a "callee" configuration:

```
caller no
protocol-class mfcr2
protocol-variant mx,2,2
on-offered answer
circuits 94-94
```

And this is a "caller" configuration:

```
caller yes
protocol-class mfcr2
protocol-variant mx,2,2
destination-no 12
originating-no 34
circuits 63-63
```

The order of the parameters does not matter. But be aware that only 1 space should exists between the parameter name and the parameter value. No blank lines are allowed, no spaces at the beggining of the line neither. Any line that does not start with "#" its supposed to have a valid parameter.

This is how the parameters work:

caller: Accepts 2 values, "yes" if you want to test outgoing calls (caller mode), or any other value, ie "no" to test incoming calls (callee mode).

protocol-class: type of protocol signaling to use, is the same as unicall.conf protocolclass. Here we use "mfcr2".
protocol-end: this does not matters for MFCR2, just ignore it.
destination-no: if we want to test outgoing calls, put here the number to dial through the channel spans. This parameter is only considered when the parameter caller is set to "yes".
originating-no: callerid to be used if we are testing in outgoing mode (caller yes)
on-offered: what to do when the call is received when testing in "callee mode" (caller no). Valid values are: accept,answer,busy,unassigned,congested,oos,random
circuits: wich zaptel circuits to use (zaptel channels) . For the example configuration we have seen, the correct value here would be 1-10, or any range between. If you are going to test outgoing calls, I recommend to use just 1 circuit. Otherwise testcall will attempt to generate as many calls as available circuits, and is confusing to see all the messages from all the calls at the same time. This parameter only accepts ranges, so, for testing with only one circuit you need to do something like "2-2" for test from the zaptel channel 2 to 2. Remember you cannot use the 16 channel, because is used for link signaling. So, for example, if you are testing incoming calls in all the channels of span 1, something like this must work:

circuits 1-15
circuits 17-31
(You repeat the parameter with different ranges, skipping the channel 16)

So far, so good, we now continue, assuming we are going to test in "callee mode", that is, the parameter caller will be "no" (caller no). Once we have the configuration file, we can test!. All the screenshots below are of my test system. My test system consists of the spans 3 and 4 of the same card we configured before with channels 1-10. I will only use channel 63 (span 3) and channel 94 (span 4). The configuration file must be named "testcall.conf", since the file name is hard coded in the C source of testcall. To run the test, execute the command (assuming you are in the same directory as the testcall utility).

```
# ./testcall
```

Now, testcall should have read the circuits parameter of testcall.conf and will open() /dev/zap/channel to select the channels you specified. Now it is waiting for the other end of the channels to request a call. Remember that this "request of call" is made by the other end using the ABCD bits signals (Supervisory Signals). Each second you must see a "Main Thread" message, indicating that testcall is running fine, and just waiting for calls on the specified channels. Now you can dial from a conventional telephone to the numbers you know you have assigned to the configured channels, and testcall will do the specified action in "on-offered" parameter.

While waiting for calls, testcall shows something like this:

```
Chan 94, class 'mfcr2', variant 'mx,10,4', end 542133587, caller 0, from '' to ''
Loading protocol mfcr2
Thread for channel 0
MFC/R2 Chan  94: Call control(8)
MFC/R2 Chan  94: Unblock
MFC/R2 Chan  94: 1001  ->     [1/40000000/Idle         /Idle        ]
MFC/R2 Chan  94: local_unblocking_expired
Chan  94: -- Local end unblocked! :-)
Chan  94: -- Local end unblocked! :-)
Main thread
Main thread
Main thread
Main thread
```

What tell us that testcall has now unblocked channel 94 and has configured the channel with MFCR2 signaling. Ignore the other numbers since the configuration file may vary.

Before going into more detailed and controlled test, I strongly recommend to edit the C source file testcall.c to change line like this:

"logging_level= 2 & UC_LOG_SEVERITY_MASK"

for this:

"logging_level= UC_LOG_SEVERITY_MASK"

And testcall will show all the possible debug messages. Additionally, for the next tests, you need to modify the file to accept as parameter the configuration file path, because we are going to run 2 different instances of testcall, with different configuration files.You can find the modified testcall.c file here:

http://moy.ivsol.net/unicall/testcall-modified.c

Download the file and compile it again with "make testcall" in the unicall sources directory.

MORE TESTING

If you want to find if the problem is on your side, or the telco side, and you have at least 2 spans available, then you can use the same card in "loop" to call yourself and see if it works. For that you need to make a E1 crossover cable , just connect cables 1 with 4 and 2 with 5, like this page says:

http://moy.ivsol.net/unicall/e1_crossover.html

Check for the configuration that says "E1/PRI crossover cable: RJ48C/RJ48C". The crossover cable must be connected between 2 spans of the zaptel card. If the crossover cable is well done, you should see "OK" in both spans. Now, you can use testcall (or Asterisk) to see if calls between the spans work. If you want to use testcall, I recommend to edit testcall.c again to make it able to receive the configuration file name as command line argument. Because using testcall to test the loop, you need 2 configurations, one to run in "caller mode" (caller yes) and the other span configured as "callee mode" (caller no).

Now you need to make sure that the configured circuits in span 1 match the configured circuits in span 2. That is, if you test with span 3 circuit 63-63, then in span 4 you need at least the circuit 94-94 configured. So, if you put "caller yes" in the channel 63-63, you need to put "caller no" in channels 94-94, because outgoing calls by channel 63 will be incoming calls in channel 94 (because we have a loop with the crossover cable).

Now we're ready to test in loop. In our example we use span 3 and 4, with channels 63(outgoing) and 94(incoming). Now lets remember how to configure span 3 (caller/outgoing) channel 63(outgoing):

```
caller yes
destination-no 12
originating-no 34
protocol-class mfcr2
protocol-variant mx,2,2
circuits 63-63
```

We put the "caller" parameter to yes. Then, since we are going to call, we need to specify wich destination number, and callerid. Here i have put "12" and "34" because I wanted the debug messages screenshot to fit. But you can use any number you want, just remember that the protocolvariant variable must reflect that in the ANI and DNIS digits. Save this file as "caller.conf"

Now span 4(calle/incoming), channel 94(incoming)

```
caller no
protocol-class mfcr2
```

protocol-variant mx,2,2

on-offered answer

circuits 94-94

The callee configuration only needs the extra parameter "on-offered" to determine what to do when the call is offered. In this case we decide to answer. Save this file as "imcalled.conf".

Before starting the 2 instances of testcall, check with zttool that the channels 63 and 94 are in blocked mode (1101). Now start the caller testcall instance with the command "./testcall imcalled.conf". After that check again zttool and the channel 94 must be in 1001 (Idle). Now start the caller testcall instance with the command "./testcall caller.conf" and you should see something like this:

caller instance.

```
ibbmexico libunicall-0.0.3 # ./testcall caller.conf
Chan 63, class 'mfcr2', variant 'mx,2,2', end 542133587, caller 1, from '34' to '12'
Loading protocol mfcr2
Thread for channel 0
MFC/R2 Chan  63: Call control(8)
MFC/R2 Chan  63: Unblock
MFC/R2 Chan  63: 1001  ->       [1/40000000/Idle           /Idle          ]
MFC/R2 Chan  63: far_unblocking_expired
MFC/R2 Chan  63: local_unblocking_expired
Chan  63: -- Far end unblocked! :-)
Chan  63: -- Far end unblocked! :-)
Chan  63: -- Local end unblocked! :-)
Chan  63: -- Local end unblocked! :-)
Chan  63: Initiating call
MFC/R2 Chan  63: Call control(1)
MFC/R2 Chan  63: Make call
MFC/R2 Chan  63: Making a new call with CRN 32769
MFC/R2 Chan  63: 0001  ->       [1/        1/Idle           /Idle          ]
Chan  63: -- Dialing on channel 0
Chan  63: -- Dialing on channel 0
MFC/R2 Chan  63:        <- 1101  [1/      40/Seize          /Idle          ]
MFC/R2 Chan  63: 1 on   ->       [2/      40/Group I        /Idle          ]
MFC/R2 Chan  63:        <- 1 on  [2/      40/Group I        /DNIS          ]
MFC/R2 Chan  63: 1 off  ->       [2/      40/Group I        /DNIS          ]
MFC/R2 Chan  63:        <- 1 off [2/      40/Group I        /DNIS          ]
MFC/R2 Chan  63: 2 on   ->       [2/      40/Group I        /DNIS          ]
MFC/R2 Chan  63:        <- 6 on  [2/      40/Group I        /DNIS          ]
MFC/R2 Chan  63: 2 off  ->       [2/      40/Group I        /DNIS          ]
MFC/R2 Chan  63:        <- 6 off [2/      40/Group I        /DNIS          ]
MFC/R2 Chan  63: Calling party category 0x0
MFC/R2 Chan  63: 1 on   ->       [2/      40/Group I        /DNIS          ]
MFC/R2 Chan  63:        <- 1 on  [2/      40/Group III      /Category      ]
MFC/R2 Chan  63: 1 off  ->       [2/      40/Group III      /Category      ]
MFC/R2 Chan  63:        <- 1 off [2/      40/Group III      /Category      ]
MFC/R2 Chan  63: 3 on   ->       [2/      40/Group III      /Category      ]
MFC/R2 Chan  63:        <- 1 on  [2/      40/Group III      /ANI           ]
MFC/R2 Chan  63: 3 off  ->       [2/      40/Group III      /ANI           ]
MFC/R2 Chan  63:        <- 1 off [2/      40/Group III      /ANI           ]
MFC/R2 Chan  63: 4 on   ->       [2/      40/Group III      /ANI           ]
MFC/R2 Chan  63:        <- 3 on  [2/      40/Group III      /ANI           ]
MFC/R2 Chan  63: 4 off  ->       [2/      40/Group III      /ANI           ]
MFC/R2 Chan  63:        <- 3 off [2/      40/Group III      /ANI           ]
MFC/R2 Chan  63: 1 on   ->       [2/      40/Group III      /ANI           ]
MFC/R2 Chan  63:        <- 1 on  [2/      40/Group II       /Category      ]
MFC/R2 Chan  63: 1 off  ->       [2/      40/Group II       /Category      ]
MFC/R2 Chan  63:        <- 1 off [2/      40/Group II       /Category      ]
Chan  63: -- Alerting on channel 0
Chan  63: -- Alerting on channel 0
MFC/R2 Chan  63:        <- 0101  [1/     200/Await answer   /Category      ]
Chan  63: -- Connected on channel 0
Chan  63: -- Connected on channel 0
Chan  63: -- '*00000001*34*12*#'
```

There you can appreciate all the MFCR2 signaling!

Now the screenshot of the callee testcall instance.

```
Chan  94: -- Local end unblocked! :-)
MFC/R2 Chan  94:      <- 1001  [1/40000000/Idle          /Idle         ]
MFC/R2 Chan  94:      <- 0001  [1/40000000/Idle          /Idle         ]
MFC/R2 Chan  94: far_unblocking_expired
MFC/R2 Chan  94: Detected
MFC/R2 Chan  94: Making a new call with CRN 32769
MFC/R2 Chan  94: 1101  ->      [2/       2/Idle          /Idle         ]
Chan  94: -- Far end unblocked! :-)
Chan  94: -- Far end unblocked! :-)
Chan  94: -- Detected on channel 0, CRN 32769
Chan  94: -- Detected on channel 0, CRN 32769
MFC/R2 Chan  94:      <- 1 on  [2/       2/Seize ack     /Seize ack    ]
MFC/R2 Chan  94: 1 on  ->      [2/       2/Seize ack     /Seize ack    ]
MFC/R2 Chan  94:      <- 1 off [2/       2/Group A       /DNIS request ]
MFC/R2 Chan  94: 1 off ->      [2/       2/Group A       /DNIS request ]
MFC/R2 Chan  94:      <- 2 on  [2/       2/Group A       /DNIS request ]
MFC/R2 Chan  94: 6 on  ->      [2/       2/Group A       /DNIS request ]
MFC/R2 Chan  94:      <- 2 off [2/       2/Group C       /Category req ]
MFC/R2 Chan  94: 6 off ->      [2/       2/Group C       /Category req ]
MFC/R2 Chan  94:      <- 1 on  [2/       2/Group C       /Category req ]
MFC/R2 Chan  94: 1 on  ->      [2/       2/Group C       /Category req ]
MFC/R2 Chan  94:      <- 1 off [2/       2/Group C       /ANI request  ]
MFC/R2 Chan  94: 1 off ->      [2/       2/Group C       /ANI request  ]
MFC/R2 Chan  94:      <- 3 on  [2/       2/Group C       /ANI request  ]
MFC/R2 Chan  94: 1 on  ->      [2/       2/Group C       /ANI request  ]
MFC/R2 Chan  94:      <- 3 off [2/       2/Group C       /ANI request  ]
MFC/R2 Chan  94: 1 off ->      [2/       2/Group C       /ANI request  ]
MFC/R2 Chan  94:      <- 4 on  [2/       2/Group C       /ANI request  ]
MFC/R2 Chan  94: 3 on  ->      [2/       2/Group C       /ANI request  ]
MFC/R2 Chan  94:      <- 4 off [2/       2/Group B       /Go to grp II ]
MFC/R2 Chan  94: 3 off ->      [2/       2/Group B       /Go to grp II ]
MFC/R2 Chan  94:      <- 1 on  [2/       2/Group B       /Go to grp II ]
Chan  94: -- Offered on channel 0, CRN 32769 (ANI: 34, DNIS: 12)
Chan  94: -- Offered on channel 0, CRN 32769 (ANI: 34, DNIS: 12)
Chan  94: -- +++ ANSWER (1155669150)
MFC/R2 Chan  94: Call control(5)
MFC/R2 Chan  94: Answer call
MFC/R2 Chan  94: 1 on  ->      [2/       4/Group B       /Go to grp II ]
MFC/R2 Chan  94:      <- 1 off [2/       4/Group B       /Accepted Paid]
MFC/R2 Chan  94: 1 off ->      [2/       4/Group B       /Accepted Paid]
MFC/R2 Chan  94: Answer guard expired
Chan  94: -- Accepted on channel 0
Chan  94: -- Accepted on channel 0
MFC/R2 Chan  94: Call control(5)
MFC/R2 Chan  94: Answer call
MFC/R2 Chan  94: 0101  ->      [1/      20/Group B       /Accepted Paid]
Chan  94: -- Answered on channel 0
Chan  94: -- Answered on channel 0
Chan  94: -- '*00000001***#'
MFC/R2 Chan  94:      <- 1101  [1/     400/Answer        /Accepted Paid]
MFC/R2 Chan  94: R2 prot. err. [1/     400/Answer        /Accepted Paid] cause 32773 - Unexpected CAS bit pattern
MFC/R2 Chan  94: 1001  ->      [1/       1/Idle          /Idle         ]
Chan  94: -- Protocol failure on channel 0, cause (32773) Unexpected CAS bit pattern
Chan  94: -- Protocol failure on channel 0, cause (32773) Unexpected CAS bit pattern
Main thread
```

At the end of this screenshot we can see a message saying "Protocol failure on channel 0, cause (32773) Unexpected CAS bit pattern", that is because after running the test, I killed the caller testcall instance with "CTRL +

C" and then suddenly appeared unexpected ABCD bits pattern in the callee side.

Thats all for now. I will appreciate any contribution, error reporting etc to this document. Thanks

Moises Silva <moy@ivsol.net>

References.

http://www.soft-switch.org/

http://www.soft-switch.org/unicall/unicall/index.html

http://www.soft-switch.org/unicall/installing-mfcr2.html

http://www.voip-info.org/wiki/view/Asterisk+MFC+R2

http://moy.ivsol.net/