Chapter 7

Developers

7.1 Build Information

7.1.1 Build Info

The following is obtained by executing "fldigi --version"

```
Build information:
built :

Fri Dec 25 15:57:13 CST 2009 by dave@linux-dev on i686-pc-linux-gnu for
i686-pc-linux-gnu

 configure flags: '--without-asciidoc' '--enable-optimizations=native'

 compiler : gcc version 4.3.3 (Ubuntu 4.3.3-5ubuntu4)

 compiler flags : -I$(srcdir) -I$(srcdir)/include
-I$(srcdir)/irrxml -I$(srcdir)/fileselector -pthread
-I/usr/local/include -I/usr/local/include -I/usr/include/freetype2
-D_THREAD_SAFE -D_REENTRANT -D_REENTRANT -I/usr/local/include
-I/usr/include/libpng12 -I/usr/local/include -pipe -Wall -fexceptions
-O2 -ffast-math -finline-functions -fomit-frame-pointer -march=native
-mfpmath=sse -DNDEBUG

 linker flags : -L/usr/local/lib -lportaudio -lm
-lpthread -L/usr/local/lib -lfltk_images -lpng -lz -ljpeg -lfltk -lXft
-lpthread -ldl -lm -lXext -lX11 -lX11 -lsndfile -lsamplerate
-lpulse-simple -lpulse -L/usr/local/lib -lhamlib -lm -lpng12
-L/usr/local/lib -lxmlrpc_server_abyss++ -lxmlrpc_server++
-lxmlrpc_server_abyss -lxmlrpc_server -lxmlrpc_abyss -lpthread
```

```
-lxmlrpc++ -lxmlrpc -lxmlrpc_util -lxmlrpc_xmlparse -lxmlrpc_xmltok
-ldl -lrt -lpthread

 libraries : FLTK 1.3.2
 libsamplerate 0.1.4
 libsndfile 1.0.17
 PortAudio 19
 PulseAudio 0.9.14
 Hamlib 1.2.10
 XMLRPC-C 1.06.31

Runtime information:
system : Linux
linux-dev 2.6.28-17-generic #58-Ubuntu SMP Tue Dec 1 18:57:07 UTC 2009
i686

 libraries : libsamplerate-0.1.4 (c) 2002-2008 Erik de Castro Lopo
 libsndfile-1.0.17

PortAudio V19-devel (built May 25 2009 06:36:24) 1899
Pulseaudio 0.9.14

Hamlib version 1.2.10
```

Return to Top of Page
Return to Main Page

7.2 Google Maps

```
snip ------------ copy the following to ~/.fldigi/scripts/map.pl

#!/usr/bin/perl

# Author: Stelios Bounanos, M0GLD
# Date:   20080625

use warnings;
use strict;
use Getopt::Std;

our $VERSION = "0.3141";
our %opts = ( "e" => 0, "m" => 1, "z" => 4);

cmdline();
open(STDOUT, '>', "/dev/null");

my $loc = exists($opts{'l'}) ? $opts{'l'} : $ENV{'FLDIGI_LOG_LOCATOR'};
die "Invalid locator\n" unless ((defined($loc) && length($loc) =~ /[2-6]/));

my $label = exists($opts{'t'}) ? $opts{'t'} : $ENV{'FLDIGI_LOG_CALL'};
$label = $loc if (!defined($label) || $label eq "");

my ($lon, $lat) = map { sprintf("%+.6f", $_) } mtoll($loc);
if ($opts{'m'}) {
    my $url = "http://maps.google.com/maps?q=${lat},${lon}(${label})&t=p&z=$opts{'z'}";
#    $url =~ s/([(),])/sprintf("%%%02X", ord($1))/ge; # encode some chars
    exec("xdg-open", $url);
    die "Could not exec xdg-open: $!\n";
}

exit(0) unless ($opts{'e'});
my $kml = (exists($ENV{'TMPDIR'}) ? $ENV{'TMPDIR'} : "/tmp") .
    "/" . $loc . ".kml";
open(KML, '>', $kml) or die "Could not write $kml: $!\n";
print KML <<EOF
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
  <Placemark>
```

```
    <name>$label</name>
    <description>
      $label
      $loc
    </description>
    <Point>
      <coordinates>$lon,$lat,0</coordinates>
    </Point>
  </Placemark>
</kml>
EOF
    ;
close(KML);

#########################################################################

sub cmdline
{
    $Getopt::Std::STANDARD_HELP_VERSION = 1;
    my $old_warn_handler = $SIG{__WARN__};
    $SIG{__WARN__} = sub { die $_[0]; };
    getopts('t:l:mz:e', \%opts);
    $SIG{__WARN__} = $old_warn_handler;
}


# Convert a 2, 4, or 6-character Maidenhead locator string
# to decimal degrees. Return a (longitude, latitude) pair.
sub mtoll
{
    my $len = length($_[0]);
    $_[0] .= join("", ("A", "A", "0", "0", "A", "A")[$len .. 5]) if ($len < 6);
    $_[0] = uc($_[0]);
    die "Invalid locator\n" unless ($_[0] =~ /[A-R]{2}\d{2}[A-X]{2}/);

    my @digits = split(//, $_[0]);
    my ($lon, $lat) = (-180, -90);

    $lon += (ord($digits[0]) - ord('A')) * 20 +
            (ord($digits[2]) - ord('0')) * 2 +
            (ord($digits[4]) - ord('A') + 0.5) / 12;
    $lat += (ord($digits[1]) - ord('A')) * 10 +
            (ord($digits[3]) - ord('0')) +
            (ord($digits[5]) - ord('A') + 0.5) / 24;

    return ($lon, $lat);
}

sub HELP_MESSAGE
{
print <<EOF

Usage: $0 [-OPTIONS [-MORE_OPTIONS]] [--] [PROGRAM_ARG1 ...]

The following single-character options are accepted:

        -t LABEL  Use LABEL as the marker label
                  The default is \$FLDIGI_LOG_CALL

        -l LOC    Place marker at IARU locator LOC
                  The default is \$FLDIGI_LOG_LOCATOR

        -m        Show in Google Maps (default)
        -z        Zoom level (Google Maps only)

        -e        Write a Google Earth kml file in
                  \$TMPDIR/LOC.kml
EOF
;
}

snip-----------------------------------------------
```

Return to Top of Page
Return to Main Page

7.3 Installing from Source

The developers recommend that you build either fldigi from source or install from the repository associated with your distribution. The repository may not be immediately available for the most current version number. In that case you can try installing the binary distribution. Keep in mind that the version numbers of the dependent shared libraries must match those on the machine used to create the binary.

The source code for fldigi is very large and has a number of dependencies that must be satisfied before a successful compile. If you are not familiar with compiling and linking source code you should probably practice on a simpler package before treading these waters. Please refer to the following web site for information on building for Linux, Windows and OS X.

`Fldigi WIKI - build instructions`

Return to Top of Page
Return to Main Page

7.4 FLDIGI KISS Hardware Commands

Custom Frame Extensions

These commands are NON-STANDARD extension(s) to the KISS interface specifications and designed to control/configure aspects of FLDIGI not normally found in Hardware TNC's. Undefined kiss frame types 6 and 7 are used to identify these custom frames.

Definition	Kiss frame ID
HARDWARE	6
RAW	7

External program which conform to the official KISS I/O specifications are expected to operate normally.

UDP/IP Connectionless communications are used to transfer information between FLDIGI and the host program. See Configure ARQ/KISS I/O and Command Line Switches for specific address and port number assignments.

7.4.1 Set Commands

Command <parameters(s)>	Use
BCHN:<ON|OFF>	Busy Channel On/Off

BCHNS:<0-999>	Busy Channel Wait Duration (seconds)
CSMA:<ON|OFF>	Enable/Disable CSMA. Used on a shared frequency
IBCHN:<N>	Set Inhibit busy channel duration to N seconds
IBCHN:0	Resets temporary duration to default setting.
IBCHN:S	Inhibit busy channel temporarily. 'S' (character) (default 5 seconds)
KISSRAW:<ON|OFF|ONLY>	Pass non HDLC encoded data to the transmitter.

```
ON   = HDLC and Unaltered data.
OFF  = HDLC data only.
ONLY = Unaltered data only.
```

Command	Use
KISSCRCM:<NONE|SMACK|CCITT|XOR|FCS>	Enable KISS frame checksum type
KPSATT:<value>	Set the fractional ratio gain value (1/value)
KPSQL:<ON|OFF>	OFF=Histogram Mode, ON=User set level
KPSQLS:<0-100>	Set squelch level (percent)
MODEM:<modem_id_string>	Set modem type. Example MODEM:PSK63RC32
RSIDBCAST:<ON|OFF>	Enable/disable RSID broadcast state change (default: OFF)
RSIDRX:<ON|OFF>	Enable/disable RX RSID
RSIDTX:<ON|OFF>	Enable/disable TX RSID
SQL:<ON|OFF>	SQL On/Off
SQLS:<0-100>	Set SQL Level (percent)
TRXSBCAST:<ON|OFF>	Enable/disable TX/RX state change broadcast (default: OFF)
TXBEBCAST:<ON|OFF>	Enable/disable TX buffer empty broadcast (default: OFF)
TXLOCK:<ON|OFF>	Enable/disable TX waterfall position lock.
WFF:<integer value>	Move TXRX cursor to frequency on waterfall.

```
NONE  = No CRC used
SMACK = Enable CRC use for Host <-> TNC ASYNC links
CCITT = Enable CCITT CRC
FCS   = Enable FCS CRC
XOR   = Enable BPQ's XOR CRC
```

Command <parameters(s)>	Use
RSIDM:<BANDPASS|MODEM>,<ACTIVE|NOTIFY>	Set the RSID mode to,...

```
BANDPASS = Search entire bandpass for RSID signals
MODEM    = Search Modem bandpass (+/- 200HZ) for RSID signals
ACTIVE   = When RSID received, move frequency transmit/receive cursor to location.
NOTIFY   = Report RSID when detected (Notify only).
```

7.4.2 Query Commands

Command / Return Status	Use
BCHN: FLDIGI returns BCHN:<ON|OFF>	Busy Channel State On/Off Query
BCHNS: FLDIGI returns BCHNS:<0-999>	Busy Channel Wait Duration Query (seconds)
BUSY: FLDIGI returns BUSY:<Y|N>	Return the state of the squelch, if there is a signal present
CSMA: FLDIGI returns CSMA:<ON|OFF>	CSMA State ON/OFF
FLSTAT: FLDIGI returns FLSTAT:<INIT|OK>,<HH:MM:SS>	Start up State and runtime. Returns OK and after initial Query

IBCHN: FLDIGI returns IBCHN:<SECONDS>	Inhibit busy channel durations in seconds.
KISSCRCM: FLDIGI returns CRCMODE:<NONE|SMACK>,<CCITT|XOR|FCS>	Host <-> TNC KISS CRC Types
KPSATT: FLDIGI returns KPSATT:<value>	Fractional value of KPSQL ratio gain figure (1/value)
KPSQL: FLDIGI returns KPSQL:<ON|OFF>	OFF=Histogram Mode, ON=User set level
KPSQLP: FLDIGI returns KPSQLP <0-100>	Current Power Level
KPSQLS: FLDIGI returns KPSQLS:<0-100>	Squelch set level Query (percent)
MODEM: FLDIGI returns MODEM:<Modem ID String>	Current Modem
MODEMBW: FLDIGI returns MODEMBW:<Bandwidth in Hz>	Current Modem Bandwidth
MODEML: FLDIGI returns MODEML:Modem1,Modem2,...	A List of comma delimited modem ID strings.
RSIDM: FLDIGI returns RSIDM:<BANDPASS|MODEM>,<ACTIVE|NOTIFY>	Return current RSID Mode
RSIDRX: FLDIGI returns RSIDRX:<ON|OFF>	RX RSID ON?
RSIDTX: FLDIGI returns RSIDTX:<ON|OFF>	TX RSID ON?
SQL: FLDIGI returns SQL:<ON|OFF>	SQL On/Off
SQLP: FLDIGI returns SQLP:<0-100>	Current Symbol Quality Level (Currently Not returning valid data)
SQLS: FLDIGI returns SQLS:<0-100>	Set SQL Level Query (percent)
TNC: FLDIGI returns TNC:FLDIGI <Version Number>	Returns the version number of FLDIGI
TRXS: FLDIGI returns TRXS:<RX|TX>	Return the current RX/TX state of FLDIGI
TXBUF: FLDIGI returns TXBUF:<count>	Return the number of byte in the transmit queue
TXLOCK: FLDIGI returns TXLOCK:<ON|OFF>	Return TX waterfall position lock state.
WFBW: FLDIGI returns WFBW:<LOWER HZ>,<UPPER HZ>	Return the active waterfall bandwidth
WFF: FLDIGI returns WFF:<integer value>	Current modem position on waterfall (center)

7.4.2.1 Query Commands (FLDIGI to HOST)

Command	Use
HOST:	Host returns HOST:<program_name> <version>

Note:

Not required

7.4.3 Broadcast Status Messages (FLDIGI to HOST)

Status <parameters(s)>	Use
RSIDN:NEW_WF_OFFSET,NEW_MODEM,OLD_WF_OFFSET,OLD_MODEM,<ACTIVE|NOTIFY|USER>	RSID NOTICE
TRXS:<RX|TX>	Transmitted during a state change between RX/TX or TX/RX.
TXBE:	Broadcast on emptied transmit buffer.

```
NEW_WF_OFFSET = 0-4000
NEW_MODEM     = Modem ID string of newly switched modem.
OLD_WF_OFFSET = 0-4000
OLD_MODEM     = Modem ID string of modem prior to switching.
ACTIVE        = Indicating TXRX cursor changed.
NOTIFY        = Notice only.
USER          = Report User or commanded via KISS interface modem change.
```

7.4.4 Format of Hardware Commands

FLDIGI will only except KISS frames with a PORT ID of '0' (zero). The hardware frame is a kiss frame type 6.

```
MODEM:PSK125RC16
FEND,PORT(0)|HRDW(6),'M','O',D','E','M',':','P','S','K','1',2','5','R','C','1','6',FEND
KISS HARDWARE FRAME="MODEM:PSK63RC32"
KISS HARDWARE FRAME="RSIDTX:ON"
KISS HARDWARE FRAME="RSIDRX:OFF"
```

7.4.5 Format of RAW Data

To enable RAW use, issue hardware command **KISSRAW**:<ON|OFF|ONLY>. The encode of the raw data is the same as the encoding of normal kiss data. The host program is responsible for the proper protocol data syntax.

Format:

Dn=8 bit data (byte)

```
KISS RAW FRAME=D1,D2,D3,...

FEND,PORT(0)|RAW(7),D1,D2,D3,...,FEND
```

Notes:

1. The transmit buffer has a ten minute timer association. In the event data has been retained in the transmit buffer for more then 10 minutes without being transmitted in part or whole the buffer is cleared. This will prevent a build up of data and a subsequent transmit for an extended period once the frequency is clear. Consider all packets in the transmit queue lost. If enabled, TXBE will be issued when this occurs.
2. Both ARQ and KISS IO ports are active for the reception of data. However, only the selected port will pass data to the higher functions and the other port will not buffer the data for later use. Reason: see note 1.
3. By keeping a complete ARQ (RAW) packet in one KISS frame both HDLC and ARQ (RAW) data can coexist. This allows for sequential data transmission without interleaving data at the byte level.

7.5 Parse UALR

A simple parser to create a formated console output for fldigi's <EXEC> macro:

```
snip-----------------------------------------
#include <ctime>
#include <cstdio>
#include <cstdlib>
#include <unistd.h>
#include <string>
#include <iostream>
#include <fstream>

using namespace std;
using std::cout;
using std::cin;

int main(int argc, char *argv[])
{
    size_t pos = 0, pos2 = 0, pos3 = 0, pos4 = 0, pos5 = 0;
    string commandline = "";
    string name = "";
```

```
    string qth = "";
    string answer = "";
    char c = cin.get();

    while (!cin.eof()) {
        commandline += c;
        c = cin.get();
    }

    if (commandline.find("No match found") != string::npos)
        goto noresponse;

    pos = commandline.find(", ");

    if (pos == string::npos)
        goto noresponse;

    pos += 2;
    pos2 = commandline.find("\n", pos);

    if (pos2 == string::npos)
        goto noresponse;

    name = commandline.substr(pos, pos2 - pos);
    pos3 = name.find(32);

    if (pos3 != string::npos)
        name = name.substr(0, pos3);

    for (size_t i = 1; i \< name.length(); i++)
        name[i] = tolower(name[i]);

    answer = "$NAME";
    answer.append(name);

    pos4 = commandline.find(", ", pos2);
    pos4 = commandline.rfind( "\n", pos4);
    pos4 += 1;
    pos5 = commandline.find("\n", pos4);

    qth = commandline.substr(pos4, pos5 - pos4);

    answer.append("$QTH");
    answer.append(qth);

    cout \<\< answer.c_str();

    return 0;

noresponse:;

    cout \<\< "$NAME?$QTH?";

    return 0;
}
snip-----------------------------------------------
```

Save the above as "parseUALR.cxx" and then compile and link as follows:

```
g++ parseUALR.cxx -o parseUALR
```

Copy the "parseUALR" executable to a directory on your shell exec PATH.

Return to Top of Page
Return to Main Page

7.6 Pseudo FSK

Using the FLdigi Pseudo FSK (Rt. Channel) function to key a transmitter

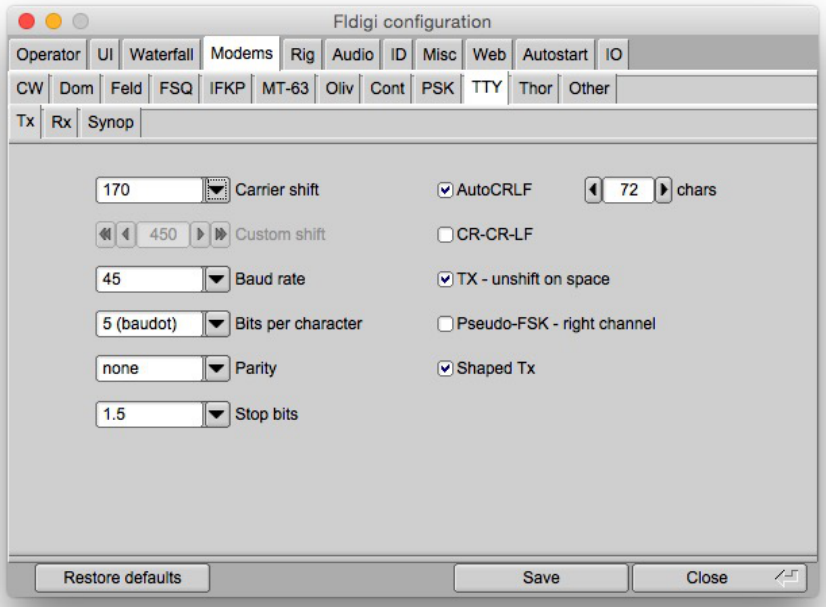

Figure 7.1: RTTY TX Configure

Select the PseudoFSK check boxes.

FLdigi is now ready to generate a 1000 hertz tone burst signal on the right channel of the stereo audio out of your sound card.

This tone burst is on when the RTTY bit is on and off when the RTTY bit is off. The left channel will be the normal AFSK signal.

The following circuit may be used to take the FLdigi PSEUDO-FSK signal from the right channel of your SOU↩ ND CARD to key your transmitter's FSK input line. You may find it necessary to invert the sense of the keying signal.

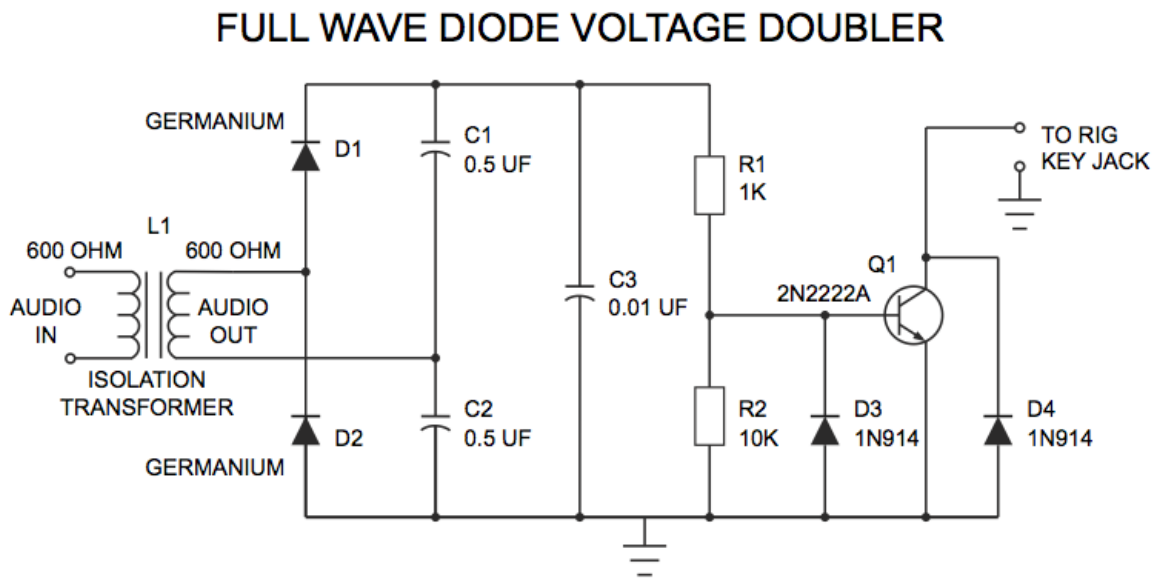

Figure 7.2: CW Keying Circuit

NOTE:

L1 - Radio Shack has two items that may be used for this isolation transformer.

`Catalog # 270-054, and Catalog # 273-1374`

Attach an audio cable from the Rt. Channel out of the your computer's SOUND CARD to the input of this FSK INTERFACE CIRCUIT (input of L1).

Attach another cable from the output of this circuit to your Rig's Keying FSK Jack.

Every PSEUDO-FSK tone that is generated by FLdigi is rectified by this FULL WAVE VOLTAGE DOUBLER circuit. The resultant voltage turns the Q1 transistor on and "grounds" the collector.

Return to Top of Page
Return to Main Page

7.7 Rig Xml How to

This document describes the contents of the rig definition file "rig.xml".

A number of transceivers have rig definition files written and tested which you may use. These are found in the xmls directory on this site: `xml archives`. You will find subdirectories by manufacturer which contain files named by rig type, ie: TS-850.xml. If you create, test and verify the proper operation for a transceiver not yet posted please share that with others by sending it as an attachment to feedback [at] w1hkj [dot] com and I will post it on the web site. You are encouraged to study the various rig definition files to learn more about how they are organized.

Comments are contained within the tag pair:

```
<!--
...
-->
```

and may appear anywhere in the rig definition file The entire rig definition must be contained within the tag pair

```
<RIGDEF>
...
</RIGDEF>
```

The text within the tag pair <RIG></RIG> specifies the transceiver to which this file applies, as in:

```
<RIG>Icom 746 PRO</RIG>
```

The text within the tag pair <PROGRAMMER></PROGRAMMER> is not used by the parser, but should as a minimum say who created and who tested the definition file, as in:

```
<PROGRAMMER>
    Dave Freese W1HKJ Tested by: W1HKJ, Dave
</PROGRAMMER>
```

The text within the tag pair

```
<STATUS> ... </STATUS>
```

is not used by the parser, but should as a minimum state whether the definition file has been "Verified", is "Alpha&", what the Version and Date of creation or update, as in:

```
<STATUS> Verified Version: 1.0 Date: 2007 Jan 5 </STATUS>
```

The title bar tag pair contains the text which will be displayed on the window decoration bar, as in:

```
<TITLE>Rig Control - IC-746 PRO</TITLE>
```

7.7.1 Serial Port Parameters

The serial port parameters may be preset in the xml file and also set or changed on the rigCAT configuration tab. These values are loaded from the xml file. If a value is changed on the configuration tab it is saved in the progdefaults.xml file if the configuration is saved. On a subsequent start of fldigi the saved parameters will override the ones in the rig definition file.

xml tag	parameter	description
<TIMEOUT>TT</TIMEOUT>	TT in milliseconds	serial port timeout
<RETRIES>NN</RETRIES>	NN integer	number of times CAT string is resent
<WRITE_DELAY>TT</WRITE↩ _DELAY>	TT in milliseconds	wait time after sending normal command to xcvr
<INIT_DELAY>IT</INIT_DEL↩ AY>	IT in milliseconds	wait time after sending init string to xcvr
<BAUDRATE>BAUD</BAUD↩ RATE>	BAUD integer	1200, 2400, 4800, 9600, 19200, 38400 ...
<STOPBITS>B</STOPBITS>	B integer	1 or 2
<RTSCTS>BOOL</RTSCTS>	BOOL true, false	h/w handshake used for data flow control
<RTSPLUS>BOOL</RTSPLU↩ S>	BOOL true, false	set RTS signal line to +12 V (default -12 V)

<RTSPTT>BOOL</RTSPTT>	BOOL true, false	toggle RTS signal line for PTT
<DTRPLUS>BOOL</DTRPLUS>	BOOL true, false	set DTR signal line to + 12 V (default -12 V)
<DTRPTT>BOOL</DTRPTT>	BOOL true, false	toggle DTR signal line for PTT
<CMDPTT>BOOL</CMDPTT>	BOOL true, false	use command string for PTT (not supported by all xcvrs)
<ECHO>BOOL</ECHO>	BOOL true, false	xcvr/interface echoes all chars (typical of CI-V interface)
<WAIT_FOR_DEVICE>nnn</WAIT_FOR_DEVICE>	nnnn in milliseconds	used for correct startup of K9MJ CI-V router
<VSP>BOOL</VSP>	BOOL true/false	serial port is a virtual device

7.7.2 Poll Interval

The rigCAT code operates in a separate thread during which the various queries are made with a default interval of 100 milliseconds between the last query and the start of the next cycle. The actual cycle period is dependent on the serial communications baud rate, the size of the commands, and the responsiveness of the transceiver. It will always be greater than the poll interval.

You can specify a poll interval in milliseconds in the xml file:

```
<POLLINT>mmm</POLLINT>
```

where mmm is the poll interval in milliseconds.

7.7.3 Modes

The transceiver modes are specified within the <MODES></MODES> tag pair. Each entry or element associated with a mode has a symbol name (text) and a way to specify what the data transfer consists of. The data transfer might be a single byte, multiple bytes, or a string

Example 1, for the Icom-746PRO

```
<MODES>
  <ELEMENT><SYMBOL>LSB</SYMBOL><BYTE>00</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>USB</SYMBOL><BYTE>01</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>AM</SYMBOL><BYTE>02</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>CW</SYMBOL><BYTE>03</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>RTTY</SYMBOL><BYTE>04</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>FM</SYMBOL><BYTE>05</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>CW-R</SYMBOL><BYTE>07</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>RTTY-R</SYMBOL><BYTE>08</BYTE></ELEMENT>
</MODES>
```

Example 2, for the Kenwood 850

```
<MODES\>
  <ELEMENT><SYMBOL>LSB</SYMBOL><BYTE>31</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>USB</SYMBOL><BYTE>32</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>CW</SYMBOL><BYTE>33</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>FM</SYMBOL><BYTE>34</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>AM</SYMBOL><BYTE>35</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>FSK</SYMBOL><BYTE>36</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>CW-R</SYMBOL><BYTE>37</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>FSK-R</SYMBOL><BYTE>39</BYTE></ELEMENT>
</MODES>
```

Example 3, for the FT-100

```
<MODES>
  <ELEMENT><SYMBOL>LSB</SYMBOL><BYTE>00</BYTE></ELEMENT>
```

```
  <ELEMENT><SYMBOL>USB</SYMBOL><BYTE>01</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>CW</SYMBOL><BYTE>02</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>CW-R</SYMBOL><BYTE>03</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>AM</SYMBOL><BYTE>04</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>DIG</SYMBOL><BYTE>05</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>FM</SYMBOL><BYTE>06</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>W-FM</SYMBOL><BYTE>07</BYTE></ELEMENT>
</MODES>
```

The modes which are supported by lower sideband in the transceiver are specified in the <LSBMODES></L↩ SBMODES> tag pair. The string data for the LSB modes must match those given in the modes id specifier For example in the Icom 746 Pro:

```
<LSBMODES>
  <STRING>LSB<STRING>
  <STRING>RTTY<STRING>
  <STRING>CW-R<STRING>
</LSBMODES>
```

7.7.4 Bandwidth Specifier

If the transceiver data stream uses identically the same format for the bandwidth data then it is specified in the <BANDWIDTHS></BANDWIDTHS> tag pair

Example for the Icom 746 Pro:

```
<BANDWIDTHS>
  <ELEMENT><SYMBOL>50</SYMBOL><BYTE>00</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>100</SYMBOL><BYTE>01</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>150</SYMBOL><BYTE>02</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>200</SYMBOL><BYTE>03</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>250</SYMBOL><BYTE>04</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>300</SYMBOL><BYTE>05</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>350</SYMBOL><BYTE>06</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>400</SYMBOL><BYTE>07</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>450</SYMBOL><BYTE>08</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>500</SYMBOL><BYTE>09</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>600</SYMBOL><BYTE>10</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>700</SYMBOL><BYTE>11</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>800</SYMBOL><BYTE>12</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>900</SYMBOL><BYTE>13</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>1000</SYMBOL><BYTE>14</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>1100</SYMBOL><BYTE>15</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>1200</SYMBOL><BYTE>16</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>1300</SYMBOL><BYTE>17</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>1400</SYMBOL><BYTE>18</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>1500</SYMBOL><BYTE>19</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>1600</SYMBOL><BYTE>20</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>1700</SYMBOL><BYTE>21</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>1800</SYMBOL><BYTE>22</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>1900</SYMBOL><BYTE>23</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>2000</SYMBOL><BYTE>24</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>2100</SYMBOL><BYTE>25</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>2200</SYMBOL><BYTE>26</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>2300</SYMBOL><BYTE>27</BYTE></ELEMENT>
  <ELEMENT><SYMBOL2400</SYMBOL><BYTE>28</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>2500</SYMBOL><BYTE>29</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>2600</SYMBOL><BYTE>30</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>2700</SYMBOL><BYTE>31</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>2800</SYMBOL><BYTE>32</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>2900</SYMBOL><BYTE>33</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>3000</SYMBOL><BYTE>34</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>3100</SYMBOL><BYTE>35</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>3200</SYMBOL><BYTE>36</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>3300</SYMBOL><BYTE>37</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>3400</SYMBOL><BYTE>38</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>3500</SYMBOL><BYTE>39</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>3600</SYMBOL><BYTE>40</BYTE></ELEMENT>
</BANDWIDTHS>
```

If the bandwidth data stream is unique for send and receive data streams then they are specified separately with a <BW-CMD></BW-CMD> tag pair for data sent to the transceiver, and a <BW-REPLY></BW-REPLY> tag pair for data returned to the computer. The number and symbol name for these must match.

Example: FT-100:

```
<BW-CMD>
  <ELEMENT><SYMBOL>300</SYMBOL><BYTE>00</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>500</SYMBOL><BYTE>01</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>2400</SYMBOL><BYTE>02</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>6000</SYMBOL><BYTE>03</BYTE></ELEMENT>
</BW-CMD>

<BW-REPLY>
  <ELEMENT><SYMBOL>300</SYMBOL><BYTE>03</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>500</SYMBOL><BYTE>02</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>2400</SYMBOL><BYTE>01</BYTE></ELEMENT>
  <ELEMENT><SYMBOL>6000</SYMBOL><BYTE>00</BYTE></ELEMENT>
</BW-REPLY>
```

7.7.5 Commands and Replies

CAT command strings are defined within a <COMMAND>...</COMMAND> block. Each block structure appears as:

```
<COMMAND>
  <SYMBOL>...</SYMBOL>   noun name of symbol
  <SIZE>...</SIZE>       number of bytes in CAT string
  <STRING>...</STRING>   string literal (optional)
  <DATA>                 data block (optional)
    <DTYPE>...</DTYPE>   type of data in data block
    <SIZE>...</SIZE>     number of bytes in data block
    <MAX>...</MAX>       maximum value of data value
    <MIN>...</MIN>       minimum value of data value
    <RESOL>...</RESOL>   resolution of data value
  </DATA>                end of data block
  <STRING>...</STRING>   string literal (optional)
</COMMAND>
```

Here are three examples from the KX3.xml definition file

```
  <COMMAND>
    <SYMBOL>INIT</SYMBOL>
    <SIZE>12</SIZE>
    <STRING>AI0;DT0;K31;</STRING>
  </COMMAND>

  <COMMAND>
    <SYMBOL>SETFREQ</SYMBOL>
    <SIZE>14</SIZE>
    <STRING>FA</STRING>
    <DATA>
      <DTYPE>DECIMAL</DTYPE>
      <SIZE>11</SIZE>
      <MAX>99999999999</MAX>
      <MIN>00000490000</MIN>
      <RESOL>1</RESOL>
    </DATA>
    <STRING>;</STRING>
  </COMMAND>

  <COMMAND>
    <SYMBOL>GETFREQ</SYMBOL>
    <SIZE>3</SIZE>
    <STRING>FA;</STRING>
    <INFO>FREQ</INFO>
  </COMMAND>
```

A command string may have a corresponding reply string sent by the transceiver. The SYMBOL noun name for each command - reply pair is the same.

Fldigi can parse and decode message returned from the transceiver that define 4 aspects of the transceiver operation:

OK	data accepted by the transceiver
BAD	data rejected by the transceiver
MODE	current operating mode of the transceiver
BW	current bandwidth setting of the transceiver
FREQ	frequency of the active VFO (might be either A or B for example)

These are all contained within multiple <REPLY></REPLY> tag pairs. The REPLY block structure is:

```
<REPLY\>
  <SYMBOL>...</SYMBOL>          noun name of symbol
  <SIZE>...</SIZE>              number of bytes in reply string
  <BYTES>...</BYTES>            space separated hexadecimal values
  <BYTE>...</BYTE>              single hexadecimal value
  <STRING>...</STRING>          returned character string
</REPLY>
```

This is an example of a fixed format message with no variable fields. It is the OK message sent back by the Icom-746 PRO:

```
<REPLY>
  <SYMBOL>OK</SYMBOL>
  <SIZE>6</SIZE>
  <BYTES>FE FE E0 66</BYTES>
  <BYTE>FB</BYTE>
  <BYTE>FD</BYTE>
</REPLY>
```

The <SYMBOL></SYMBOL> pair and the command definition are mandatory. The <SIZE></SIZE> field is mandatory and specifies the number of bytes contained in this reply. The above definition could also have been coded as:

```
<REPLY>
  <SYMBOL>OK</SYMBOL>
  <SIZE>6</SIZE>
  <BYTES>FE FE E0 66 FB FD</BYTES>
</REPLY>
```

When the reply contains variable data it is specified in a contained tag pair <DATA></DATA>. This data field contains specifiers that describe the kind and size of the data. The <DTYPE></DTYPE> tag pair may be one of:

BINARY or
DECIMAL

This is an example for the reply to a mode query that is returned by the Icom-746 PRO:

```
<REPLY>
  <SYMBOL>MODE</SYMBOL>
  <SIZE>8</SIZE>
  <BYTES>FE FE E0 66</BYTES>
  <BYTE>04</BYTE>
  <DATA>
    <DTYPE>BINARY</DTYPE>
    <SIZE>1</SIZE>
   </DATA>
 <FILL>1</FILL>
 <BYTE>FD</BYTE>
</REPLY>
```

Fldigi rigCAT will check for both the preample and postamble to insure that a valid reply has been sent by the transceiver.

This is an example for the reply to a frequency query that is returned by the Elecraft KX3. The corresponding query command structure is shown above.

```
<REPLY>
  <SYMBOL>FREQ</SYMBOL>
  <SIZE>14</SIZE>
  <STRING>FA</STRING>
  <DATA>
    <DTYPE>DECIMAL</DTYPE>
    <SIZE>11\</SIZE>
    <MAX>99999999999</MAX>
    <MIN>00001500000</MIN>
    <RESOL>1</RESOL>
  </DATA>
  <STRING>;</STRING>
</REPLY>
```

7.7.6 Meters

fldigi rigCAT can query the transceiver for both Smeter and Power Meter. The fldigi user interface controls that display these values have a range of 0 to 100. You need to provide the scale conversion from queried value to fldigi UI control scale. The <SMETER>...</SMETER> and <PMETER>...</PMETER> are reserved pairs that are used to define the conversions. The program will interpolate between values as it converts the queried value to the user interface control range.

Here are two examples of scale specifiers:

```
<!-- smeter scale mapping for FT-950 -->
<SMETER> 0,0; 255,100 </SMETER>

<!-- power meter scale mapping for FT-950 -->
<PMETER>
  0,0; 16,1; 32,4; 48,7;
  64,12; 80,18; 96,24; 112,32;
  128,40; 144,50; 160,61; 176,73;
  192,85; 208,100
</PMETER>

<!-- smeter scale mapping for IC-706MkIIG -->
<SMETER> 0,0; 255,100 </SMETER>
```

The query and respective responses are coded as any other command/reply pair:

For the FT-950:

```
<REPLY>
  <SYMBOL>SMETER</SYMBOL>
  <SIZE>7</SIZE>
  <STRING>RM1</STRING>
  <DATA>
    <DTYPE>DECIMAL</DTYPE>
    <SIZE>3</SIZE>
  </DATA>
  <STRING>;</STRING>
</REPLY>

<COMMAND>
  <SYMBOL>GET_SMETER</SYMBOL>
  <SIZE>4</SIZE>
  <STRING>RM1;</STRING>
  <INFO>SMETER</INFO>
</COMMAND>
```

```
<REPLY>
  <SYMBOL>PWRMETER</SYMBOL>
  <SIZE>7</SIZE>
  <STRING>RM5</STRING>
  <DATA>
    <DTYPE>DECIMAL</DTYPE>
    <SIZE>3</SIZE>
  </DATA>
  <STRING>;</STRING>
</REPLY>

<COMMAND>
  <SYMBOL>GET_PWRMETER</SYMBOL>
  <SIZE>4</SIZE>
  <STRING>RM5;</STRING>
  <INFO>PWRMETER</INFO>
</COMMAND>
```

and for the IC-706MkIIG:

```
<REPLY>
    <SYMBOL>SMETER</SYMBOL>
    <SIZE>10</SIZE>
    <BYTES>FE FE E0 58 15 02</BYTES>
    <DATA>
        <DTYPE>BCD</DTYPE>
        <SIZE> 3 </SIZE>
        <MAX> 255 </MAX>
        <MIN> 0 </MIN>
        <RESOL> 1 </RESOL>
     </DATA>
    <BYTE>FD</BYTE>
</REPLY>

<COMMAND>
    <SYMBOL>GET SMETER</SYMBOL>
    <SIZE>7</SIZE>
    <BYTES>FE FE 58 E0 15 02 FD</BYTES>
    <INFO>SMETER</INFO>
</COMMAND>
```

7.7.7 Notch Control

The transceiver manual notch can be both read and controlled using rigCAT.

There are a few requirements that may not be met by all transceivers.

- it must support a CAT manual notch on/off string
- it must support a CAT manual notch value string
- the conversion between notch audio frequency and the CAT value must be bilateral.

Here is an example for the FT-950:

```
<NOTCH>
  1,10; 300,3000;
</NOTCH>

<COMMAND>
    <SYMBOL>SET_NOTCH_ON</SYMBOL>
    <SIZE>8</SIZE>
    <STRING>BP00001;</STRING>
</COMMAND>
```

```
<COMMAND>
    <SYMBOL>SET_NOTCH_OFF</SYMBOL>
    <SIZE>8</SIZE>
    <STRING>BP00000;</STRING>
</COMMAND>

<COMMAND>
    <SYMBOL>SET_NOTCH_VAL</SYMBOL>
    <SIZE>8</SIZE>
    <STRING>BP01</STRING>
    <DATA>
        <DTYPE>DECIMAL</DTYPE>
        <SIZE>3</SIZE>
    </DATA>
    <STRING>;</STRING>
</COMMAND>

<REPLY>
  <SYMBOL>NOTCH_ON</SYMBOL>
  <SIZE>8</SIZE>
  <STRING>BP00001;</STRING>
</REPLY>

<COMMAND>
  <SYMBOL>GET_NOTCH_ON</SYMBOL>
  <SIZE>5</SIZE>
  <STRING>BP00;</STRING>
  <INFO>NOTCH_ON</INFO>
</COMMAND>

<REPLY>
  <SYMBOL>NOTCH</SYMBOL>
  <SIZE>8</SIZE>
  <STRING>BP01</STRING>
  <DATA>
    <DTYPE>DECIMAL</DTYPE>
    <SIZE>3</SIZE>
  </DATA>
  <STRING>;</STRING>
</REPLY>

<COMMAND>
  <SYMBOL>GET_NOTCH</SYMBOL>
  <SIZE>5</SIZE>
  <STRING>BP01;</STRING>
  <INFO>NOTCH</INFO>
</COMMAND>
```

The notch is controlled from fldigi using the alt-ctl-left-click on the waterfall. The same keyboard-mouse combination is used to both set and clear the notch. Point the cursor to an offending signal, and then use the keyboard-mouse combination to set the manual notch at that frequency. Repeat the keyboard-mouse combination anywhere in the waterfall to clear the notch. The SET notch is indicated by a dashed vertical line on the waterfall display at the audio frequency being notched. You should also see a pronounced reduction in signal at that point. fldigi will also annunciate any transceiver changes made to the manual notch.

7.7.8 Power Level Control

rigCAT can control the power level of the transceiver if that is a supported CAT command. The definitions for the power level are similar to the notch in that the conversion should be bilateral.

Another example using the FT-950 transceiver:

```
<PWRLEVEL>
  0,0; 100,100;
</PWRLEVEL>

<REPLY>
  <SYMBOL>PWRLEVEL</SYMBOL>
  <SIZE>6</SIZE>
```

```
  <STRING>PC</STRING>
  <DATA>
    <DTYPE>DECIMAL</DTYPE>
    <SIZE>3</SIZE>
  </DATA>
  <STRING>;</STRING>
</REPLY>

<COMMAND>
  <SYMBOL>GET_PWRLEVEL</SYMBOL>
  <SIZE>3</SIZE>
  <STRING>PC;</STRING>
  <INFO>PWRLEVEL</INFO>
</COMMAND>

<COMMAND>
  <SYMBOL>SET_PWRLEVEL</SYMBOL>
    <SIZE>6</SIZE>
    <STRING>PC</STRING>
    <DATA>
        <DTYPE>DECIMAL</DTYPE>
        <SIZE>3</SIZE>
    </DATA>
    <STRING>;</STRING>
</COMMAND>
```

The rigCAT controls for

- Transceiver-Mode
- Transceiver-Bandwidth
- Smeter
- Power-Meter and
- Power-Level

all share a common space on the main fldigi display.

Figure 7.3: Mode/Bandwidth controls

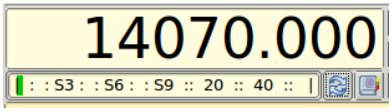

Figure 7.4: S-meter

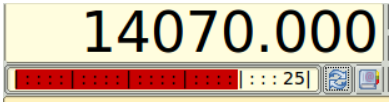

Figure 7.5: Power-meter

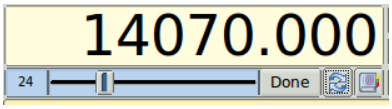

Figure 7.6: Power Level

The transition between Mode/Bandwidth and the other control/displays is made using the button just to the right of the bandwidth control.

The transition from either S-meter or Power-meter to the Power Level is make by left clicking on the S-meter or Power-meter.

7.7.9 Debugging

To assist in debugging an xml file you may place the following statement within the body of the <rigdef>...</rigdef> pair.

```
<DEBUG>true</DEBUG>
```

fldigi will then record critical events as they occur during the execution of the rigCAT loop. Remove the debug statement from the xml when the file has been proven and before publishing.

You can test an xml to observe the CAT send sequences by inhibiting the actual connection to the serial port.

```
<NOSERIAL>true/false</NOSERIAL> - default false
```

The serial i/o events are normally recorded as a sequence of HEX values. This behavior can be changed to record the events as a string of ASCII characters.

```
<ASCII>true/false</ASCII> - default false
```

You can use the xml remarks brackets

```
<!--
...
-->
```

to surround sections of the xml document to inhibit certain functions if they have already been proven.

You can increase the polling interval to slow things down during debugging.

Return to Top of Page
Return to Main Page

7.8 ualr telnet

```
snip ------------ copy the following to ~/.fldigi/scripts/ualr-telnet.pl

#!/usr/bin/perl

# Author: Stelios Bounanos, M0GLD
# Date: 20090103
#
```

```
# ualr-telnet is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 3 of the License, or
# (at your option) any later version.
#
# ualr-telnetl is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.
# ------------------------------------------------------------------------

use strict;
use warnings;

die "Usage: $0 CALLSIGN\n" unless (@ARGV == 1);

use Net::Telnet ();

sub error { print "\$NAME?\$QTH?\n"; exit(1); }
my $t = new Net::Telnet( Host => "callsign.ualr.edu", Port => 2000, Timeout => 10,
             errmode => \&error );
$t->open();
$t->waitfor('/LOOKUP>.*$/');
$t->print($ARGV[0]);

$_ = $t->getline();      # blank line
$_ = $t->getline();      # call
error() if (m/No match found/);

$_ = $t->getline();      # name
chomp; s/.+,\s+//; s/\s.+$//;
print "\$NAME$_";
$_ = $t->getline();      # addr
$_ = $t->getline();      # qth
chomp;
$_ =~ ",";
$_ = $`;

print "\$QTH$_\n";

$t->waitfor('/LOOKUP>.*$/');
$t->print("bye");

snip-------------------------------------------------
```

Return to Top of Page
Return to Main Page

7.9 Xmlrpc Control

XML-RPC data is transported via simple HTTP and client implementations exist for most programming languages. A Perl client that can be used as a control script is included in the source tar ball as scripts/fldigi-shell. This control method is currently used by several external programs including flrig, logger32 and Xlog.

The following command line arguments become available when XML-RPC support is compiled into fldigi, as described in the build instructions:

```
 --xmlrpc-server-address HOSTNAME
   Set the XML-RPC server address. The default is 127.0.0.1.

 --xmlrpc-server-port PORT
   Set the XML-RPC server port. The default is 7362.

 --xmlrpc-allow REGEX
```

```
   Allow only the methods whose names match REGEX

 --xmlrpc-deny REGEX
   Allow only the methods whose names don't match REGEX

 --xmlrpc-list
   List all available methods
```

The –xmlrpc-deny and –xmlrpc-allow switches can be used as a simple access control mechanism. REGEX specifies a POSIX extended regular expression. This invocation disables the methods that may cause fldigi to transmit:

```
 --xmlrpc-deny 'main\.(tx|tune|run_macro)'
```

By default all methods are allowed.

The –xmlrpc-list switch outputs the method list and exits the program. If preceded by –xmlrpc-deny or –xmlrpc-allow, it shows the list of methods as filtered by those switches.

The methods are listed below. The three columns are method name, signature (return_type:argument_types), and description. Refer to the XML-RPC specification for the meaning of the signature characters

7.9.1 XML Command Symbol Interpretation

Symbol	Interpretation
6	bytes
A	array
b	boolean
d	double
i	integer
n	nil
s	string
S	struct

7.9.2 Table of XML Commands

Method Name	Sig (ret:arg)	Description
fldigi.config_dir	s:n	Returns the name of the configuration directory
fldigi.list	A:n	Returns the list of methods
fldigi.name	s:n	Returns the program name
fldigi.name_version	s:n	Returns the program name and version
fldigi.terminate	n:i	Terminates fldigi. “i" is bitmask specifying data to save: 0=options; 1=log; 2=macros
fldigi.version	s:n	Returns the program version as a string
fldigi.version_struct	S:n	Returns the program version as a struct
io.enable_arq	n:n	Switch to ARQ I/O
io.enable_kiss	n:n	Switch to KISS I/O
io.in_use	s:n	Returns the IO port in use (ARQ/KISS).
log.clear	n:n	Clears the contents of the log fields
log.get_az	s:n	Returns the AZ field contents
log.get_band	s:n	Returns the current band name
log.get_call	s:n	Returns the Call field contents
log.get_country	s:n	Returns the Country field contents
log.get_exchange	s:n	Returns the contest exchange field contents
log.get_frequency	s:n	Returns the Frequency field contents
log.get_locator	s:n	Returns the Locator field contents
log.get_name	s:n	Returns the Name field contents

log.get_notes	s:n	Returns the Notes field contents
log.get_province	s:n	Returns the Province field contents
log.get_qth	s:n	Returns the QTH field contents
log.get_rst_in	s:n	Returns the RST(r) field contents
log.get_rst_out	s:n	Returns the RST(s) field contents
log.get_serial_number	s:n	Returns the serial number field contents
log.get_serial_number_sent	s:n	Returns the serial number (sent) field contents
log.get_state	s:n	Returns the State field contents
log.get_time_off	s:n	Returns the Time-Off field contents
log.get_time_on	s:n	Returns the Time-On field contents
log.set_call	n:s	Sets the Call field contents
log.set_exchange	n:s	Sets the contest exchange field contents
log.set_locator	n:s	Sets the Locator field contents
log.set_name	n:s	Sets the Name field contents
log.set_qth	n:s	Sets the QTH field contents
log.set_rst_in	n:s	Sets the RST(r) field contents
log.set_rst_out	n:s	Sets the RST(s) field contents
log.set_serial_number	n:s	Sets the serial number field contents
main.abort	n:n	Aborts a transmit or tune
main.get_afc	b:n	Returns the AFC state
main.get_char_rates	s:n	Returns table of char rates.
main.get_char_timing	n:6	Input: value of character. Returns transmit duration for specified character (samples:sample rate).
main.get_frequency	d:n	Returns the RF carrier frequency
main.get_lock	b:n	Returns the Transmit Lock state
main.get_max_macro_id	i:n	Returns the maximum macro ID number
main.get_reverse	b:n	Returns the Reverse Sideband state
main.get_rsid	b:n	Returns the RSID state
main.get_txid	b:n	Returns the TxRSID state
main.get_squelch	b:n	Returns the squelch state
main.get_squelch_level	d:n	Returns the squelch level
main.get_status1	s:n	Returns the contents of the first status field (typically s/n)
main.get_status2	s:n	Returns the contents of the second status field
main.get_trx_state	s:n	Returns T/R state
main.get_trx_status	s:n	Returns transmit/tune/receive status
main.get_tx_timing	n:6	Returns transmit duration for test string (samples:sample rate:secs).
main.get_wf_sideband	s:n	Returns the current waterfall sideband

main.inc_frequency	d:d	Increments the RF carrier frequency. Returns the new value
main.inc_squelch_level	d:d	Increments the squelch level. Returns the new level
main.run_macro	n:i	Runs a macro
main.rx	n:n	Receives
main.rx_only	n:n	Disables Tx.
main.rx_tx	n:n	Sets normal Rx/Tx switching.
main.set_afc	b:b	Sets the AFC state. Returns the old state
main.set_frequency	d:d	Sets the RF carrier frequency. Returns the old value
main.set_lock	b:b	Sets the Transmit Lock state. Returns the old state
main.set_reverse	b:b	Sets the Reverse Sideband state. Returns the old state
main.set_rsid	b:b	Sets the RSID state. Returns the old state
mmain.set_txid	b:b	Sets the TxRSID state. Returns the old state
main.set_squelch	b:b	Sets the squelch state. Returns the old state
main.set_squelch_level	d:d	Sets the squelch level. Returns the old level
main.set_wf_sideband	n:s	Sets the waterfall sideband to USB or LSB
main.toggle_afc	b:n	Toggles the AFC state. Returns the new state
main.toggle_lock	b:n	Toggles the Transmit Lock state. Returns the new state
main.toggle_reverse	b:n	Toggles the Reverse Sideband state. Returns the new state
main.toggle_rsid	b:n	Toggles the RSID state. Returns the new state
main.toggle_txid	b:n	Toggles the TxRSID state. Returns the new state
main.toggle_squelch	b:n	Toggles the squelch state. Returns the new state
main.tune	n:n	Tunes
main.tx	n:n	Transmits
modem.get_afc_search_range	i:n	Returns the modem AFC search range
modem.get_bandwidth	i:n	Returns the modem bandwidth
modem.get_carrier	i:n	Returns the modem carrier frequency
modem.get_id	i:n	Returns the ID of the current modem
modem.get_max_id	i:n	Returns the maximum modem ID number
modem.get_name	s:n	Returns the name of the current modem

modem.get_names	A:n	Returns all modem names
modem.get_quality	d:n	Returns the modem signal quality in the range [0:100]
modem.inc_afc_search_range	n:i	Increments the modem AFC search range. Returns the new value
modem.inc_bandwidth	n:i	Increments the modem bandwidth. Returns the new value
modem.inc_carrier	i:i	Increments the modem carrier frequency. Returns the new carrier
modem.olivia.get_bandwidth	i:n	Returns the Olivia bandwidth
modem.olivia.get_tones	i:n	Returns the Olivia tones
modem.olivia.set_bandwidth	n:i	Sets the Olivia bandwidth
modem.olivia.set_tones	n:i	Sets the Olivia tones
modem.search_down	n:n	Searches downward in frequency
modem.search_up	n:n	Searches upward in frequency
modem.set_afc_search_range	n:i	Sets the modem AFC search range. Returns the old value
modem.set_bandwidth	n:i	Sets the modem bandwidth. Returns the old value
modem.set_by_id	i:i	Sets the current modem. Returns old ID
modem.set_by_name	s:s	Sets the current modem. Returns old name
modem.set_carrier	i:i	Sets modem carrier. Returns old carrier
navtex.get_message	s:i	Returns next Navtex/SitorB message with a max delay in seconds.. Empty string if timeout.
navtex.send_message	s:s	Send a Navtex/SitorB message. Returns an empty string if OK otherwise an error message.
rig.get_bandwidth	s:n	Returns the name of the current transceiver bandwidth
rig.get_bandwidths	A:n	Returns the list of available rig bandwidths
rig.get_mode	s:n	Returns the name of the current transceiver mode
rig.get_modes	A:n	Returns the list of available rig modes
rig.get_name	s:n	Returns the rig name previously set via rig.set_name
rig.get_notch	s:n	Reports a notch filter frequency based on WF action
rig.release_control	n:n	Switches rig control to previous setting
rig.set_bandwidth	n:s	Selects a bandwidth previously added by rig.set_bandwidths
rig.set_bandwidths	n:A	Sets the list of available rig bandwidths
rig.set_frequency	d:d	Sets the RF carrier frequency. Returns the old value

rig.set_mode	n:s	Selects a mode previously added by rig.set_modes
rig.set_modes	n:A	Sets the list of available rig modes
rig.set_name	n:s	Sets the rig name for xmlrpc rig
rig.set_pwrmeter	n:i	Sets the power meter returns null.
rig.set_smeter	n:i	Sets the smeter returns null.
rig.take_control	n:n	Switches rig control to XML-RPC
rx.get_data	6:n	Returns all RX data received since last query.
rxtx.get_data	6:n	Returns all RXTX combined data since last query.
spot.get_auto	b:n	Returns the autospotter state
spot.pskrep.get_count	i:n	Returns the number of callsigns spotted in the current session
spot.set_auto	n:b	Sets the autospotter state. Returns the old state
spot.toggle_auto	n:b	Toggles the autospotter state. Returns the new state
text.add_tx	n:s	Adds a string to the TX text widget
text.add_tx_bytes	n:6	Adds a byte string to the TX text widget
text.clear_rx	n:n	Clears the RX text widget
text.clear_tx	n:n	Clears the TX text widget
text.get_rx	6:i	Returns a range of characters (start, length) from the RX text widget
text.get_rx_length	i:n	Returns the number of characters in the RX widget
tx.get_data	6:n	Returns all TX data transmitted since last query.
wefax.end_reception	s:n	End Wefax image reception
wefax.get_received_file	s:i	Waits for next received fax file, returns its name with a delay. Empty string if timeout.
wefax.send_file	s:i	Send file. returns an empty string if OK otherwise an error message.
wefax.set_adif_log	s:b	Set/reset logging to received/transmit images to ADIF log file
wefax.set_max_lines	s:i	Set maximum lines for fax image reception
wefax.set_tx_abort_flag	s:n	Cancels Wefax image transmission
wefax.skip_apt	s:n	Skip APT during Wefax reception
wefax.skip_phasing	s:n	Skip phasing during Wefax reception
wefax.state_string	s:n	Returns Wefax engine state (tx and rx) for information.

Deprecated methods:

Method Name	Sig	Resolution
log.get_sideband	s:n	use main.get_wf_sideband
main.get_rig_bandwidth	s:n	use rig.get_bandwidth
main.get_rig_bandwidths	n:A	use rig.get_bandwidths

main.get_rig_mode	s:n	use rig.get_mode
main.get_rig_modes	A:n	use rig.get_modes
main.get_sideband	s:n	use main.get_wf_sideband and/or rig.get_mode
main.rsid	n:n	use main.{get,set,toggle}_rsid
main.set_rig_bandwidth	n:s	use rig.set_bandwidth
main.set_rig_bandwidths	n:A	use rig.set_bandwidths
main.set_rig_frequency	d:d	use rig.set_frequency
main.set_rig_mode	n:s	use rig.set_mode
main.set_rig_modes	n:A	use rig.set_modes
main.set_rig_name	n:s	use rig.set_name
main.set_sideband	n:s	use main.set_wf_sideband and/or rig.set_mode

7.9.3 Minimized WF Window

If your external control program duplicates some of the fldigi controls such as the Rx and Tx pane you can run fldigi in a fully minimized mode. Fldigi then only provides the controls necessary for signal acquisition and waterfall management. Minimization is accomplished by setting the command line switch (--wo). The user interface then has this appearance:

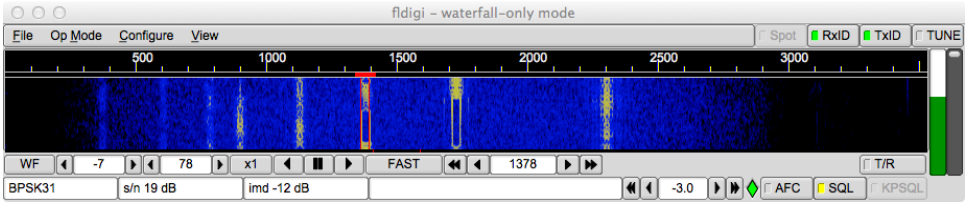

Figure 7.7: Simplified WF Window Display

The documentation for the external control program will provide additional information if this user interface is desired.

Return to Top of Page
Return to Main Page