LCLS How-to Setup the RTEMS Debugger

```
h1. How-to Setup the RTEMS Debugger (GDB)
```

Till Strauman has done all of the work for RTEMS to be able to use the GDB debugger. However, unlike the old RMX system where the debugger was built into the image, the image had all of the symbols and the MD386 debugger was a standard command, for RTEMS you need to do a few steps to enable the use of the GDB debugger. All of the information that follows is in the RTEMS documentation written by Till so this just distills that into a few easy steps.

- RTEMS at SLAC
- RTEMS at LCLS

Cheetsheet:

1. Set up your PATH to point to the location of your IOC image, the RTEMS path you're booting from and the linux host RTEMS version you're using.
2. Change your st.cmd to dynamically load and start the IOC part of the debugger.
3. link your IOC image with all of the symbols so GDB can find it's way around.
4. Start GDB and connect to your IOC.

Here are examples that I use for the upgrade project using a powerpc.

STEP 1:
Add items to your PATH to point to the RTEMS version for the GDB part the runs on Linux (in our case powerpc-rtems-gdb), your IOC RTEMS boot path so gdb can find the rtems-gdb-sub and other symbols and finally to your IOC image path.

PATH=$PATH:/afs/slac/package/rtems/vol12/4_9_3/host/i386_linux2x/bin <--- contains powerpc-rtems-gdb
PATH=$PATH:/afs/slac/package/rtems/vol12/4_9_3/target/rtems_p0/ssrlApps_p0/powerpc-rtems/beatnik/bin <--- IOC RTEMS boot path

Now point to the EPICS bin directory that YOU are loading in your st.cmd file.
Below is an example:
PATH=$PATH:/afs/slac/g/lcls/epics/iocTop/<your_path>/bin/RTEMS-beatnik

STEP 2:
Load and start the IOC part of the debugger. The rtems-gdb-stub.obj should be in the IOC RTEMS path you are using. I put it before the databases are loaded but anywhere is probably OK. Note that the lsmod() command that lists all of the modules loaded is optional but you may find it useful.

ld("rtems-gdb-stub.obj") <---- The part of GDB that runs on the IOC
rtems_gdb_start(200,0) <---- Start the debugger at priority 200
lsmod() <---- List all loaded modules

STEP 3:
Link your IOC image without symbols. The "make" default is to use -S which strips all symbols from the image. The easiest way is to have a "gdbmake" alias that sets the OP_SYS_LDFLAGS symbol to NULL like this:

alias gdbmake="make 'OP_SYS_LDFLAGS='"

When you want to use GDB on an IOC image, build it with "gdbmake" which prevents the symbols from being stripped from the image. This of course has no run-time effect but just makes the image larger.

STEP 4:
Start GDB on Linux and connect to your IOC with the "target" command. It's useful to have an alias to start the RTEMS specific version of GDB. If you're debugging the same IOC all of the time, it's also easiest to put this in your .gdbinit file in your HOME directory or you can type it after you start GDB.

alias rdb='powerpc-rtems-gdb' <---- Starts the RTEMS-specific version of GDB on Linux.
target rtems-remote ioc-li18-cv01:2159 <---- In .gdbinit connects to the IOC part of GDB at the specific port expected.

So when I'm debugging all I type is "rdb" and I'm connected to the target IOC.

That's it!

Of course there are a few caveats which are also called out in Till's documents so you should peruse them. The one most likely to fool you is that after you set breakpoints, they don't become effective until you "CONTINUE".

You may also want to turn off optimization in the Makefile that compiles the modules you're debugging by setting USR_CFLAGS += O0 (that's Oh zero).

Beats the heck out of printf statements! Most of the time.

SETTING A BREAK POINT from rdb:

break <filename>:<line#>
Example:
drvCV.c:449