EPICS Stream Device Programming

Marty Smith
APS Engineering Support Division
Agenda

- Introduction to Stream Device
- EPICS Databases and Stream Device
- Stream Device Protocols and Protocol Files
- Adding Stream Device Support to an Application
- Short Break
- Lab Session: Controlling a Network Attached Device
Introduction to Stream Device

- Generic EPICS device support for devices with a "byte stream" based communication
  - RS-232 (Local serial port or LAN/Serial adapter)
  - TCP/IP
  - VXI-11
  - GPIB (Local interface or LAN/GPIB adapter)
  - USB-TMC (Test and Measurement Class)
- A single stream device module can serve to communicate using any of the above communication mechanisms.
Introduction to Stream Device

- Command / Reply messages
  - *IDN?
  - xx:SetVoltageOut 1.2
  - Can include non-printable characters as well
- Command and reply parsing configured by protocols
- Formatting and interpretation handled with format converters
  - Similar to C printf and scanf format converters
  - Write your own converters too using the API
Introduction to Stream Device

- Provides interface to ASYN
  - Not a replacement for ASYN
  - ASYN driver provides string exchange to/from device
  - Uses AsynOctet interface

- Stream Device is not:
  - Programming language
  - No looping or conditionals available
  - Protocols are linear running from start to end
  - Rudimentary exception handlers

- How do we get stream device in a EPICS database?
Simple Command and Reply Message

- Simple command generating long response message

Data Sent: Q
Receive Data:
:SN=AT267,UN=id13,IP=164.054.008.127,V3=3390,V5=5135,V+12=12160,V-12=12396,T1=30,T2=28,T3=37,T4=00,F1=02160,F2=02130,F3=02160,F4=00000,F5=00000,F6=00000,F7=00000,F8=00000,F9=00000,OT=0,OV=0,OC=0000,PS1=1,PS2=1,MSG=0,SW=1,PROT=TEL,I3=00,I5=00,I12=00,I-12=00,CODE=64-113426F39,ENET=D6.10,POH=28896.5,MAXTMP=43,MINTMP=22,PROC=31,LOAD=FF,PWRCYCL=00083
;EV000000000ET000000000EF00000000000OT0OV0OC0000PS11MSG0SW1

- Protocol entries can be long
Stream Device EPICS Database

```c
record(bo, "$(P)$R(query") { 
    field(DESC, "Timed readback")
    field(SCAN, "10 second")
    field(PINI, "YES")
    field(FLNK, "$(P)$RVP3")
}
record(ai, "$(P)$RVP3"){
    field(DTYP, "stream")
    field(INP, "@devDawnRuSH.proto query($P$r) $(PORT) 0")
    field(FLNK, "$(P)$RVP5")
}
```

- **DTYP=stream**
- **INP/OUT fields** specify protocol file name, protocol entry (with optional arguments), ASYN port and address.
- **$(P)$R** is a protocol argument, up to nine arguments can be provided
- Address can be any value (typically 0) for single-address interfaces
Stream Device EPICS Database

- DTYP ≠ stream for protocol entry additional records:

```plaintext
record(stringin, "$(P)$(R)Serial"){
    field(DESC, "Serial number")
    field(DTYP, "Soft Channel")
}

record(ai, "$(P)$(R)VP5"){
    field(DESC, "+5V supply")
    field(DTYP, "Raw Soft Channel")
    field(EGU, "V")
    field(PREC, "3")
    ...
```
Stream Device Protocol Files

- Example protocol file
  - Use multiple lines to format for easy reading

```plaintext
query {
    out "Q";
    in ":SN=%(\$1Serial.VAL)39[^,],"
        "UN=%(\$1Name.VAL)39[^,],"
        "IP=%[^,],"
        "V3=%d,"
        ...
        "PWRCYCL=%(\$1PowerCycle.VAL)d";

    ExtraInput = Ignore;
}
```

- Notice the use of the width field – guard against buffer overruns!
Stream Device Protocols

- Defined in a plain ASCII text protocol file
- No need to compile, protocol is read by IOC at boot time
- A single entry can read/write multiple fields in one or many records
- Output records can be initialized from instrument at IOC startup
  - Providing instrument is powered on and communicating at IOC boot time
- Each protocol file describes communication to ONE device
- Protocols are defined for each function of a device
Stream Device Protocol Files

- All lines beginning with a # to the end of line are comments
- A protocol consists of a name followed by a body in {}
- Protocol entries contain statements to produce output and request input
  - Look similar to C functions
    - GetOutput {
      out "\$1";
      in "%f";
    }
    - $1 is a protocol argument, up to nine arguments can be provided
Stream Device Protocol Files

- By default the VAL or RVAL field is used as the data source/destination
- Can refer to any field, even in another record
- C-style escape sequence can be used (\r, \n, \033, \e)
- Can reload a protocol or all protocols without rebooting
  - Good for development of frequently changing protocol files
    - streamReload("recordname") – Reloads protocol for recordname
    - streamReload() – Reloads all protocols in a file
Stream Device Protocol Files

- Protocol file terminators
  - Terminators can be set globally or per entry
- Some interfaces can handle only a single character. If device replies with '\r\n' then specify In Terminator='\n' and ignore the '\r' in the reply
  - In Terminator = "\n";
  - Out Terminator = "\r";
- Better practice to use the ASYN terminators in IOC boot file
  - asyn Octet Set Output EOS and asyn Octet Set Input EOS
Stream Device Protocol Files

- Initial read back from device at IOC boot time
  - Useful to set initial value of output records to match the value presently in the instrument
  - `@init` ‘exception handler’
  - Often the same as the read back protocol entry

```plaintext
getF {
  out "\$1?";
  in "%f";
}

setF {
  @init { out "\$1?"; in "%f"; }
  out "\$1 %f";
}
```
Adding Stream Device Support

- Make changes to configure/RELEASE file
  - Add entries for streams and ASYN
    - ASYN=$(IOCAPPS)/modules/soft/asyn/4-21-asd2
    - STREAMS=$(IOCAPPS)/modules/soft/streamDevice/2-5-asd8
- Modify the application src/Makefile

```bash
.....
streams_DBD += base.dbd
streams_DBD += $(ASYN)/asyn.dbd
streams_DBD += $(ASYN)/drvAsynIPPort.dbd
streams_DBD += $(STREAMS)/dbd/stream.dbd
.....
streams_LIBS += asyn stream
.....
```
Adding Stream Device Support

- Make changes to application Db/Makelfile
  - Add entries for the instruments and ASYN

.....

DB += StreamsExample.db
DB_INSTALLS += $(TOP)/streamsApp/Db/streamEx.proto
DB_INSTALLS += $(ASYN)/db/asynRecord.db

- This copies the database and the protocol file to TOP/db directory

- The protocol file must be stored in one of the directories listed in the environment variable STREAM_PROTOCOL_PATH
Adding Stream Device Support

- Modify the IOC startup script
  
  epicsEnvSet ("STREAM_PROTOCOL_PATH", ".:${TOP}/db")
  
  ......
  
  drvAsynIPPortConfigure("$(USER)", "Device IP Address:Port", 0, 0, 0)
  
  asynOctetSetInputEos("$(USER)", -1, "Add Input Terminator Here")
  
  asynOctetSetOutputEos("$(USER)", -1, "Add Output Terminator Here")
  
  ## Load record instances
  
  dbLoadRecords "db/myDatabase.db", "P=$(USER):,PORT=$(USER),ADDR=0"
  
  dbLoadRecords "db/asynRecord.db",
  "P=$(USER):,R=device,PORT=$(USER),ADDR=0,OMAX=10,IMAX=10"
  
  ......

- P,R – PV name prefixes – PV names are $(P)$(R)name
- PORT – ASYN port name from corresponding devxxxConfigure command
Lab Session:
Control a Network Attached Device

- Host [www.xxx.yyy.zzz](http://www.xxx.yyy.zzz) – TCP Port 24742
- '\n' command terminator, '\r\n' reply terminator
- *IDN*?
  - Returns device identification string (up to 100 characters)
- LOAD?
  - Returns three floating-point numbers separated by spaces (1, 5, 15 minute load average)
- VOLTS?
  - Returns most recent voltage setting
- CURR?
  - Returns current readback (±11A)
Lab Session: Control a Network Attached Device

- **ON?**
  - Returns the current on/off status

- **ON [0,1]**
  - Turns supply OFF/ON (0/1)

- **VOLTS x.xxxx**
  - Sets voltage (±10V range)