Using MDSplus Thin Client with Long Pulse Extensions

Prepared By: Peter Milne

Date: November 14, 2011

Rev	***Date***	***Description***
2	5/11/2011	Add live scope feature
3	14/11/2011	Long Pulse

Table of Contents

1 Introduction....4
1.1 Use Cases....4
1.1.1 “Original” - regular length single shot....4
1.1.2 “Live Scope” - data is recaptured rapidly and displayed....4
1.1.3 “Repeating Shots” - data is recaptured and stored repeatedly....4
1.1.4 “Very Long Capture” - 4
1.2 References....4
1.2.1 MDSplus: 4
1.2.2 Tree creation script : 4
1.2.3 Firmware....4
2 Demo Method....5
2.1 Pre-requisites....5
2.2 Host Side Setup....5
2.3 Capture Data....6
2.4 Example Upload Script....6
2.5 Example Output....7
2.5.1 Plotting Volts:....8
3 mdsValuePut command....9
4 mdsPutCal command....9
5 Live Scope....10
5.1 Configure a Short Transient....10
5.2 mdsput-and-auto-rearm.tcl....11
5.3 Plot Data....12
5.4 Running the complete procedure....12
6 Very Long Upload....13

Copyright and Attribution.

Document created using OpenOffice.Org www.openoffice.org.

This document and D-TACQ Software comprising platform Linux port, Linux kernel modules and most applications are released under GNU GPL/FDL:

Document:

Copyright (c) 2004-11 Peter Milne, D-TACQ Solutions Ltd.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2, with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

Software:

Copyright (C) 2004-11 Peter Milne, D-TACQ Solutions Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.

This program 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, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

1 Introduction

D-TACQ *Intelligent Digitizers* include an *MDSplus* Thin Client application. The Thin Client is used to send data on the network to an *MDSIP* server. The Thin Client has been deployed for a number of years in "traditional one-shot" mode. However, it now supports the "long pulse extensions" segmented upload feature, and this allows additional flexibility

1.1 Use Cases

1.1.1 "Original" - regular length single shot

mdsPutCh is still available.

1.1.2 "Live Scope" - data is recaptured rapidly and displayed

Networked digitizer eg *ACQ196CPCI* used to make repeated short captures.

User wants to upload all data to a single MDSplus shot file, one segment per pulse.

1.1.3 "Repeating Shots" - data is recaptured and stored repeatedly

Networked digitizer eg *ACQ196CPCI* used to make repeated short captures.

The application wants to store all the data, but doesn't want the overhead of creating a new shot file each pulse.

1.1.4 "Very Long Capture" -

The Thin Client single shot mode suffers from buffer overflow when the capture is very long (>16MSamples). Segmented upload is a natural way to bypass this problem.

1.2 References

ACQxxx : one of *ACQ196CPCI, ACQ132CPCI, ACQ216CPCI, ACQ164CPCI*

1.2.1 MDSplus:

www.mdsplus.org

1.2.2 Tree creation script :

http://www.d-tacq.com/resources/make_acqtree.tar

1.2.3 Firmware

`acq2xx-2.6.21-acqX00-190.2112.3669-201111131747.tar`

2 Demo Method

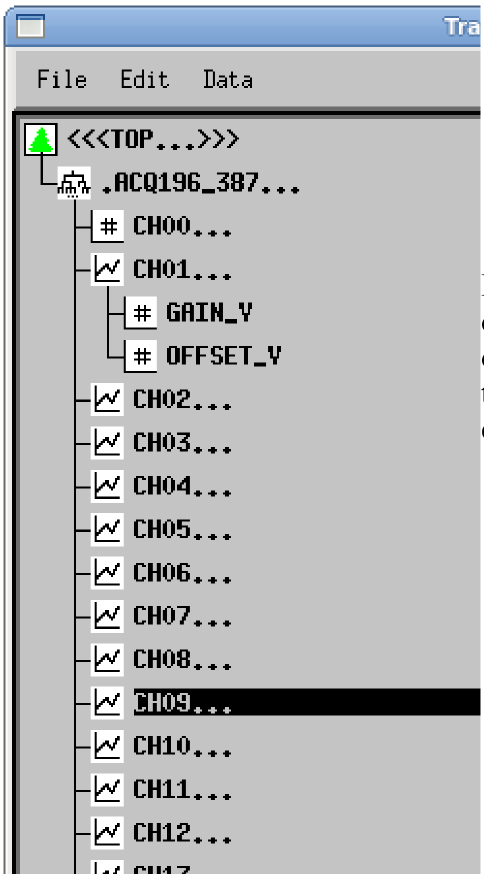
2.1 Pre-requisites

1. **MDSIP** server [1.2.1]
2. Suitable canonical tree created by **make_acqtree** [1.2.2]
3. Up to date firmware [1.2.3]

2.2 Host Side Setup

Create canonical tree using **make_acqtree**.

Tree looks like this:



```
<<<TOP...>>>
  .ACQ196_387...
    CH00...
    CH01...
      GAIN_V
      OFFSET_V
    CH02...
    CH03...
    CH04...
    CH05...
    CH06...
    CH07...
    CH08...
    CH09...
    CH10...
    CH11...
    CH12...
```

- ACQ196_387: card top level node
- CHXX : signal per channel
- CHXX.GAIN_V : scalar gain constant
- CHXX.GAIN_V : scalar offset constant

It is anticipated that each *ACQxxx* will have its own tree. This avoids *MDSplus* locking conflict on concurrent update. It is also anticipated that the trees will be grouped in sub-trees in a larger experiment.

2.3 Capture Data

Capture data in any normal way, eg

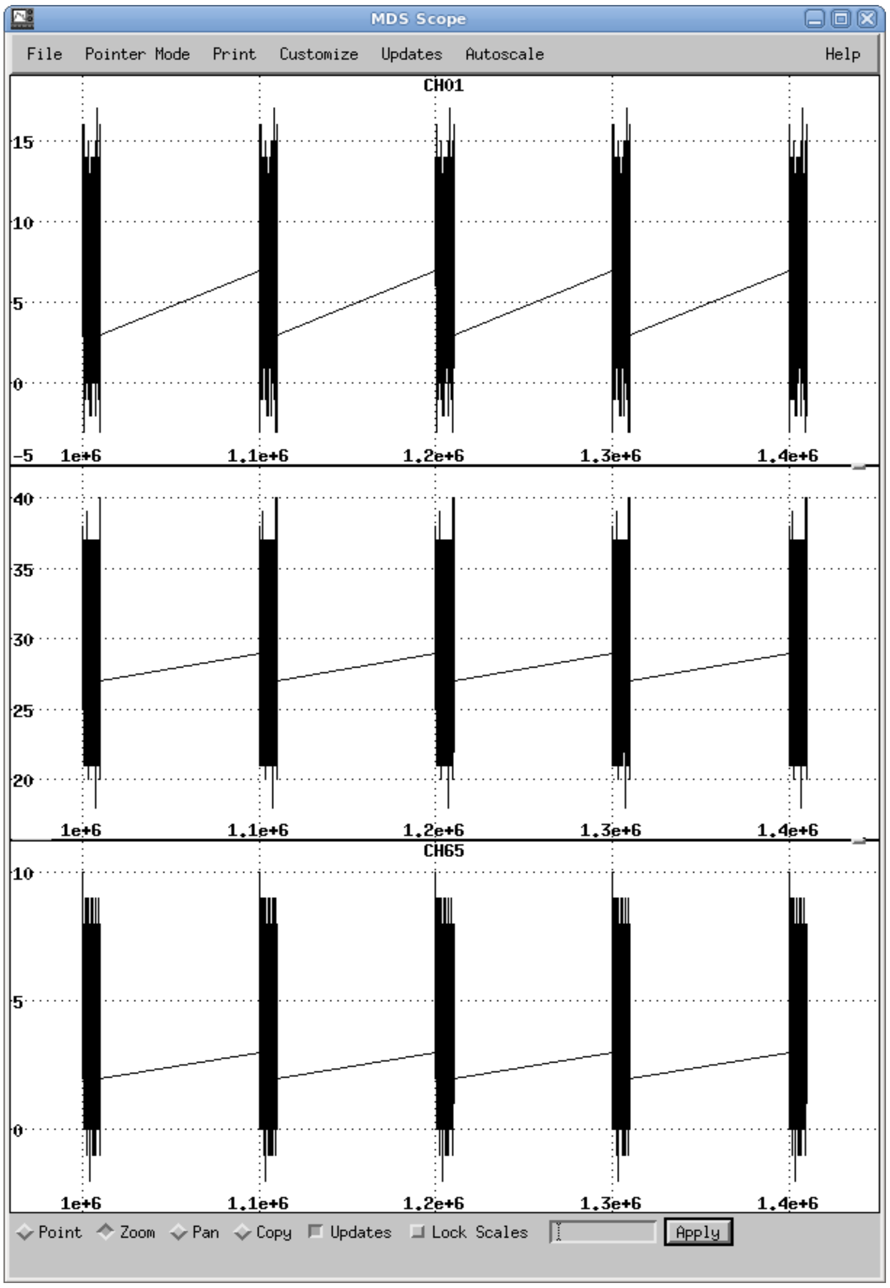
```
acqcmd setInternalClock 250000
acqcmd setMode SOFT_TRANSIENT 10240
acqcmd setArm
```

2.4 Example Upload Script

```
#!/bin/sh

# example program writes N segments
# usage:
# ./mds-write-segs
# ENV ./mds-write-segs NSEGS

let NSEGS=${1:-9}            # first argument

MDSIP=${MDSIP:-rhum4}          # MDSIP=myhost ./mds-write-segs
TREE=${TREE:-rtmt}             # TREE=mytree  ./mds-write-segs
CARD=${CARD:-ACQ196_387}       # CARD=mycard ./mds-write-segs
CHX=${CHX:-1:96}

let T1=1000000        # Start time of segment
let T2=1010000        # End time of segment
let T3=1            # Intersample interval

let TDEAD=100000       # Time betwen shots
let shot=0

EXPR='BeginSegment('$CARD'.CH%02d,$2,$3,make_dim(*,$2 : $3 : $4),$1)'

mdsConnect $MDSIP
mdsOpen $TREE
mdsPutCal --field “$CARD.CH%02d” $CHX

while [ $shot -lt $NSEGS ]
do
# a real example would run a capture here..
	mdsValuePut --ch1 $CHX --k234 $T1,$T2,$T3 "$EXPR"


	let T1=$T1+$TDEAD
	let T2=$T2+$TDEAD
	let shot=$shot+1
done

mdsClose $TREE
mdsDisconnect
```

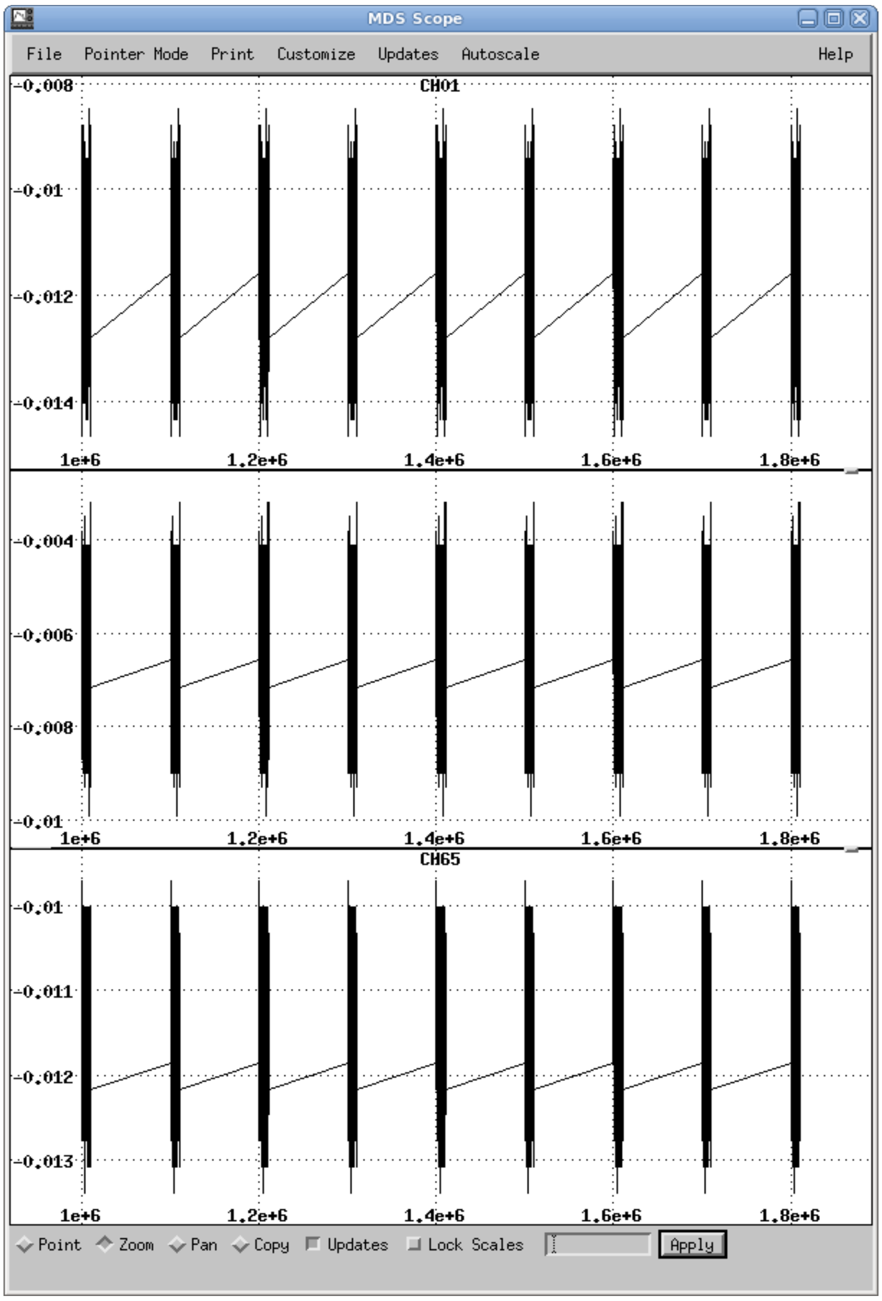
2.5 Example Output

The script mds-write-segs is simply writing the same data many times with an arbitrary timebase:



2.5.1 Plotting Volts:

Set plot expression to CH01*CH01.GAIN_V + CH01.OFFSET_V, plots calibrated volts:



3 mdsValuePut command

For a range of channels, write a segment to a field. The field name may be derived from the channel number using sprintf() convention

- **mdsValuePut** [options] EXPR

```
Usage: mdsValuePut [OPTION...]
  -d, --data=STRING      data from file [not implemented]
  -c, --ch1=STRING      channel spec subs expr arg $1 eg -c 1:96
  -k, --k234=STRING      constant spec subs expr $2,$3,$4
  --ch1_type=STRING      type of raw data [ushort (only)]
  --k234_type=STRING     type of constant [ull (only)]
  -t, --timeout=INT     timeout in s [0 : off]
  -T                    generate 64 bit timebase in CH00
  -h, --help
  -u, --usage
```

- example:

```
mdsValuePut --ch1 1:96 --k234 1300000,1400000,1 \
'BeginSegment(ACQ196_387.CH%02d,$2,$3,make_dim(*,$2 : $3 : $4),$1)'
```

4 mdsPutCal command

Writes OFFSET and GAIN calibration factors for each channel.

- **mdsPutCal** --field FIELDFMT channels
- CHANNELS: list | range
  - selects channels to read from memory
  - list : 1,2,3,6,9,12
  - range: 1-16 (or 1:16 or :)
  - channels outside channel mask are ignored.

Examples

```
mdsPutCal --field "ACQ196_387.CH%02d" 1:96
```

Environment variables OFFSET_V, GAIN_V may be used to vary the OFFSET, GAIN fields from the canonical ".OFFSET_V", ".GAIN_V" defaults.

The original one-shot implementation, **mdsPutCh** inserts voltage calibration factors into the expression. This is a valid thing to do, although it makes the expression longer, however it's not so useful for segmented access, since the expression is sent with every segment, a redundant transfer. So for the segmented access, the tree is augmented with subnodes OFFSET_V, GAIN_V, and the gain constants are set ONCE at the start of the shot.

5 Live Scope

ACQxxx can be configured to use *MDSplus* as a "Live Scope", with updates up to 1Hz achieved. By setting **dwscope** to plot on the last segment, and to update on event, continuously displays the latest data. In addition, because the data is segmented, all data is stored on disk and is available for analysis later.

Ensure that transform is ENABLED.

Ensure that **expect**/TCL is installed on the ACQ and enabled (type **expect**, if you get an expect> prompt, it's good. Otherwise, please contact D-TACQ for support.

Configure the *ACQxxx* to run a short transient capture, eg 1024 points.

Then run **/usr/local/CARE/msput-and-auto-rearm.tcl**

This is a control script that will push data to MDSplus and re-arm. The script is in control all the time, and no post-shot script is required.

5.1 Configure a Short Transient.

```
/usr/local/CARE/set.scope.prams
```

5.2 mdsput-and-auto-rearm.tcl

```
#!/usr/local/bin/expect
# runs ACQ in repeat short transients with MDSplus "live scope"
# environment vars:
# MDSHOST    - hostname or ip address of mdsip [MDSHOST]
# TREE       - tree name [hostname]
# FIELD      - field in TREE [AI.CH%02]
# CHX        - channel selection [1:96]
# ISI        - intersample interval [2 usec]
# NSAM       - samples in transient [512]

proc read_command {command}  # omitted for brevity
proc read_knob {knob}            # omitted for brevity

set HN [read_command hostname]
set US 1000000

# pull from environment or use defaults .. # Omitted for brevity

set CAPTIME "$NSAM*$ISI"
set EXPR "BeginSegment($FIELD,\$2,\$3,make_dim(*,\$2 : \$3 : \$4),\$1)"

proc putData { T1 T2 T3 } {
   global CHX EXPR
   puts "mdsValuePut -T --ch1 $CHX --k234 $T1,$T2,$T3 $EXPR"
   exec mdsValuePut -T --ch1 $CHX --k234 $T1,$T2,$T3 "$EXPR"
   exec mdsValue "setevent('HN',$T1)"
}

spawn statemon

exec mdsConnect $MDSHOST
exec mdsOpen $TREE
exec mdsPutCal --field $FIELD $CHX
exec mdsValue 42

exec acqcmd setArm

while { 1 } {
   expect {
               "ST_STOP" {
                  set shot [read_knob /dev/dtacq_drv/shot]
                  set T1 [expr $shot * $US]
                  set T2 [expr $T1 + $CAPTIME]
                  putData $T1 $T2 $ISI
                  exec mdsValue "setevent('$HN',$shot)"

                  exec acqcmd setArm
               }
   }
}

exec mdsClose ; exec mdsDisconnect
```

5.3 Plot Data

The Tree is created using **make_acqtree**.

We recommend the convention:

- Name of tree: ACQ196 HOSTNAME
- Name of “card” : AI

eg

make_acqtree acq196_137 AI,96

An example live plot state file for **dwscope**:

```
Scope.geometry: 600x1103+14+20
Scope.title: "MDS Scope"
Scope.icon_name: "Scope"
Scope.title_event:
Scope.print_file: dwscope.ps
Scope.print_event:
Scope.print_portrait: 0
Scope.print_window_title: 0
Scope.printer: To file
Scope.font: -Misc-Fixed-Bold-R-Normal--13-120-75-75-C-70-ISO8859-1
Scope.columns: 2
Scope.global_1_1.experiment: acq132_026
Scope.global_1_1.shot: 0
Scope.global_1_1.default_node: AI
Scope.global_1_1.x: rgm_t_us(1)
Scope.global_1_1.y: _ch=_index+1, rgm_v(_ch,1)
Scope.global_1_1.event: acq132_026
Scope.global_1_1.title: ''//rgm_tmin_title_us('acq132_026',_ch)
Scope.global_1_1.global_defaults: -483303
Scope.rows_in_column_1: 16
```

This makes use of a number of short tdi scripts in tdi/d-tacq

- **rgm_v(ch)** : plot latest segment for ch in volts
- **rgm_t_us()** : use hi-resolution timebase (stored in CH00)
- **rgm_tmin_title_us()** : creates a sensible title

nb: this takes extreme care to avoid numeric overflow in the timebase plot.

5.4 Running the complete procedure

- Create the tree
- Setup **dwscope**
- Run MDSHOST=myhost /usr/local/CARE/**mdsput-and-auto-rearm.tcl**
- Start a capture – eg using **dt100rc,** leave it set to the CAPTURE page to monitor progress.

6 Very Long Upload.

Very long in-memory shots (eg ACQ216, 4 channel mode, >16M samples/channel) will cause a traditional one-shot mdsPut, mdsPutCh upload to run out of memory and fail. This problem can be eliminated by using the –nsegs option to split the upload into a number of smaller segments. There's a secondary advantage in that the system is more responsive, data is available before the end of transfer.

By plotting segments, it becomes possible to deal with very large data sets that might otherwise choke output tools such as dwscope.

Example upload script:

```
mdsValuePut --window 0,64000000 --ch1 1:4 --nsegs 64
--k234=0,64000000,1 \
'BeginSegment(AI.CH%02d,$2,$3,make_dim(*,$2 : $3 : $4),$1)'


--window: range of samples to extract
--ch1     range of channels (subs $1)
--k234    timescale parameters to (subs $2,$3,$4)
--nsegs   specify number of segments
```

NB: the timebase definition assumes 64M items with a 1 unit interval. At 50Mhz, one unit is 20nSec. It's probably more physics-friendly to define the timebase in integer nano-seconds. The integers are 64bit, and so are not going to overflow during the lifetime of the equipment.

D-TACQ validated this method using code based on MDSOBJECTS to extract raw data from the tree to compare with the original.