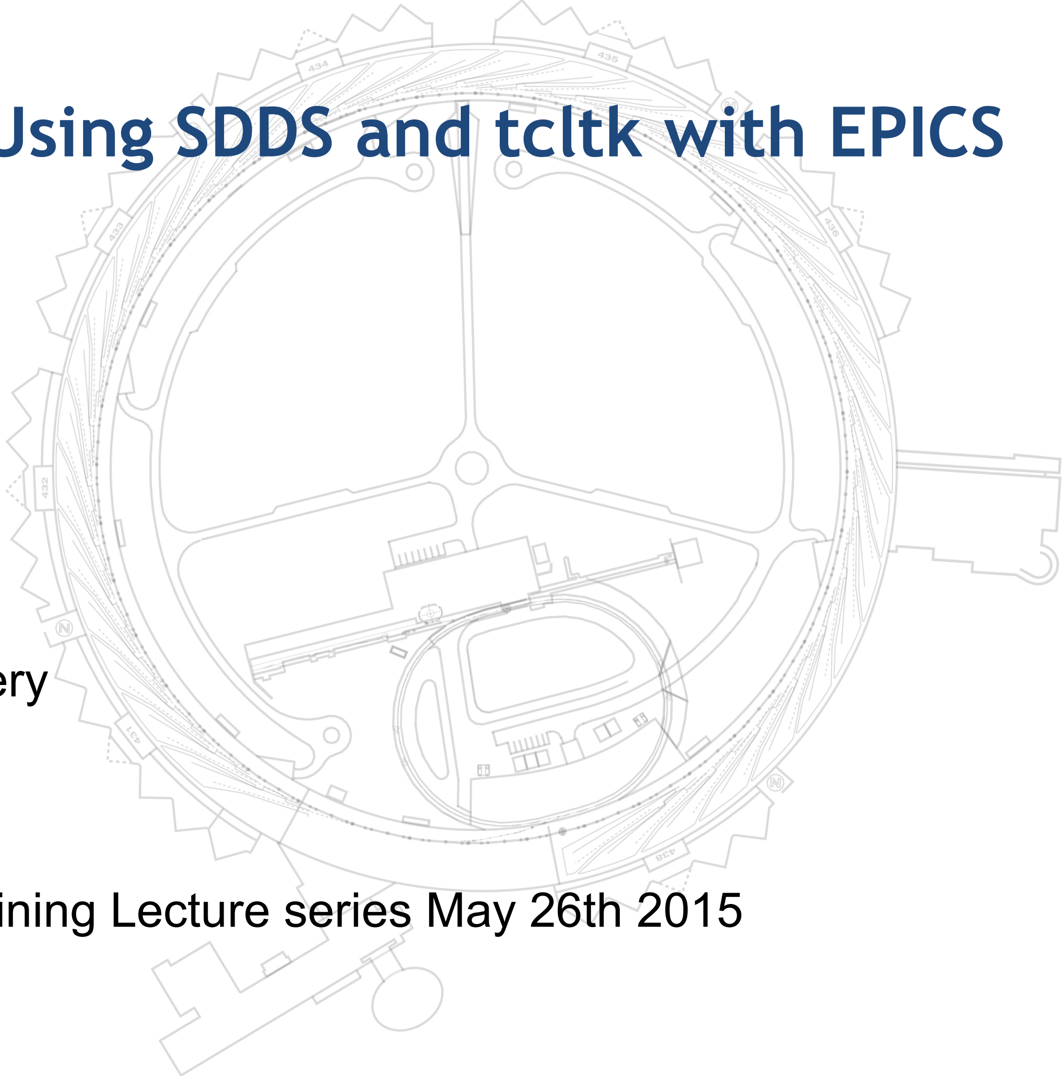
Using SDDS and tcltk with EPICS



Louis Emery
AOP
ASD

EPICS training Lecture series May 26th 2015

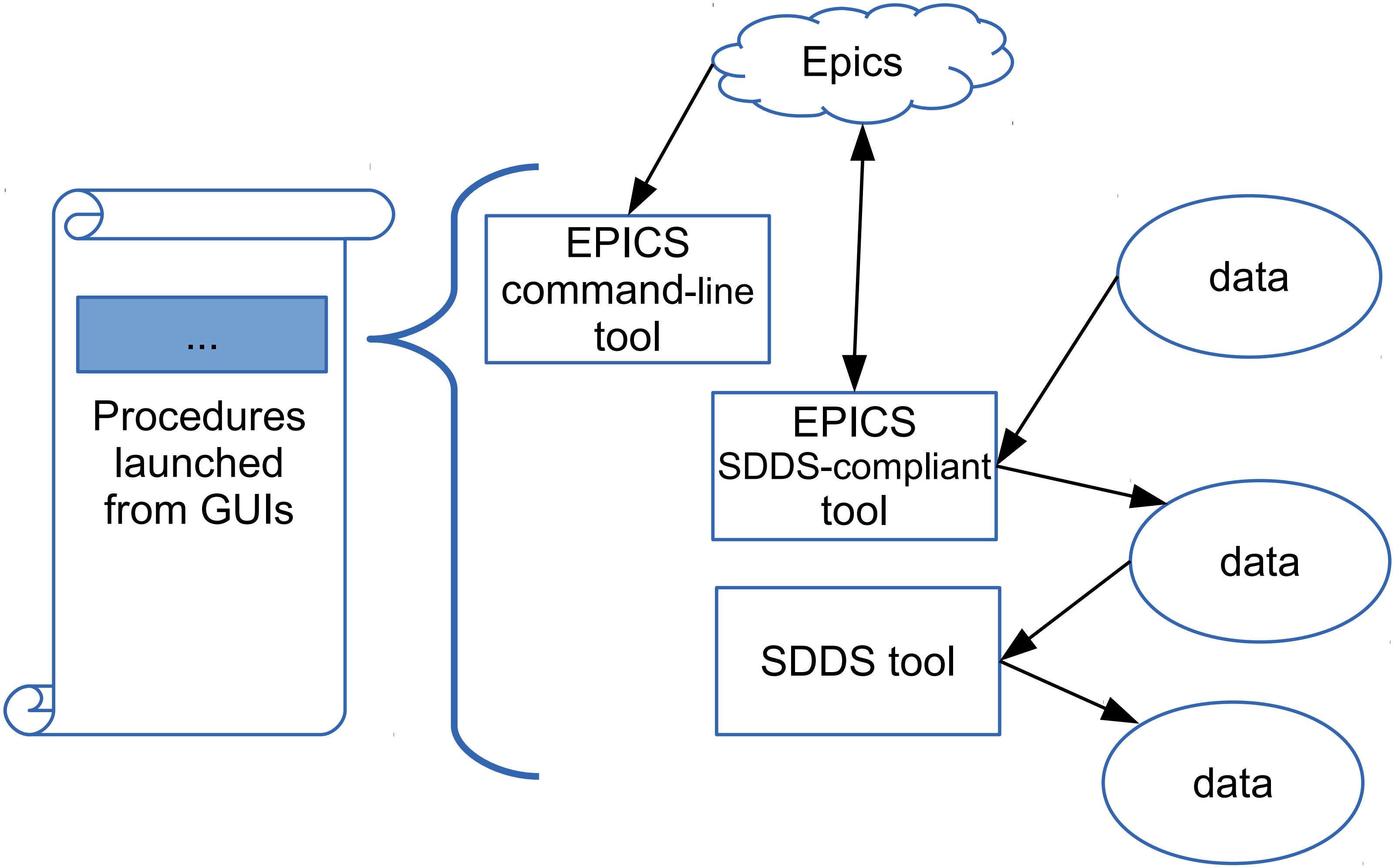
Outline

- Overall picture and philosophy of data archiving, data taking and analysis
- What is SDDS
- What is Tcl/Tk
- Examples of application
  - Setting up storage ring magnets
  - Orbit correction
  - Various beam experiments collecting data from EPICS waveforms and also non-EPICS instrumentation (which could be digitial oscilloscopes, etc.)
  - Generalized feedback: transfer lines, thermal effect compensation
  - Knobs
- Documentation of software found in http://www.aps.anl.gov/Accelerator_Systems_Division/Accelerator_Operations_Physics/oagSoftware.shtml

Use of Data Files in Operations of APS

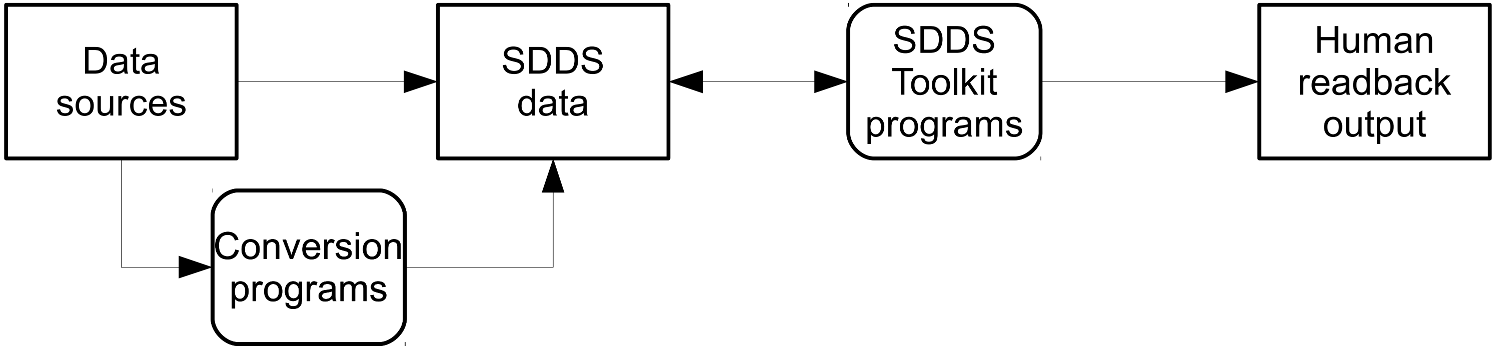
- EPICS hold live data
- Record all data of interest into file
  - To better organize efforts of various kinds, we require a file protocol to read, store and handle all data
- Applications: Put all data and configuration settings in files (as opposed to resident memory) and use software tools as filters from which you can build larger and complicated processes
  - Modularity (make complex operation from smaller pieces)
  - Reusability (e.g. feedback configurable by supplied PV list)
  - Repurposed by others

Control Room Work



What is SDDS

- SDDS stands for "Self Describing Data Sets"
- A standardized way to store and access data, i.e., a "file protocol"
- A group of ~100 programs use this file protocol
- These programs are the "tools" in the SDDS Toolkit
- Programs in the SDDS toolkit can be used to sequentially transform SDDS data sets
  - Use of Unix pipes is extensive
- Each tool makes others more useful without advance planning by developers



Examples of SDDS Toolkit Function

General tools

- Data display
- Plotting (2 programs)
- Printing data as formatted text
- Summarizing data set contents
- Data processing
- Equation evaluation
- Data filtering and outlier removal
- Statistics, histograms, and correlations
- Fitting and smoothing
- Matrix operations (e.g., SVD)
- Cross-referencing, sorting, and collation
- FFTs and digital filtering

EPICS tools

- Data collection from EPICS
- Logging data at fixed time intervals
- Event-driven data logging
- Alarm logging
- N-dimensional experiments
- Save/restore of EPICS data
- Control functions for EPICS
- Generalized feedback control
- Generalized optimization

Tutorial on SDDS

- Protocol requires every data element has a name and forbids access to data except via name
- File header: namelist description of a structure of an arbitrary number of parameters and arrays, and a data table of arbitrary rows and columns
- Zero or more instances of the structure
- Simplicity of protocol makes the SDDS toolkit feasible

```
SDDS1
&parameter name=pauseBetweenReadings, type=double, &end
&column name=ControlName, type=string,  &end
&column name=LowerLimit, type=double,  units=kV &end
&column name=UpperLimit, type=double,  units=kV &end
&column name=InitialChange, type=double,  units=kV &end
&data mode=ascii, &end
! page number 1
 0.000000000000000e+00
                2
S:IK3:VoltageSetSendAO  0.000000000000000e+00  1.100000000000000e+01  2.000000000000000e-01
S:IK4:VoltageSetSendAO  0.000000000000000e+00  1.500000000000000e+01  2.000000000000000e-01
```

Example of a small file

http://www.aps.anl.gov/Accelerator_Systems_Division/Accelerator_Operations_Physics/SDDSIntroTalk/slides.html

Other Use of SDDS

- SDDS file protocol can be used for physics simulation codes
  - Outside codes can be modified to use SDDS input and produce SDDS output
- Data obtained from the “off-line” optimization of simulation work can be used in the control room without the need of conversion programs

What is Tcl

- Scripting language “Tool Command Language”
- Simple syntax, few special characters
- Data structures such as list and arrays
- The usual control structures (`if, foreach while,` etc)
- Built-in event-driven programming
- GUI toolkit Tk seen elsewhere (perl, python) was designed for Tcl
- Launch with tclsh (or oagtclsh, our version)
- These are the features I was interested in. More features listed in http://wiki.tcl.tk/299

Tutorial on Tcl/Tk

- Launch with `tclsh`
- Variables; there is no types

```
set phase 180
puts $phase
```

- Lists

```
set a {1 2 3}
puts $a
```

- User-defined procedures

```
proc calculateSquare x {
    return [expr x*x]
}
```

- […] means run the procedure inside and return a value

Fun recipe.tcl example

```
proc ? L {
   # returns an random index
   lindex $L [expr {int(rand()*[llength $L])}]
}

proc recipe {} {
    # create four lists
    set a {
        {3 eggs} {an apple} {a pound of garlic}
        {a pumpkin} {20 marshmallows}
    }
    set b {
        {Cut in small pieces} {Dissolve in lemonade}
        {Bury in the ground for 3 months}
        {Bake at 300 degrees} {Cook until tender}
    }
    set c {parsley snow nutmeg curry raisins cinnamon}
    set d {
        ice-cream {chocolate cake} spinach {fried potatoes} rice {soy sprouts}
    }
    # returns a recipe, i.e. one of each list
    return "Take [? $a].\n[? $b].\nTop with [? $c].\nServe with [? $d]."
}
```

Run recipe example

```
>oagtclsh
% source recipe.tcl
% recipe
Take 20 marshmallows.
Cut in small pieces.
Top with raisins.
Serve with spinach.
% recipe
Take an apple.
Bury in the ground for 3 months.
Top with parsley.
Serve with rice.
% exit
```

Other Tcl Features

- Run external commands with `exec command <arguments..>`
  - `set Idipole [exec caget -n SBM:CurrentAO]`
- Catching error codes. Used extensively in our software.
- Regexp
- Some file I/O, though most of the time our complex data files are handled by SDDS toolkit
- Socket communication, e.g. HP instruments
- Events
- Many extensions available, including
  - "pv" package for channel access, and
  - "sdds" package for writing tcl data to and from SDDS files directly.

Command Syntax of pv package

- Examples:
  ```
  pv linkw S35BeamCurrent S35DCCT:currentCC
  pv linkw $apsTopupTclVarList $apsTopupPvList
  ```
- Second word: one of command operation `link, unlink, get, getw, put, putw, putq, info, stat, mon, umon, cmon`
- Third word: list of names of tcl variables
- Other words: depend on operation. For `linkw` it is the list of names of PVs

http://www.aps.anl.gov/Accelerator_Systems_Division/Accelerator_Operations_Physics/manuals/APStclCA/APStclCA.html

Example of pv get

```
pv linkw S35BeamCurrent S35DCCT:currentCC
pv getw S35BeamCurrent
if {$S35BeamCurrent < 102} {
    inject-more-beam
}
```

More Complex Example of pv Package

```
# make links between tcl variables and PV names
# The tcl variables are arrays, BPx(1), BPx(2), etc, one for each sector of APS
# There are 4 elements in the tcl variables list
# The 4 PV names are setpoints for BPMs of a particular sector
set sector 1
set Sn 1
set Sn1 2
pv linkw \
      [list BPx($sector) BPy($sector) APx($sector) APy($sector)] \
      [list S${Sn}B:P1:x:SetpointAO S${Sn}B:P1:y:SetpointAO \
        S${Sn1}A:P1:x:SetpointAO S${Sn1}A:P1:SetpointAO ]
# set up monitor with a script to execute
pv umon BPx($sector) {set  BPx($sector) $BPx($sector)}
pv umon BPy($sector) {set  BPy($sector) $BPy($sector)}
```

Example of pv Package with Catch Statements

```
# make links between tcl variables and PV names
# The tcl variables are arrays, BPx(1), BPx(2), etc, one for each sector of APS
# The PV names are setpoints for BPMs of a particular sector
set sector 1
set Sn 1
set Sn1 2
If { [pv linkw \
        [list BPx($sector) BPy($sector) APx($sector) APy($sector)] \
        [list S${Sn}B:P1:x:SetpointAO S${Sn}B:P1:y:SetpointAO \
           S${Sn1}A:P1:x:SetpointAO S${Sn1}A:P1:SetpointAO ] ] != 0} {
    APSAlertBox .alert -errorMessage "linkw error $errorCode"
    exit
}
# set up monitor with a script to execute
If { [pv umon BPx($sector) {set  BPx($sector) $BPx($sector)}  ] != 0} {
    APSAlertBox .alert -errorMessage "umon error $errorCode"
    exit
}
If { [pv umon BPy($sector) {set  BPy($sector) $BPy($sector)}  ] != 0} {
    APSAlertBox .alert -errorMessage "umon error $errorCode"
    exit
}
```

About 60 GUIs use pv package

- ExperimentDesigner
- PVmonitor
- SRIDSteering, SRBMIntensityOptimization
- SRBunchTrain
- SREnergyApVoltScan
- SRRFPhaseSliders
- TclKnobs
- Also many libraries, cautils.tcl devices.tcl APSRunControl.tcl

What is Tk

- Windowing toolkit, extension of Tcl
- Creates and manipulates widgets in a window
- Buttons, labels, text boxes, list boxes, scrollbar
- Extensions available for fancy elements like tabs.
- Launch with `wish` (or `oagwish`, our version)
- More at http://wiki.tcl.tk/487
- APS has built a standard set of Tk calls for APS GUIs

APS Tcl/Tk Library (1996)

- Collection of widget procedures for creating Tk applications with a consistent look and feel
- Calling convention:

  ```
  APSWhatever <widget> [<option-list>]
  ```

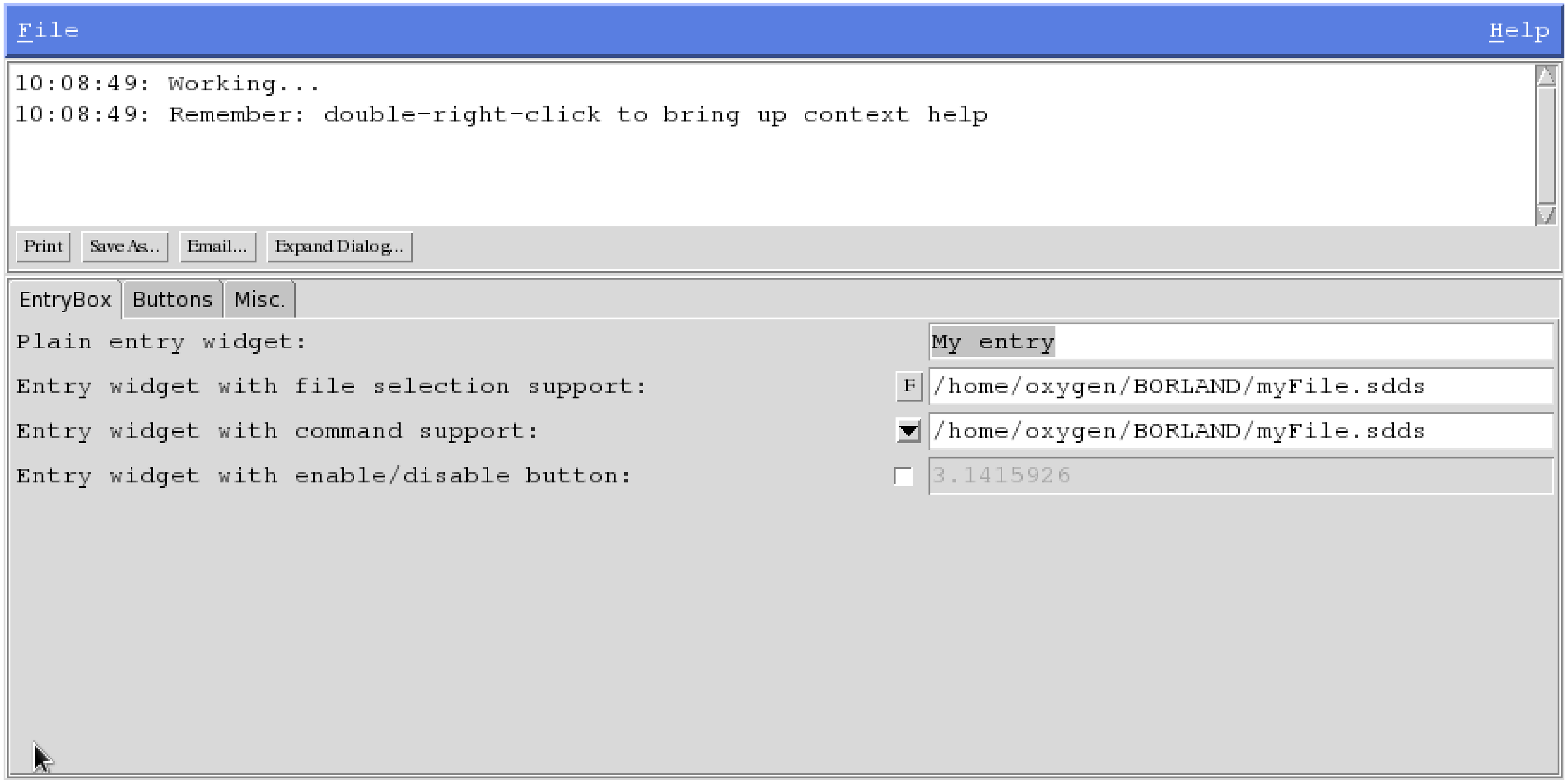
  where <option-list> is a list of -name value pairs.
- Options common to most procedures

  ```
  -parent <widget>
  -packOption <list>
  -contextHelp <string>
  ```
- Use `APSHelp` to get the latest list of procedures and usage

http://www.aps.anl.gov/Accelerator_Systems_Division/Accelerator_Operations_Physics/manuals/APSTk/APSTk4.html

Demo Script of APS widgets

/usr/local/oag/apps/bin/linux-x86_64/demoScript



```
10:08:49: Working...
10:08:49: Remember: double-right-click to bring up context help
```

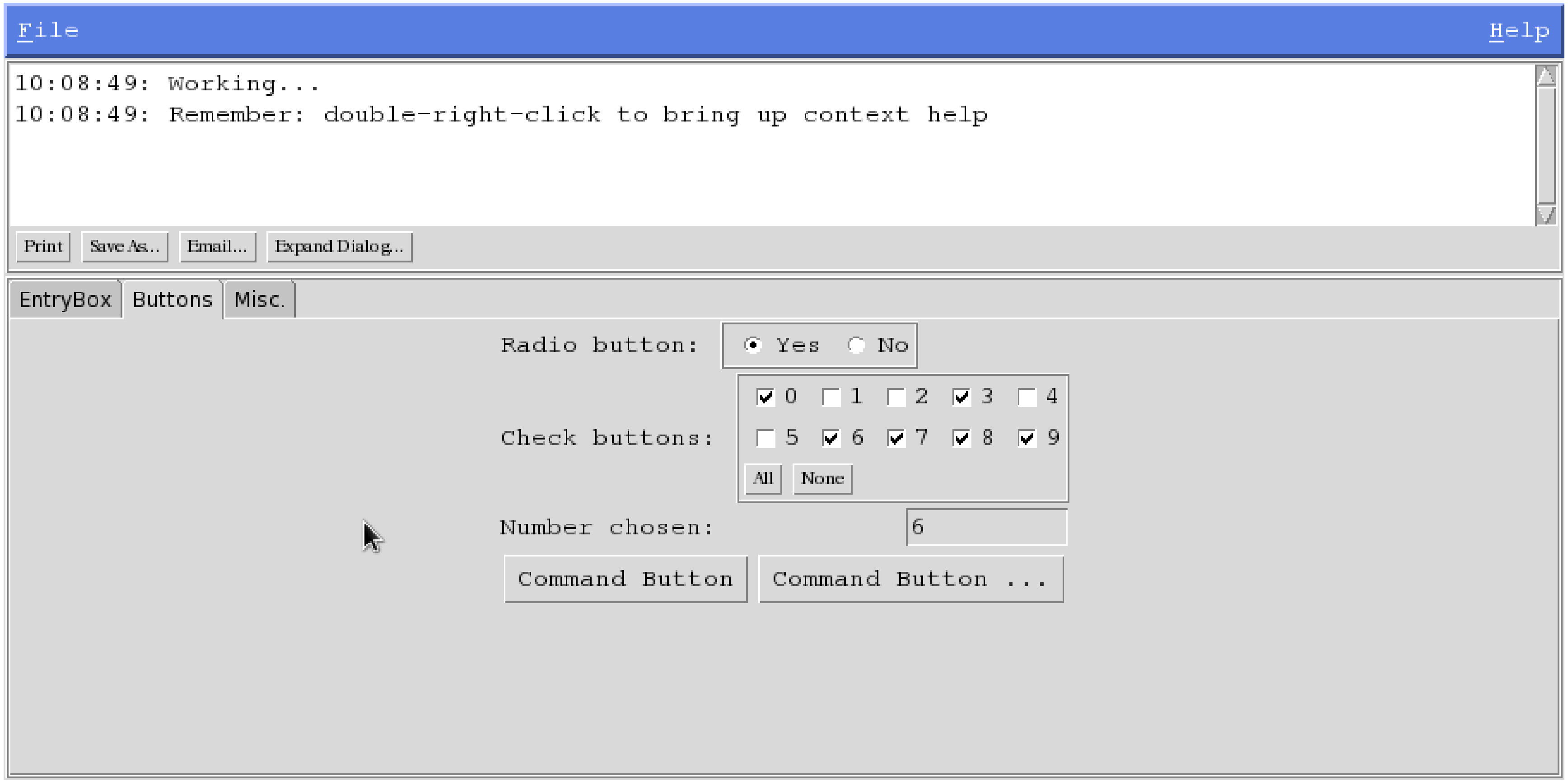
Plain entry widget: My entry

Entry widget with file selection support: /home/oxygen/BORLAND/myFile.sdds

Entry widget with command support: /home/oxygen/BORLAND/myFile.sdds

Entry widget with enable/disable button: □ 3.1415926

Demo Script of APS widgets



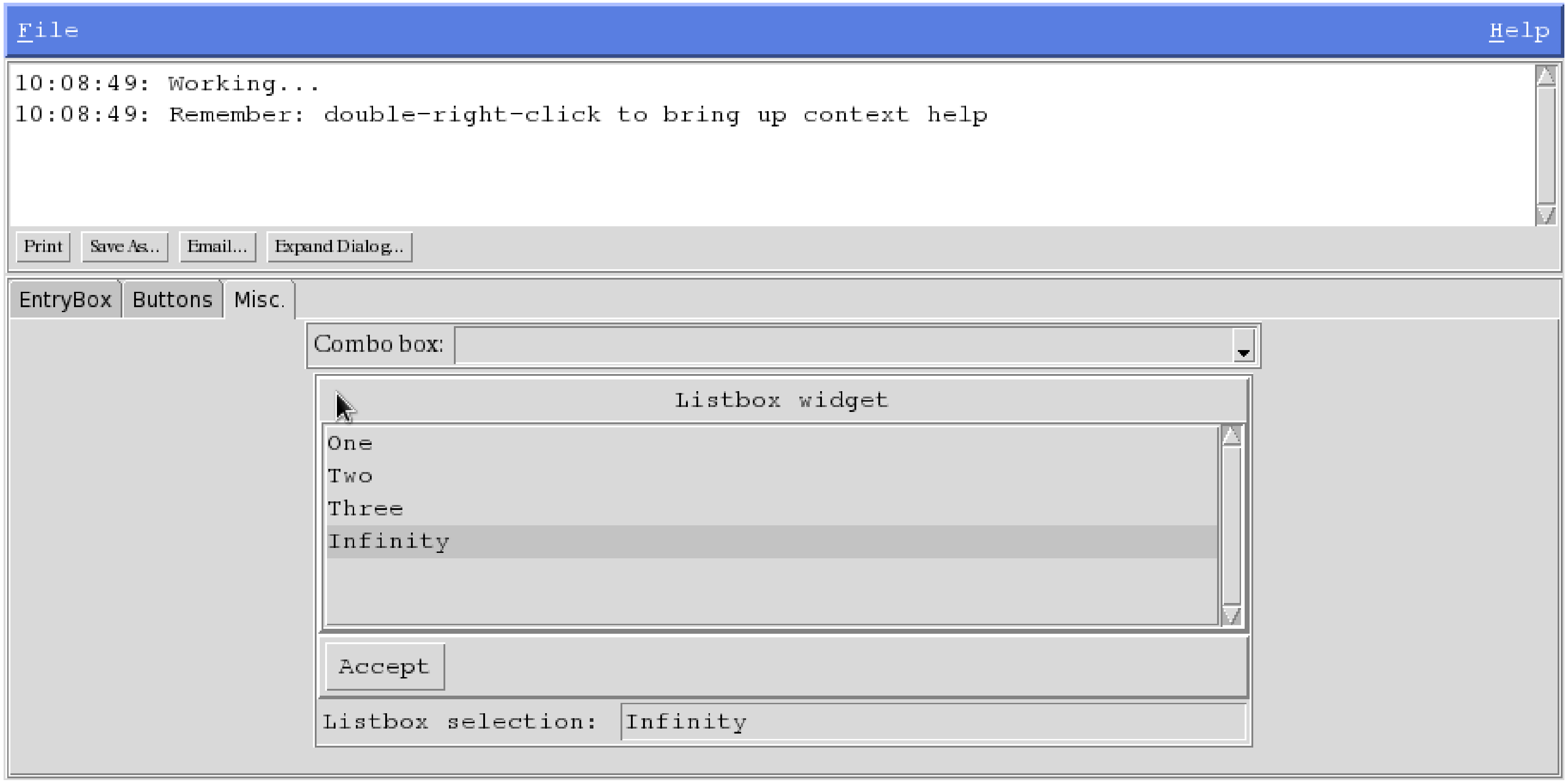
```
10:08:49: Working...
10:08:49: Remember: double-right-click to bring up context help
```

Radio button: ● Yes ○ No

Check buttons:
☑ 0 □ 1 □ 2 ☑ 3 □ 4
□ 5 ☑ 6 ☑ 7 ☑ 8 ☑ 9

Number chosen: 6

Demo Script of APS widgets



```
10:08:49: Working...
10:08:49: Remember: double-right-click to bring up context help
```

Combo box:

Listbox widget

One
Two
Three
Infinity

Listbox selection: Infinity

Use of tag/value pairs in Tcl/Tk calling procedures

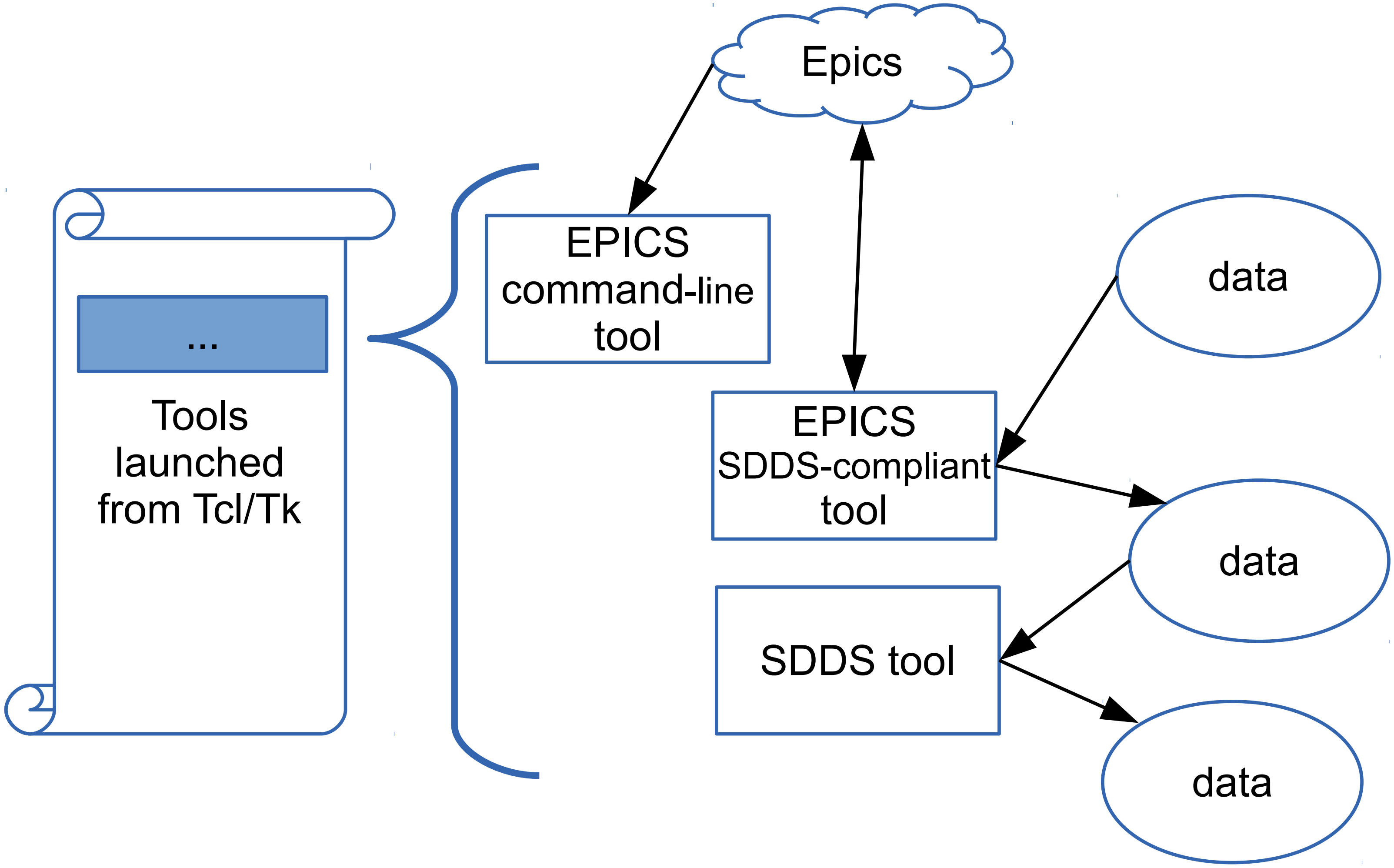
- APS Tcl scripts and procedures are called with tag-value pair arguments, making it unnecessary to remember the order of arguments.

  ```
  myAPScommand -var1 val1 -var2 val2
  ```

- These items are converted into local variables inside procedure disregarding the order

  ```
  proc myAPScommand {args} {
      # new local variables will be creates from
      # the args list
      APSParseArguments {var1 var2}
      puts $var1
      puts $var2
  }
  ```

Script working with SDDS and EPICS toolkits



Three ways to communicate with EPICS from Tcl

- pv extension package, e.g.
  - `pv putw SRdipoleMain 450`
- External command-line interface, e.g.
  - `exec caput S:BM:CurrentAO 450`

  -
    ```
    exec cavput -list=S -range=beg=1,end=40 \
    -list=A:QS4:CurrentAO=0.1 -delta
    ```
- APS extension library

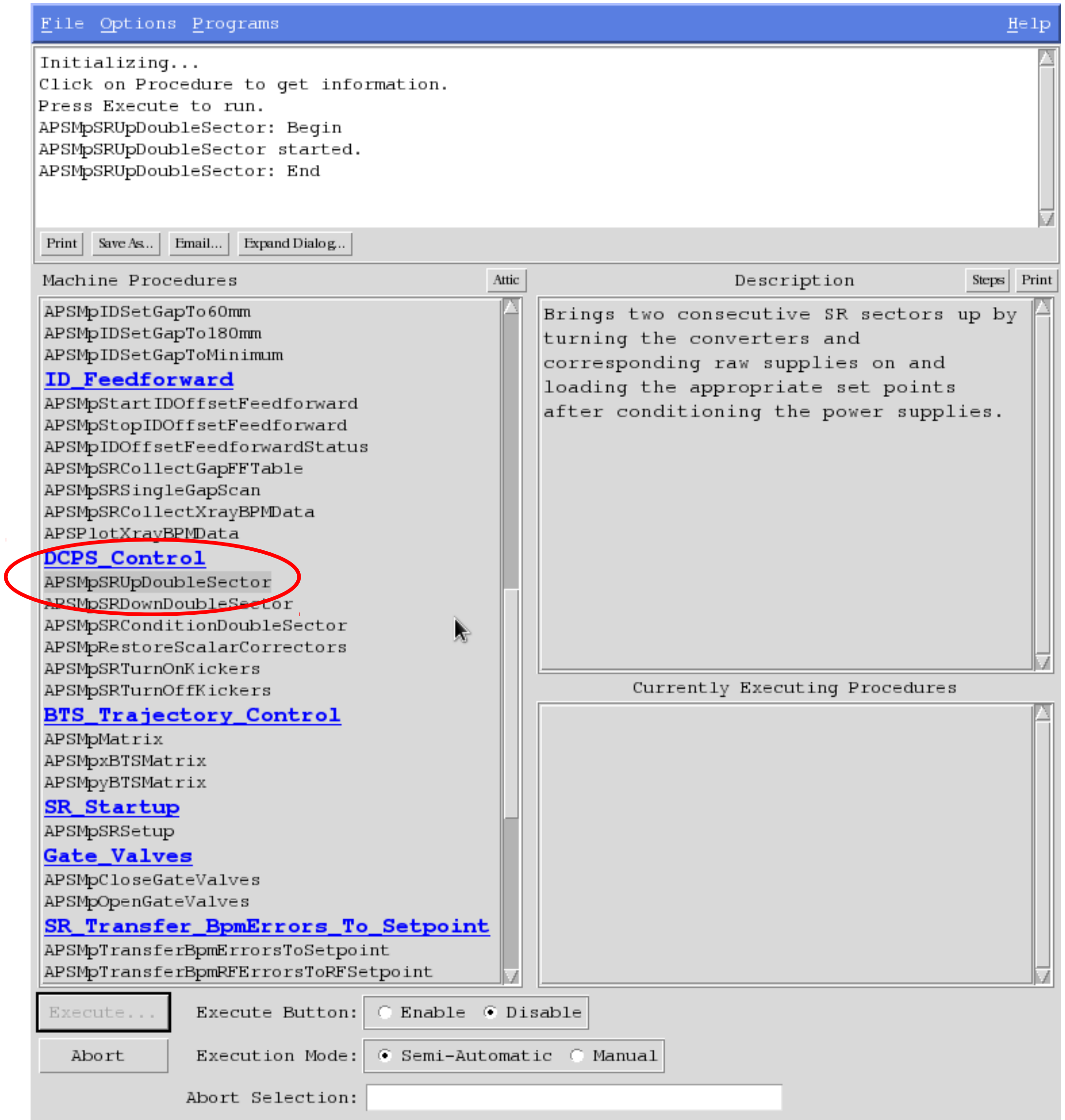
  -
    ```
    APScavput -list=S -range=beg=1,end=40 \
    -list=A:QS4:CurrentAO=0.1 -delta
    ```
- SDDS-compliant toolkit, e.g.
  - `exec sddscasr -restore SR.snp`
  - `exec sddsmonitor PV.mon PV-01 -interval=1 -time=1,day`
  - `exec sddscontrollaw matrix -interval=1 -gain=0.5`
- Choice depends on complexity and requirement for leaving CA connections open

Example of procedure: Starting up SR magnets

- Goal: run the necessary steps to turn on power supplies of SR magnets, run conditioning cycles, and leave magnets running with currents of predetermined set points. Also we need to monitor progress.
- A high-level procedure was written that
  - Requests from the operator which archived file to use for current set points
  - Selected possible a subset of sectors or magnet types
  - Creates a file of PVs that configures the power supplies cycling
  - Sends “configure” commands to power supplies ioc
  - Sends “On” commands to all power supplies. Waits for completion
  - Sends “Start conditioning” commands. Waits for completion
  - Sends final values for current to power supplies
  - Pop-up window gives progress.

http://www.aps.anl.gov/Accelerator_Systems_Division/Accelerator_Operations_Physics/manuals/APSPEM/APSPEM4.html

Procedure Launched from PEM interface



```
Initializing...
Click on Procedure to get information.
Press Execute to run.
APSMpSRUpDoubleSector: Begin
APSMpSRUpDoubleSector started.
APSMpSRUpDoubleSector: End
```

APSMpIDSetGapTo60mm
APSMpIDSetGapTo180mm
APSMpIDSetGapToMinimum
ID_Feedforward
APSMpStartIDOffsetFeedforward
APSMpStopIDOffsetFeedforward
APSMpIDOffsetFeedforwardStatus
APSMpSRCollectGapFFTable
APSMpSRSingleGapScan
APSMpSRCollectXrayBPMData
APSPlotXrayBPMData
DCPS_Control
APSMpSRUpDoubleSector
APSMpSRDownDoubleSector
APSMpSRConditionDoubleSector
APSMpRestoreScalarCorrectors
APSMpSRTurnOnKickers
APSMpSRTurnOffKickers
BTS_Trajectory_Control
APSMpMatrix
APSMpxBTSMatrix
APSMpyBTSMatrix
SR_Startup
APSMpSRSetup
Gate_Valves
APSMpCloseGateValves
APSMpOpenGateValves
SR_Transfer_BpmErrors_To_Setpoint
APSMpTransferBpmErrorsToSetpoint
APSMpTransferBpmRFErrorsToRFSetpoint

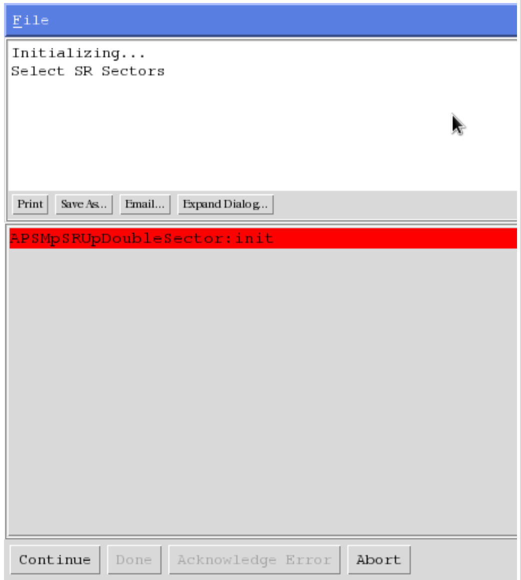
Brings two consecutive SR sectors up by turning the converters and corresponding raw supplies on and loading the appropriate set points after conditioning the power supplies.

Execute Button: ○ Enable ● Disable

Execution Mode: ● Semi-Automatic ○ Manual

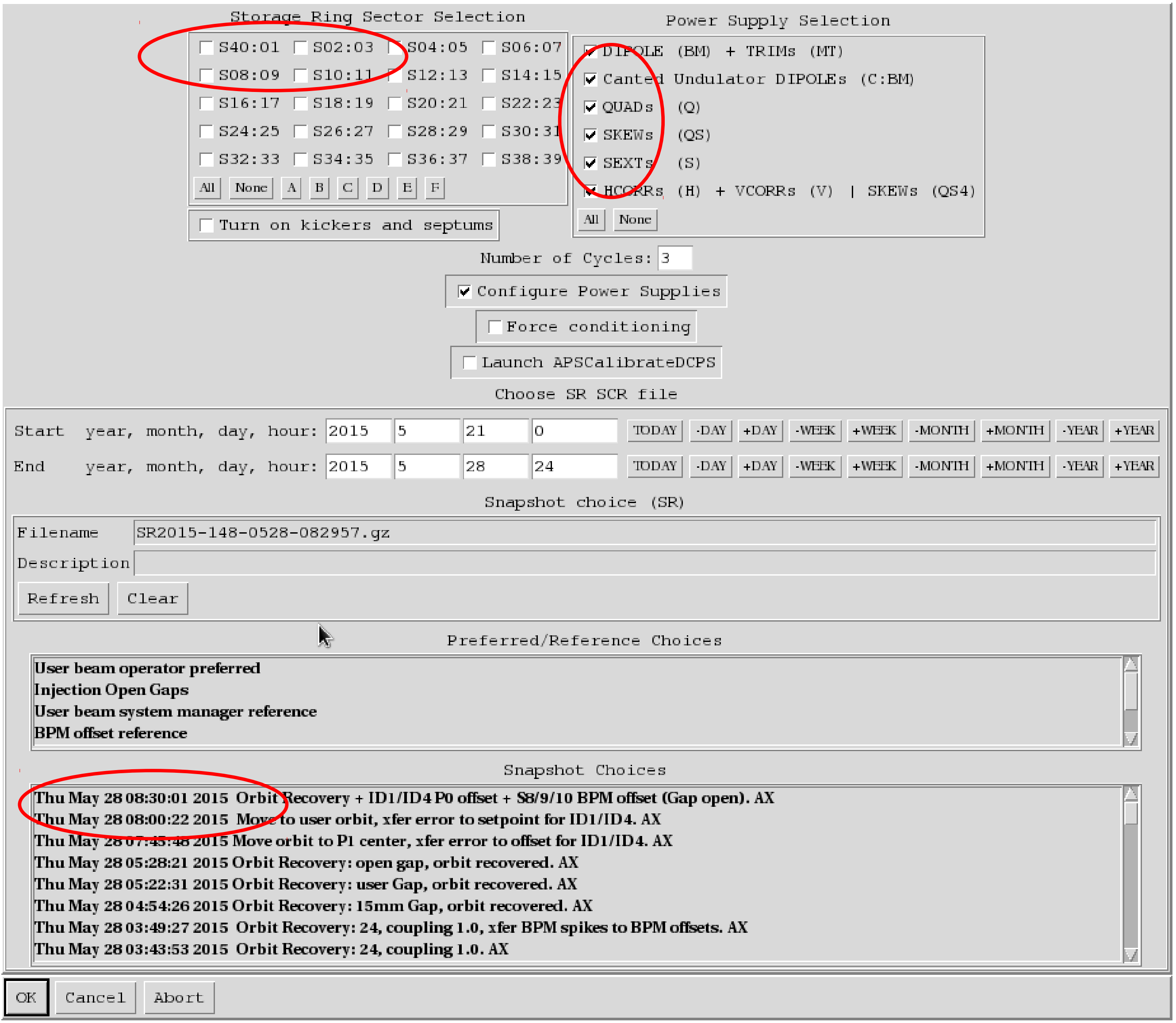
Abort Selection:

Windows for Starting SR Magnets



```
Initializing...
Select SR Sectors
```

```
APSMpSRUpDoubleSector:init
```



Storage Ring Sector Selection

□ S40:01 □ S02:03 □ S04:05 □ S06:07
□ S08:09 □ S10:11 □ S12:13 □ S14:15
□ S16:17 □ S18:19 □ S20:21 □ S22:23
□ S24:25 □ S26:27 □ S28:29 □ S30:31
□ S32:33 □ S34:35 □ S36:37 □ S38:39

□ Turn on kickers and septums

Power Supply Selection

☑ DIPOLE (BM) + TRIMs (MT)
☑ Canted Undulator DIPOLEs (C:BM)
☑ QUADs (Q)
☑ SKEWs (QS)
☑ SEXTs (S)
☑ HCORRs (H) + VCORRs (V) | SKEWs (QS4)

Number of Cycles: 3

☑ Configure Power Supplies

□ Force conditioning

□ Launch APSCalibrateDCPS

Choose SR SCR file

Start year, month, day, hour:	2015	5	21	0
End year, month, day, hour:	2015	5	28	24

Snapshot choice (SR)

Filename SR2015-148-0528-082957.gz

Description

Preferred/Reference Choices

```
User beam operator preferred
Injection Open Gaps
User beam system manager reference
BPM offset reference
```

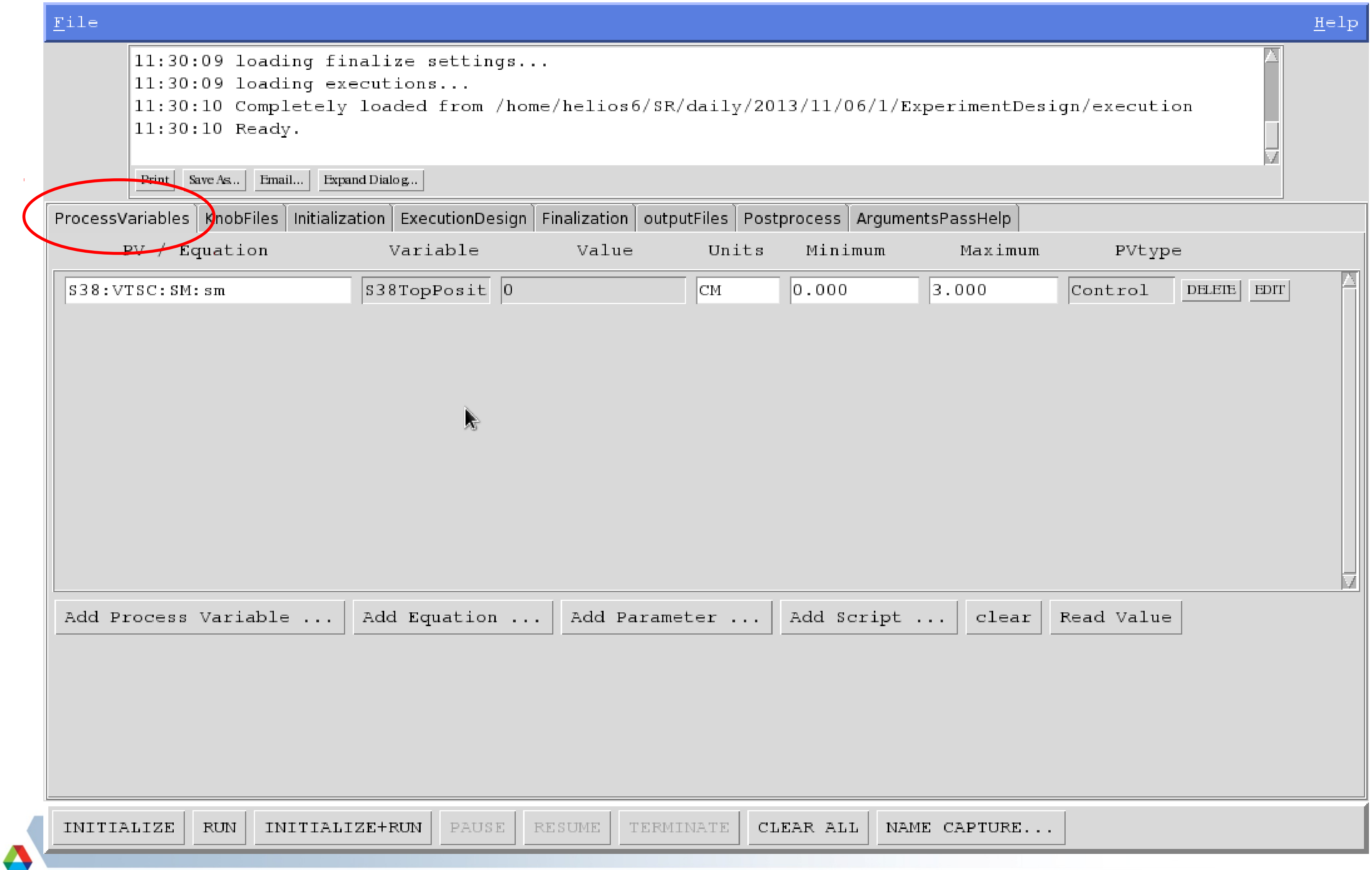
Snapshot Choices

```
Thu May 28 08:30:01 2015 Orbit Recovery + ID1/ID4 P0 offset + S8/9/10 BPM offset (Gap open). AX
Thu May 28 08:00:22 2015 Move to user orbit, xfer error to setpoint for ID1/ID4. AX
Thu May 28 07:45:48 2015 Move orbit to P1 center, xfer error to offset for ID1/ID4. AX
Thu May 28 05:28:21 2015 Orbit Recovery: open gap, orbit recovered. AX
Thu May 28 05:22:31 2015 Orbit Recovery: user Gap, orbit recovered. AX
Thu May 28 04:54:26 2015 Orbit Recovery: 15mm Gap, orbit recovered. AX
Thu May 28 03:49:27 2015 Orbit Recovery: 24, coupling 1.0, xfer BPM spikes to BPM offsets. AX
Thu May 28 03:43:53 2015 Orbit Recovery: 24, coupling 1.0. AX
```

Choices in Running an Experiment

- For the purpose of this discussion, an “experiment” is a scan of one or more control variables (could be discrete values form a list) and the collection of physcially-dependent quantities
- Use a loop in a Tcl/Tk script or GUI
- Use sddsexperiment with configuration file describing with namelist commands the control variable and the monitored variables, and also the pauses between the steps
- Use ExperimentDesigner for very complex situations that need scripts to be run in between steps

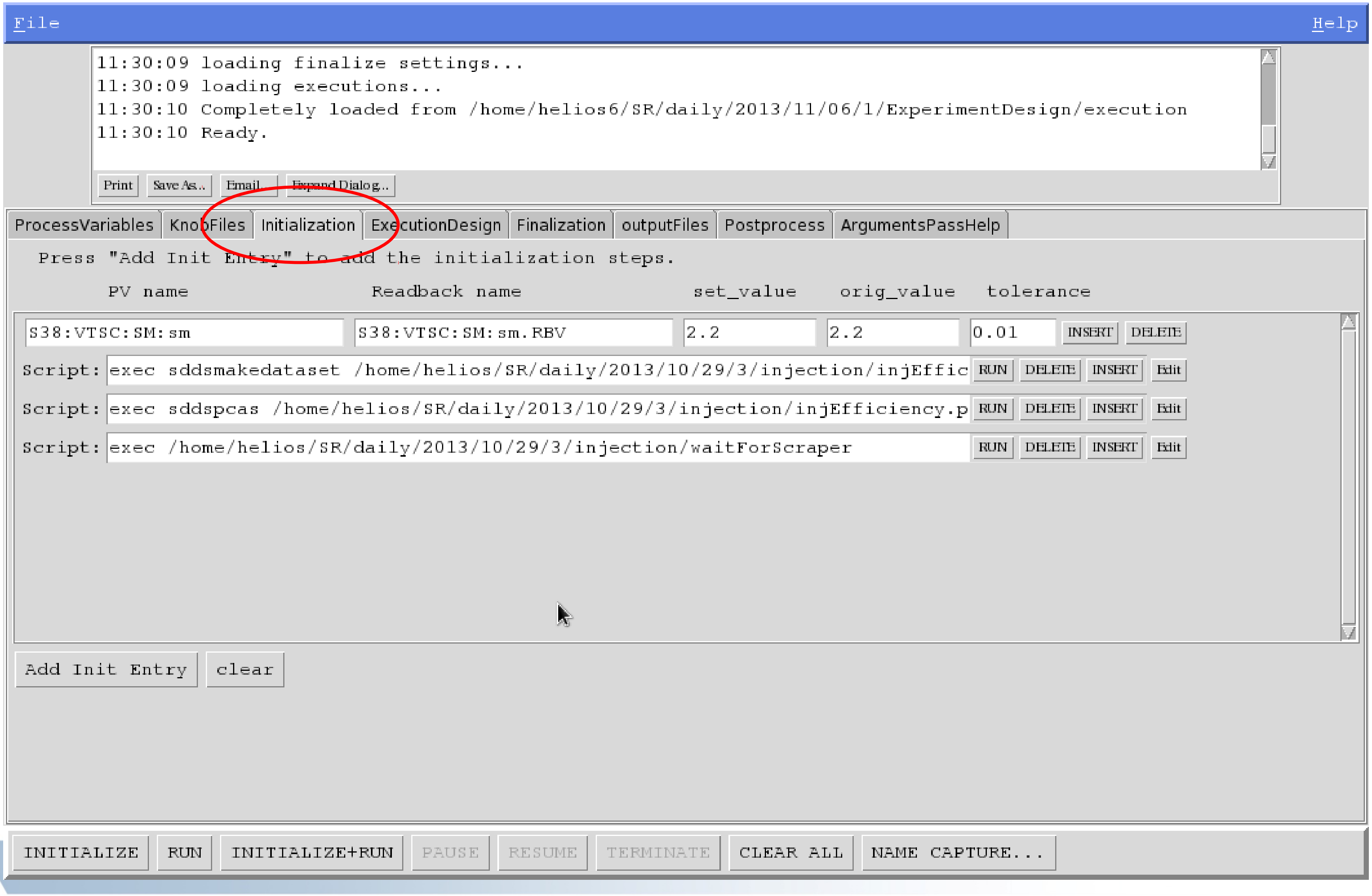
Experiment Designer



```
11:30:09 loading finalize settings...
11:30:09 loading executions...
11:30:10 Completely loaded from /home/helios6/SR/daily/2013/11/06/1/ExperimentDesign/execution
11:30:10 Ready.
```

PV / Equation	Variable	Value	Units	Minimum	Maximum	PVtype		
S38:VTSC:SM:sm	S38TopPosit	0	CM	0.000	3.000	Control	DELETE	EDIT

Experiment Designer



```
11:30:09 loading finalize settings...
11:30:09 loading executions...
11:30:10 Completely loaded from /home/helios6/SR/daily/2013/11/06/1/ExperimentDesign/execution
11:30:10 Ready.
```

Press "Add Init Entry" to add the initialization steps.

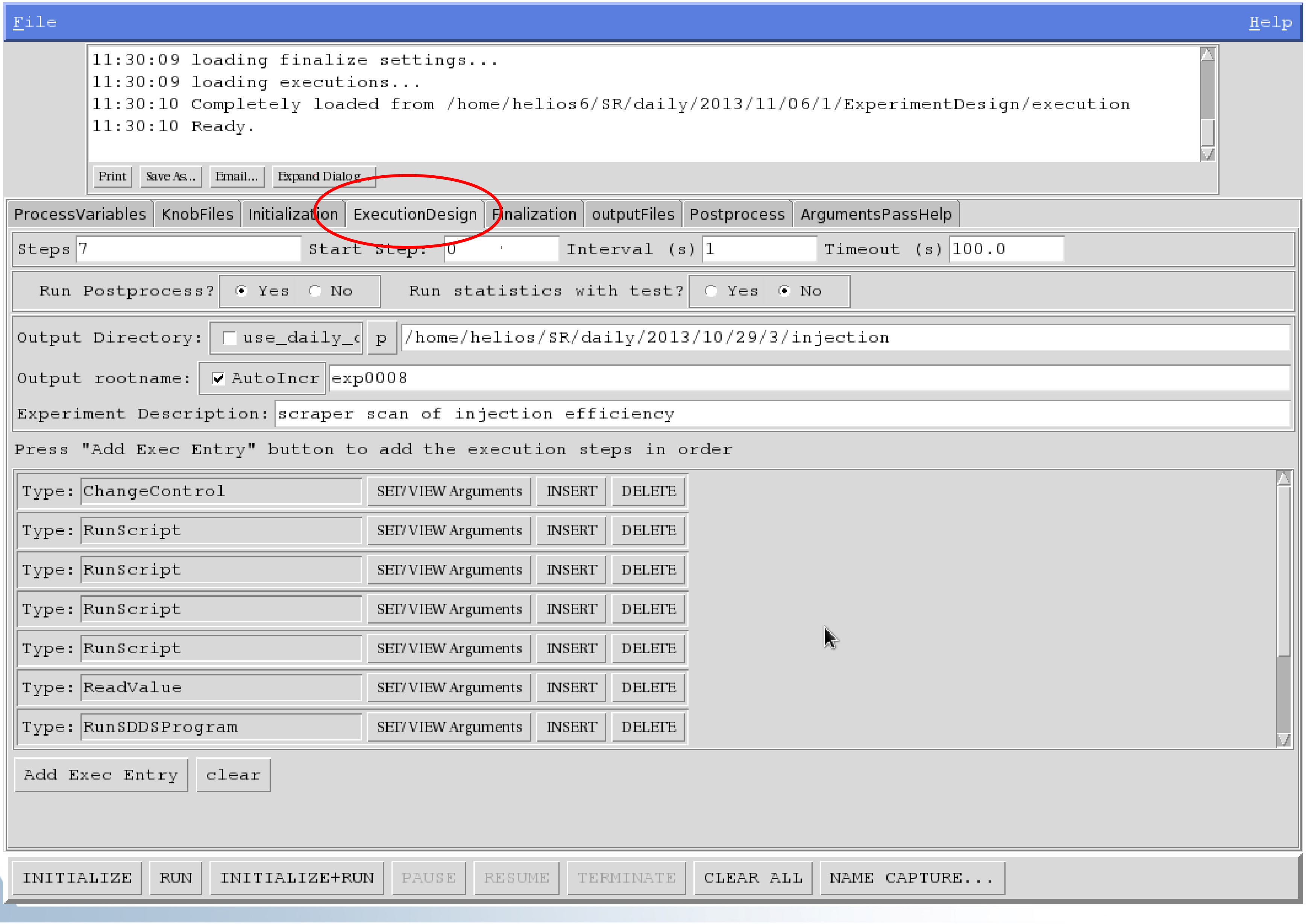
PV name	Readback name	set_value	orig_value	tolerance
S38:VTSC:SM:sm	S38:VTSC:SM:sm.RBV	2.2	2.2	0.01

Script: exec sddsmakedataset /home/helios/SR/daily/2013/10/29/3/injection/injEffic

Script: exec sddspcas /home/helios/SR/daily/2013/10/29/3/injection/injEfficiency.p

Script: exec /home/helios/SR/daily/2013/10/29/3/injection/waitForScraper

Experiment Designer



```
11:30:09 loading finalize settings...
11:30:09 loading executions...
11:30:10 Completely loaded from /home/helios6/SR/daily/2013/11/06/1/ExperimentDesign/execution
11:30:10 Ready.
```

Steps 7 Start Step: 0 Interval (s) 1 Timeout (s) 100.0

Run Postprocess? ● Yes ○ No Run statistics with test? ○ Yes ● No

Output Directory: □ use_daily_c p /home/helios/SR/daily/2013/10/29/3/injection

Output rootname: ☑ AutoIncr exp0008

Experiment Description: scraper scan of injection efficiency

Press "Add Exec Entry" button to add the execution steps in order

Type: ChangeControl

Type: RunScript

Type: RunScript

Type: RunScript

Type: RunScript

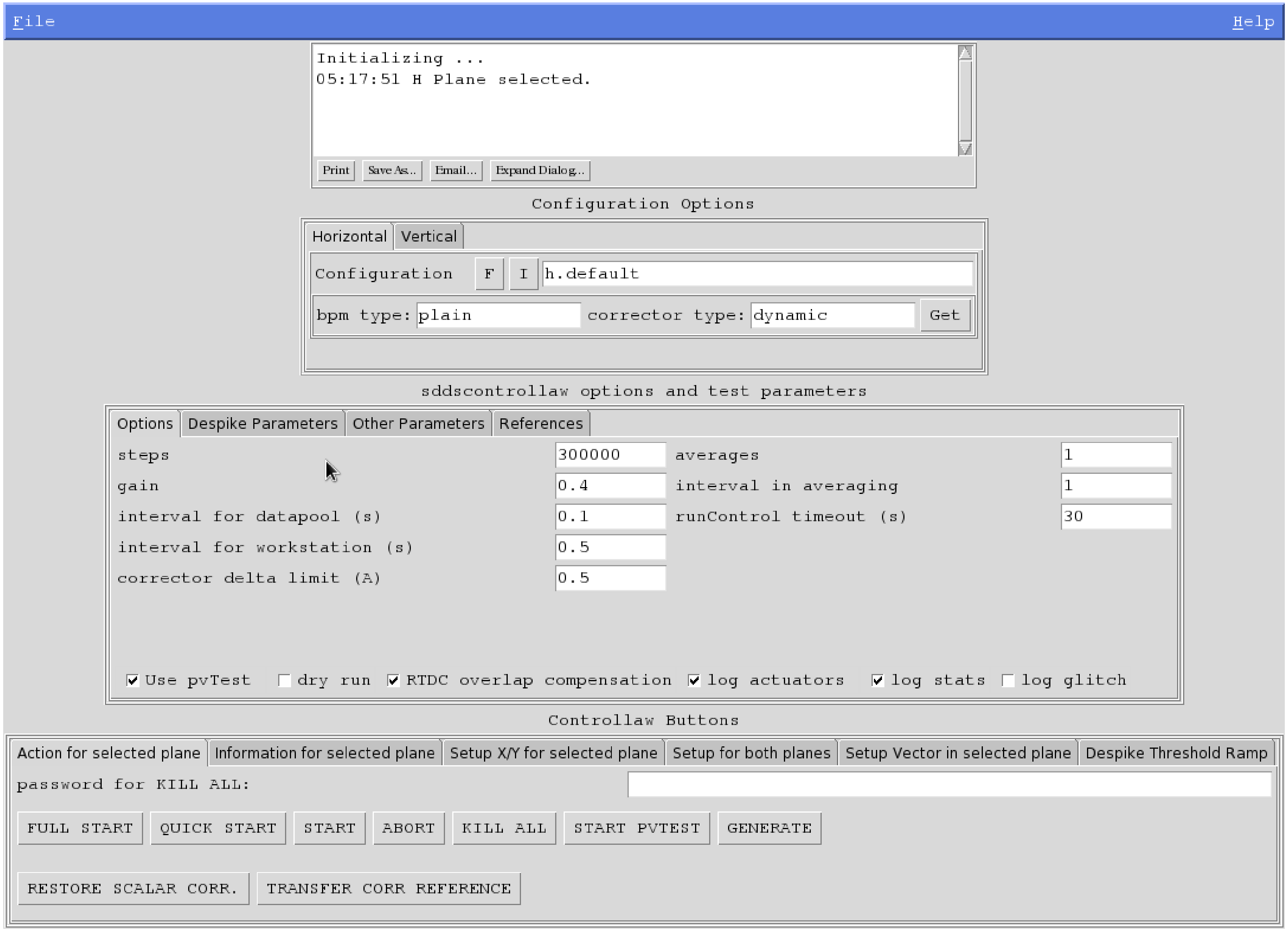
Type: ReadValue

Type: RunSDDSProgram

Other Tcl Applications

- See following slides

Orbit Correction Launcher



```
Initializing ...
05:17:51 H Plane selected.
```

Configuration Options

Configuration: h.default

bpm type: plain   corrector type: dynamic

sddscontrollaw options and test parameters

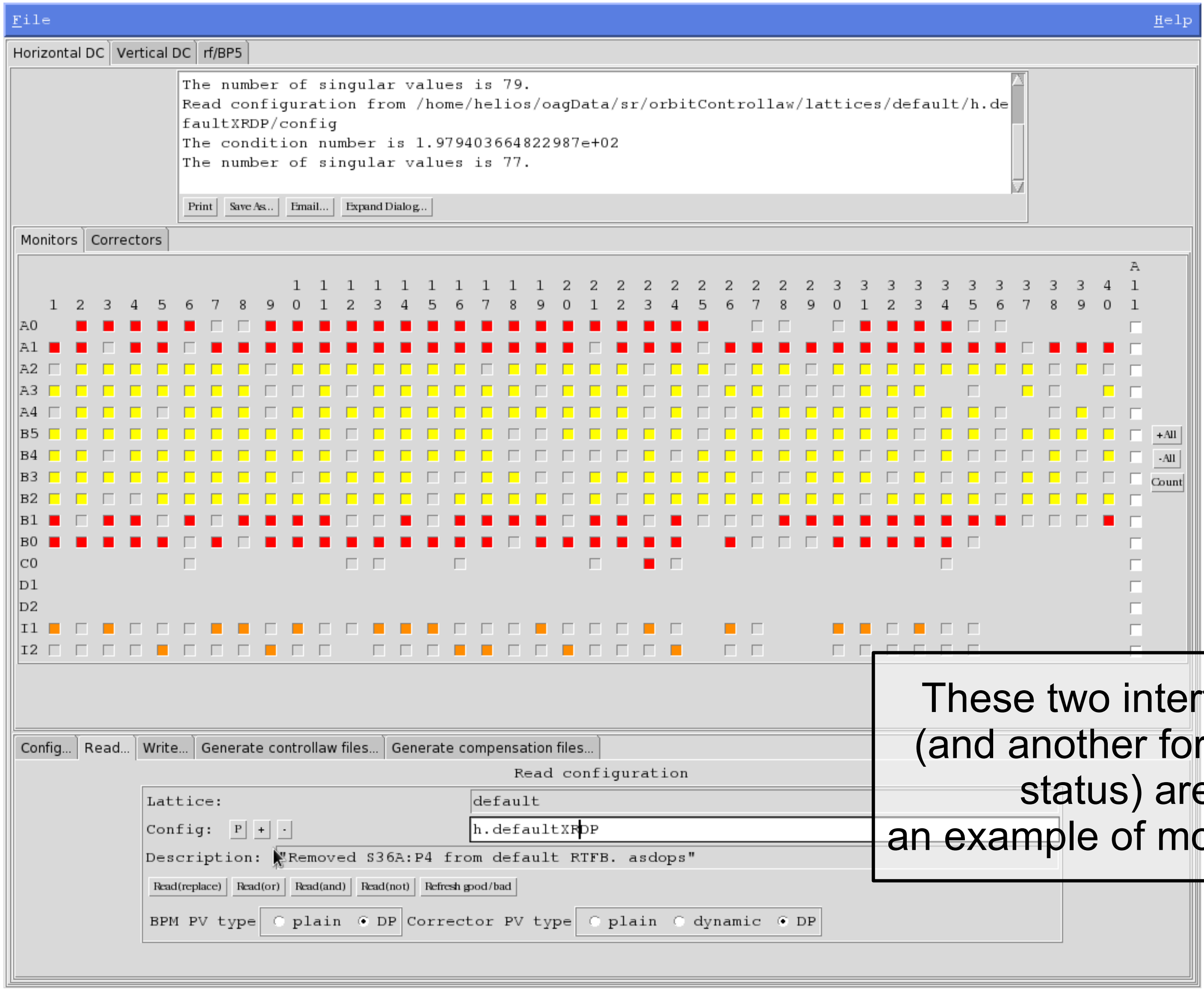
steps	300000	averages	1
gain	0.4	interval in averaging	1
interval for datapool (s)	0.1	runControl timeout (s)	30
interval for workstation (s)	0.5		
corrector delta limit (A)	0.5		

☑ Use pvTest □ dry run ☑ RTDC overlap compensation ☑ log actuators ☑ log stats □ log glitch

Controllaw Buttons

password for KILL ALL:

Orbit Correction Configuration



```
The number of singular values is 79.
Read configuration from /home/helios/oagData/sr/orbitControllaw/lattices/default/h.de
faultXRDP/config
The condition number is 1.979403664822987e+02
The number of singular values is 77.
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	A11
A0		🟥	🟥	🟥	🟥	🟥	□	□	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥		□	□		□	🟥	🟥	🟥	🟥	□	□					□
A1	🟥	🟥	□	🟥	🟥	□	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	□	🟥	🟥	🟥	□	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	□	🟥	🟥	🟥	□
A2	□	🟨	🟨	🟨	🟨	🟨	🟨	🟨	□	🟨	🟨	🟨	🟨	🟨	🟨	🟨	□	🟨	🟨	🟨	🟨	🟨	□	🟨	🟨	□	🟨	🟨	□	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	□	🟨	□	□
A3	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	□	□	🟨	□	🟨	🟨	🟨	🟨	🟨	🟨	□	🟨	🟨	🟨	□	🟨	□	🟨	🟨	□	□	□	🟨	🟨	🟨		□		🟨	□		🟨	□
A4	□	🟨	🟨	🟨	□	🟨	🟨	🟨	□	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	□	🟨	□	□	🟨	🟨	🟨	🟨	🟨	🟨	□	🟨	🟨	□		□	🟨	□	□
B5	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	□	🟨	🟨	🟨	🟨	🟨	□	□	🟨	🟨	🟨	🟨	🟨	□	🟨	🟨	🟨	🟨	🟨	🟨	🟨	□	🟨	🟨	□	🟨	🟨	🟨	🟨	□
B4	🟨	🟨	□	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	□	🟨	🟨	🟨	🟨	🟨	□	□	□	□	□	🟨	□	🟨	🟨	🟨	🟨	🟨	🟨	□	🟨	□	🟨	□	□	□	🟨	□	🟨	□
B3	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	□	🟨	🟨	🟨	🟨	□	□	🟨	🟨	□	□	🟨	🟨	🟨	🟨	🟨	🟨	□	□	🟨	🟨	🟨	□	🟨	□	🟨	□	🟨	🟨	□	□	□
B2	🟨	🟨	□	□	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	□	🟨	🟨	🟨	🟨	□	🟨	□	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	□	🟨	🟨	🟨	🟨	□	🟨	🟨	🟨	🟨	□
B1	🟥	□	🟥	🟥	□	🟥	□	🟥	🟥	🟥	🟥	□	□	🟥	□	🟥	🟥	🟥	🟥	□	🟥	🟥	□	🟥	□	□	□	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	□	□	□	🟥	□
B0	🟥	🟥	🟥	🟥	🟥	□	🟥	□	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	□	🟥	🟥	🟥	🟥	🟥	🟥		🟥	□	□	□	🟥	🟥	🟥	🟥	🟥	□						□
C0						□						□	□			□					□		🟥	□										□							□
D1																																									□
D2																																									□
I1	🟧	□	🟧	□	□	□	🟧	🟧	□	🟧	□	□	🟧	🟧	🟧	□	□	□	🟧	□	□	□	🟧	□		🟧	□			🟧	🟧	□	🟧	□	□						□
I2	□	□	□	□	🟧	□	□	□	🟧	□	□		□	□	□	🟧	🟧	□	□	🟧	□	□	□	🟧		□	□			□	□	□	□	□	□						□

Read configuration

Lattice: default

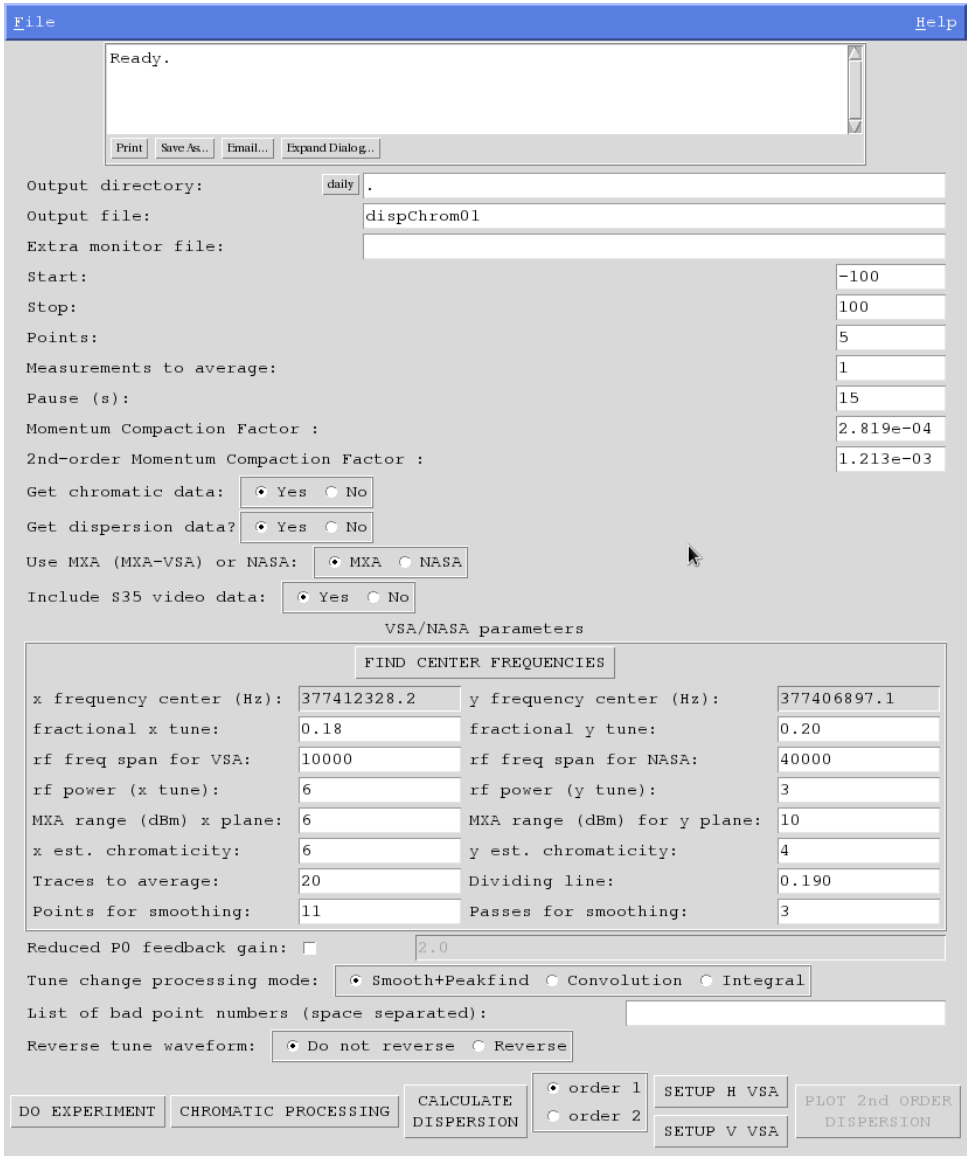
Config: P + - h.defaultXRDP

Description: "Removed S36A:P4 from default RTFB. asdops"

BPM PV type ○ plain ● DP Corrector PV type ○ plain ○ dynamic ● DP

These two interfaces (and another for BPM status) are an example of modularity

Dispersion and Chromaticity Measurement



Ready.

Output directory: .

Output file: dispChrom01

Extra monitor file:

Start: -100

Stop: 100

Points: 5

Measurements to average: 1

Pause (s): 15

Momentum Compaction Factor : 2.819e-04

2nd-order Momentum Compaction Factor : 1.213e-03

Get chromatic data: ● Yes ○ No

Get dispersion data? ● Yes ○ No

Use MXA (MXA-VSA) or NASA: ● MXA ○ NASA

Include S35 video data: ● Yes ○ No

VSA/NASA parameters

x frequency center (Hz):	377412328.2	y frequency center (Hz):	377406897.1
fractional x tune:	0.18	fractional y tune:	0.20
rf freq span for VSA:	10000	rf freq span for NASA:	40000
rf power (x tune):	6	rf power (y tune):	3
MXA range (dBm) x plane:	6	MXA range (dBm) for y plane:	10
x est. chromaticity:	6	y est. chromaticity:	4
Traces to average:	20	Dividing line:	0.190
Points for smoothing:	11	Passes for smoothing:	3

Reduced P0 feedback gain: □ 2.0

Tune change processing mode: ● Smooth+Peakfind ○ Convolution ○ Integral

List of bad point numbers (space separated):

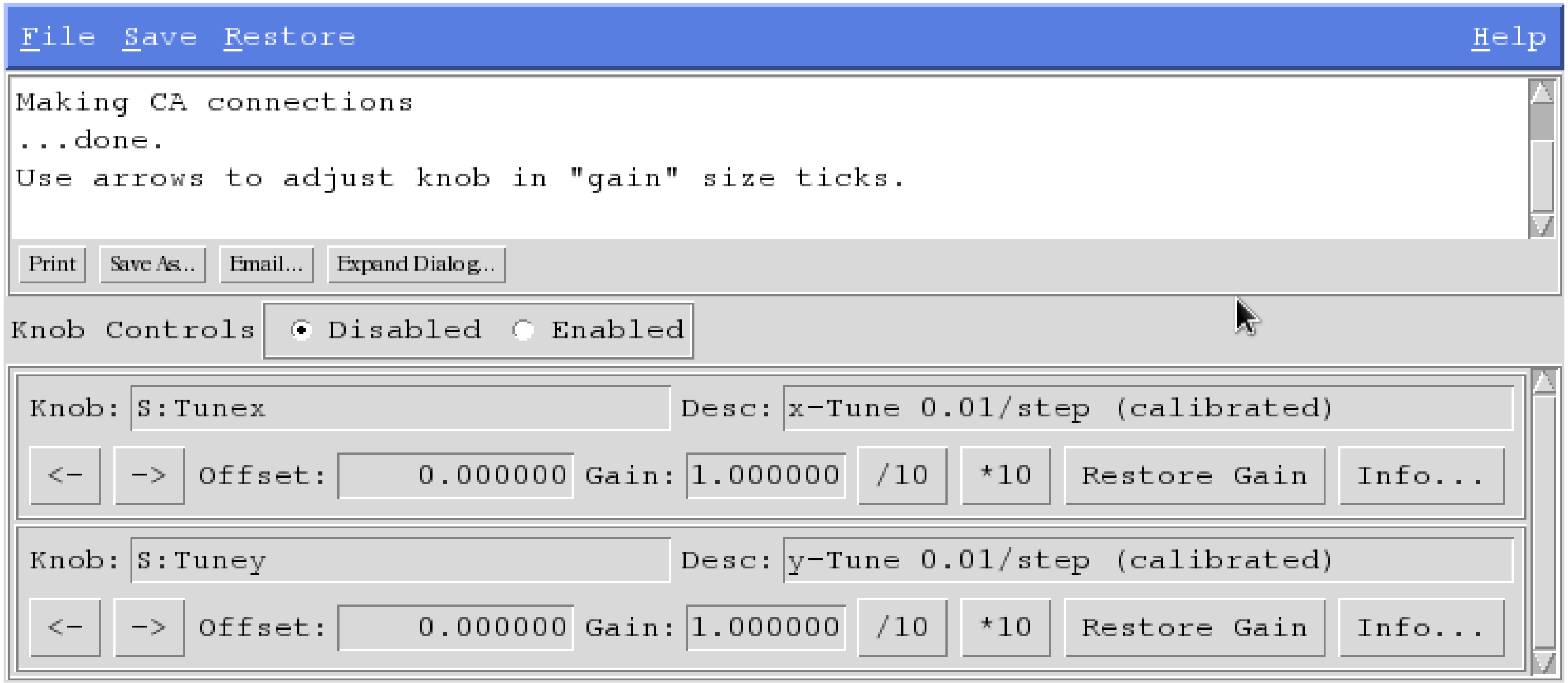
Reverse tune waveform: ● Do not reverse ○ Reverse

● order 1 ○ order 2

Control of a network analyzer and rf frequency

Tcl knobs for SR Tune

- Use File menu to configure the knobs to a different set of PVs



```
Making CA connections
...done.
Use arrows to adjust knob in "gain" size ticks.
```

Knob Controls ● Disabled ○ Enabled

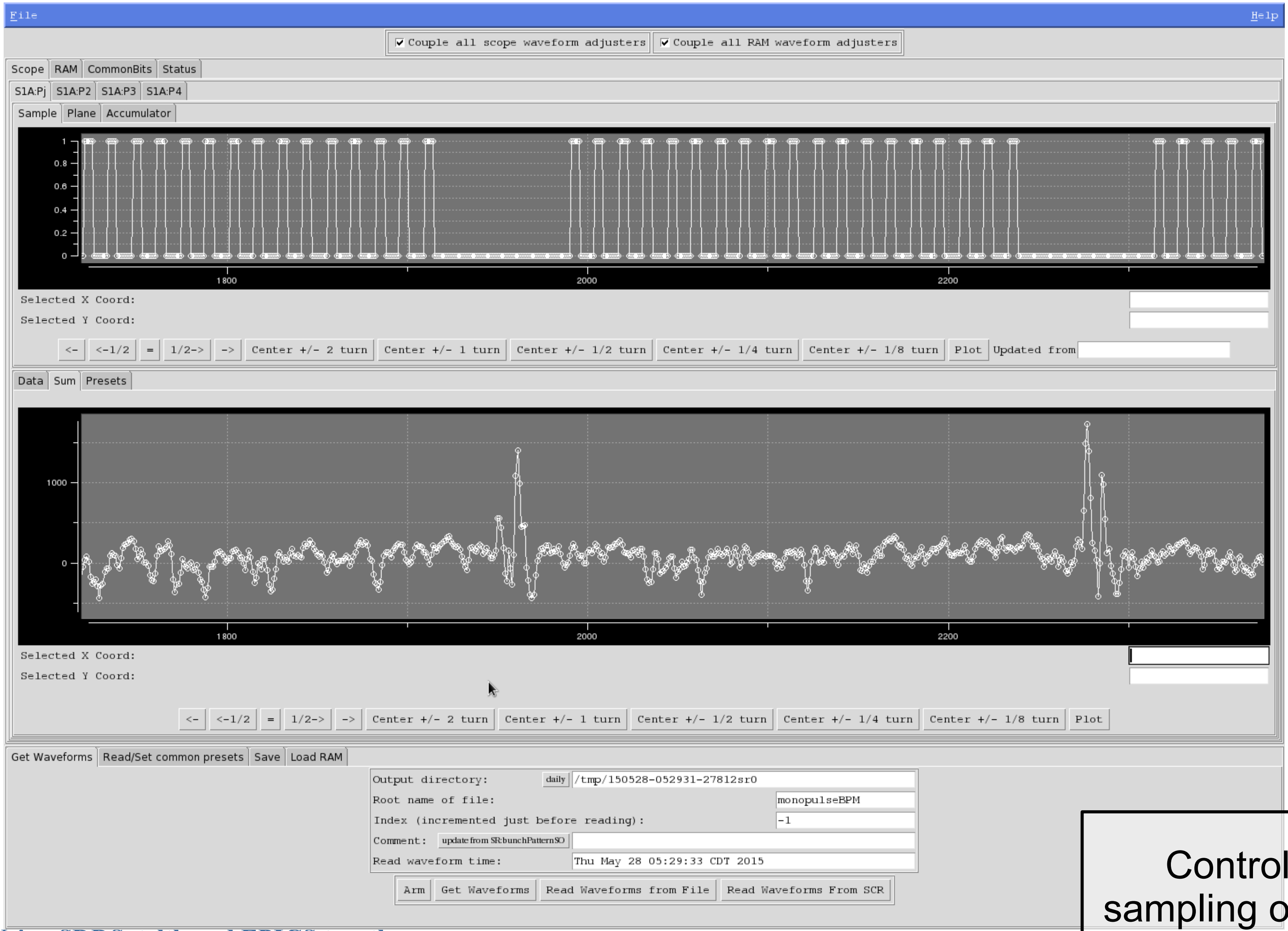
Knob: S:Tunex Desc: x-Tune 0.01/step (calibrated)

<- -> Offset: 0.000000 Gain: 1.000000 /10 *10 Restore Gain Info...

Knob: S:Tuney Desc: y-Tune 0.01/step (calibrated)

<- -> Offset: 0.000000 Gain: 1.000000 /10 *10 Restore Gain Info...

BSP-100 BPM Control Waveform Viewer



☑ Couple all scope waveform adjusters ☑ Couple all RAM waveform adjusters

Selected X Coord:

Selected Y Coord:

Updated from

Selected X Coord:

Selected Y Coord:

Output directory: /tmp/150528-052931-27812sr0

Root name of file: monopulseBPM

Index (incremented just before reading): -1

Comment:

Read waveform time: Thu May 28 05:29:33 CDT 2015

Control of sampling of bpms

Conclusions and Comments

- Complex applications have been constructed using Tcl/Tk working with EPICS and with SDDS and EPICS tools
- All this could have been done similarly with other scripting languages. I find Tcl/Tk (espeically other's code) cleaner to read.
- Other labs have used MATLAB in the role of Tcl/Tk, and instead of using SDDS toolkit, they have used MATLAB functions.
  - SDDS extension is available in MATLAB, BTW.