TINE Services

Part I. Background Information

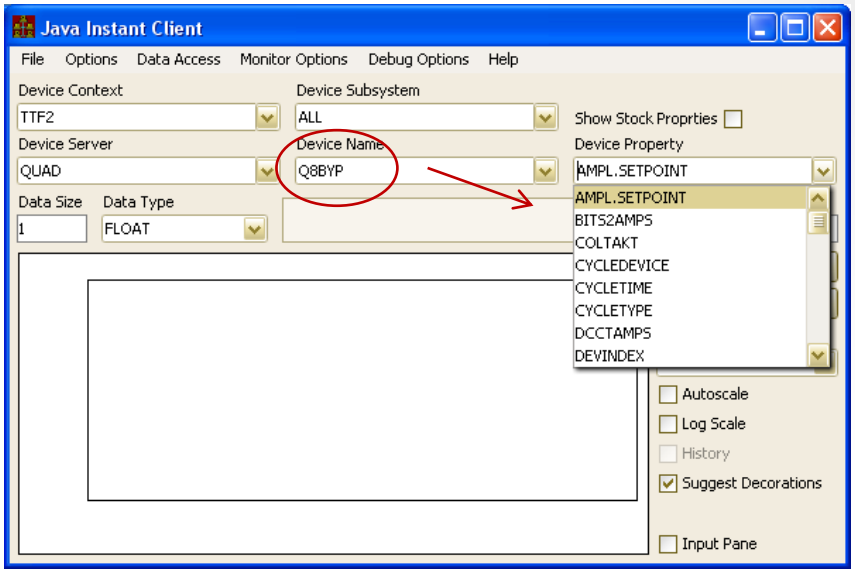
- Control elements are *distributed*
  - Many (disparate) platforms
    - Windows, Unix, VxWorks, java, (DOS , VMS, NIOS)
  - And Frameworks
    - .NET, LabView, MatLab, Python, Perl
    - Device Servers and Property Servers
    - Servers with device instances, Services, …
  - Client Applications
    - Native
    - Frameworks: (jddd, CSS, COMA)
    - RAD tools: ACOP

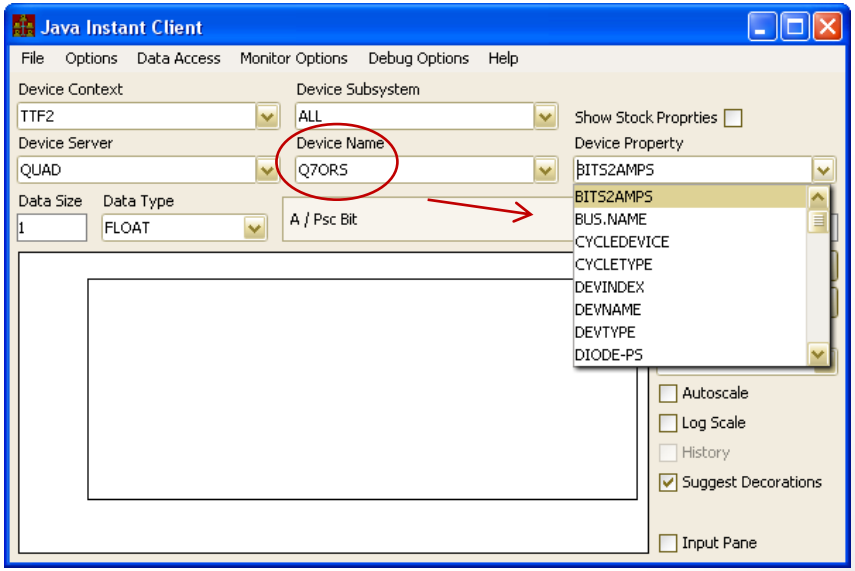
FECs

- *Front End Controller* (a process)
- Unique *address* (IP + port offset)
- Unique *name*
  - Not 'visible' to most API routines.
- Platforms *with virtual memory* (Win32, Win64, Unix, VMS)
  - Many FECs per host possible
    - Same IP, different port offsets
- Platforms *without virtual memory* (DOS, Win16, VxWorks)
  - One FEC per host
- Each FEC can have many *Equipment Modules* (EQM)
  - 'Equipment Module' is what is exported as a 'Server'
    - Has a *local* name tag (6-characters, not 'visible')
  - All EQMs share the same address space.
  - Most FECs only have 1 EQM.

Servers

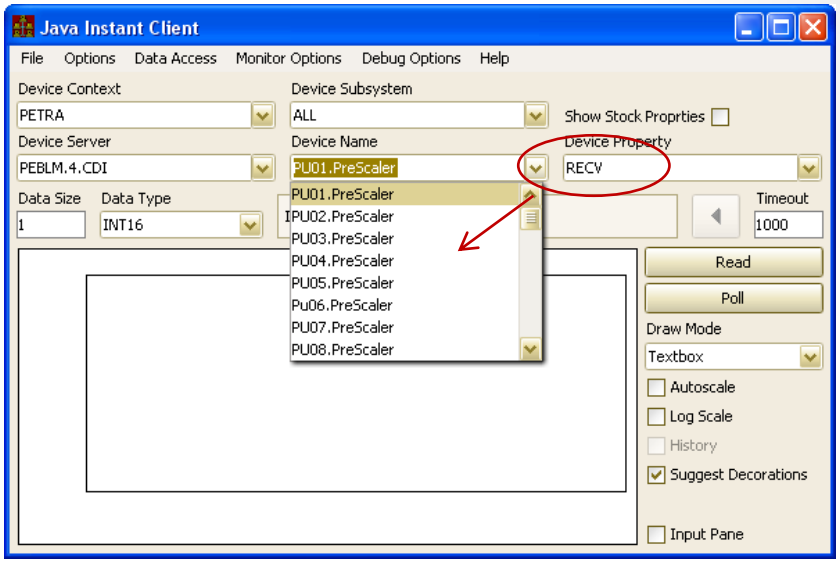
- Device Servers
  - *Instances* of hardware devices
  - Hierarchy is *flat* or *device-oriented*
    - Each device supports a set of properties
  - *The device has properties* !
  - e.g. BPM, BLM, VAC.ION_PUMP, …

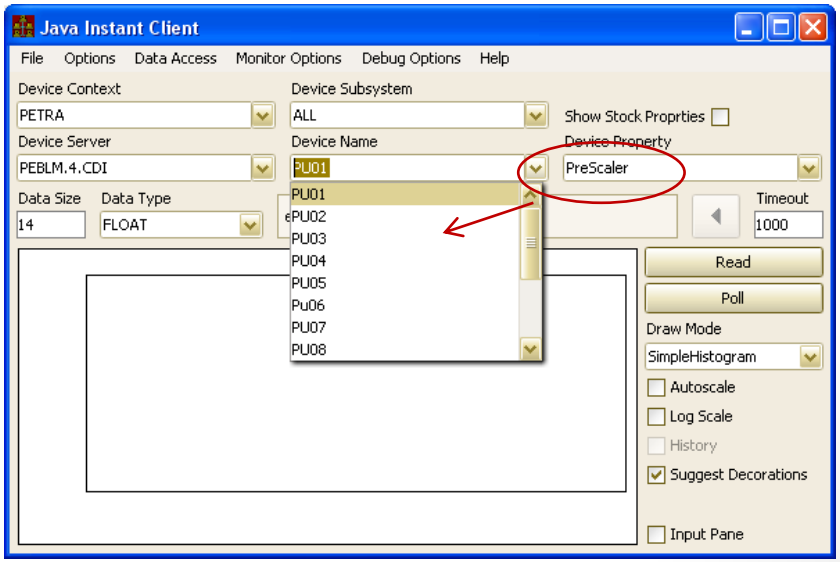




Servers

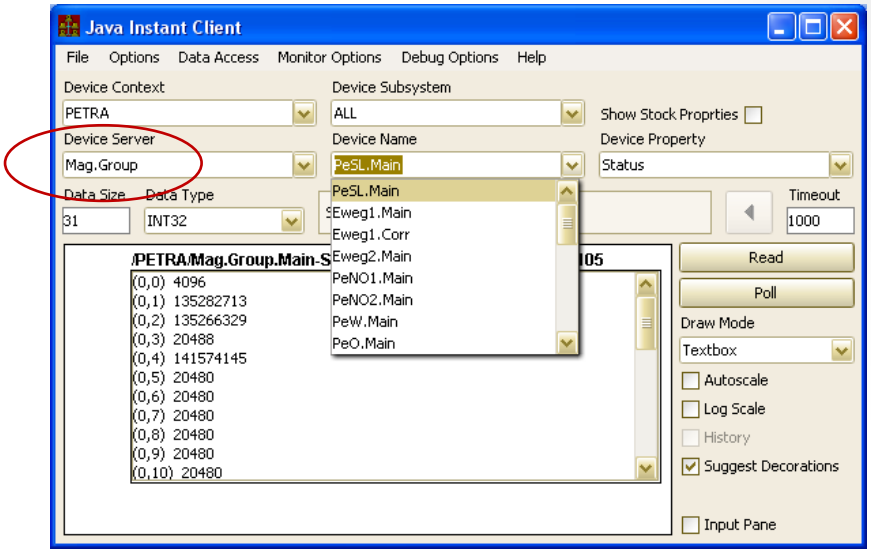
- Property Servers
  - *Services* accessed by properties
  - Hierarchy is *flat* or *property-oriented*
    - Each property supports a set of ‘devices’ (i.e. keywords)
  - *The server has properties* !
  - e.g. ARCHIVER, CAS, any CDI server, …

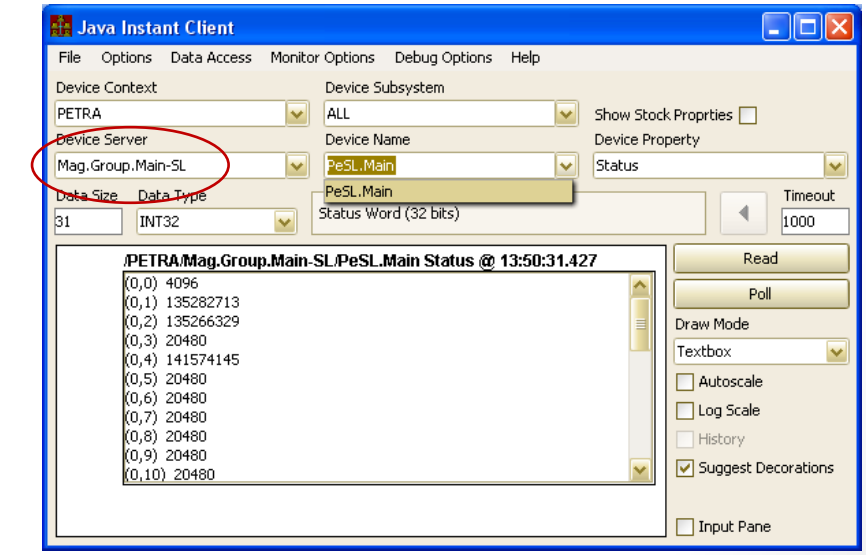




Servers

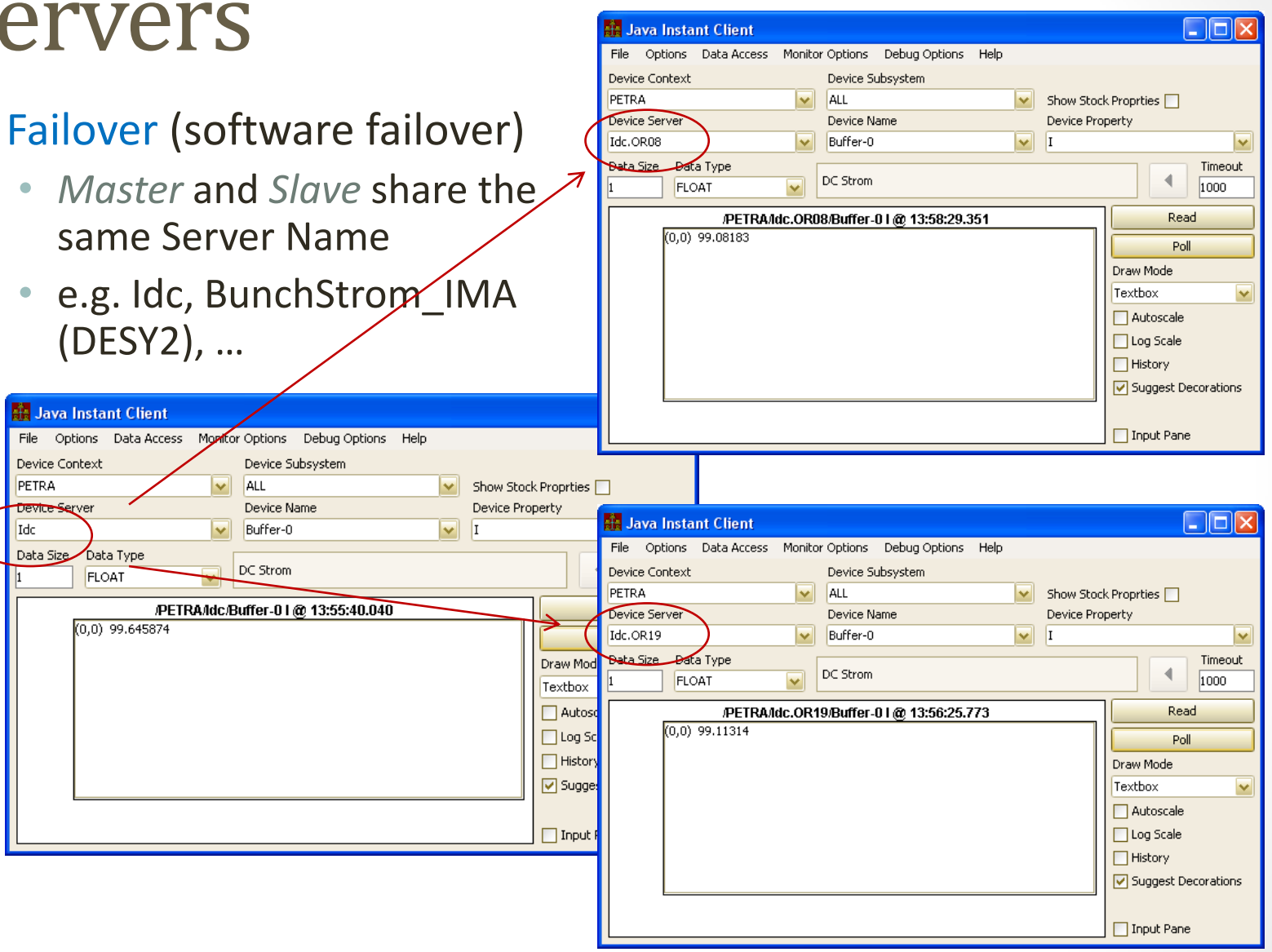
- Logical (i.e. ‘group’) Servers
  - Distributed among *multiple device servers* (members)
  - Each ‘member’ device server has unique set of devices
  - e.g. PiConditions, Mag.Group, …





Servers

- Failover (software failover)
  - *Master* and *Slave* share the same Server Name
  - e.g. Idc, BunchStrom_IMA (DESY2), …



Properties

- Define the '*action*' part of a request to an equipment module
- Are really '*Methods*'
- *Attribute style* properties :
  - READ/WRITE (i.e. get/set) '*something*'
  - Department of Redundancy Department:
    No need to register "rdSomething" and "wrSomething"
- Command style properties :
  - WRITE command initializes action
    - e.g. "RESET", "INIT", "START", "STOP"
- Call style properties :
  - READ (or WRITE or WRITE/READ) involves sending input and receiving output (not necessarily of the same data type)
    - e.g. "CorrectOrbit"
- *Properties can be overloaded*!
  - Registered multiple times with different input/output data.
    - e.g. return simple or detailed information based on requested output data type.

Properties

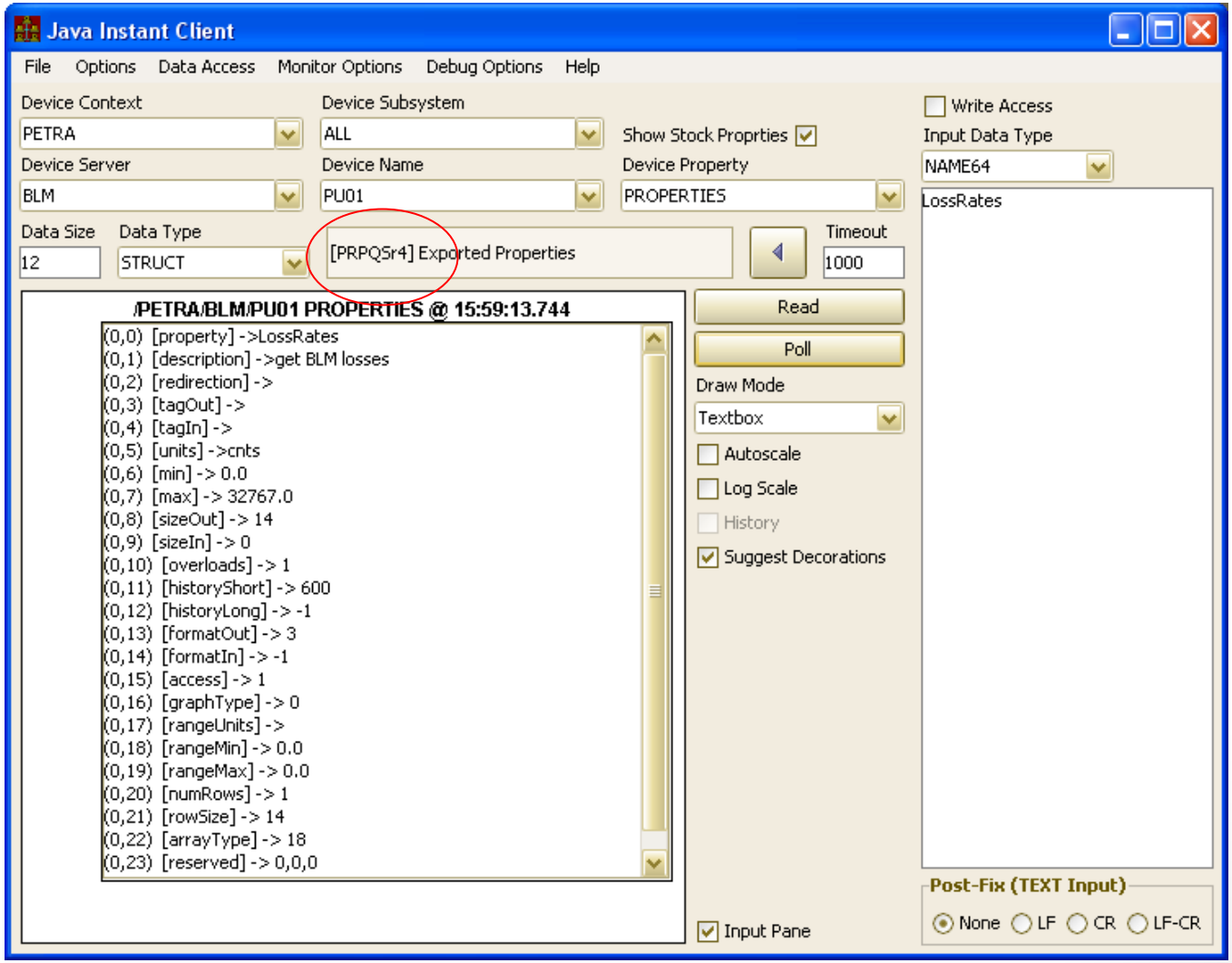
- Need to be registered (along with all meta information) !
  - *Units*, *settings*, *description*, etc. +
  - *Array type* !
    - Make use of multi-channel arrays where possible !
    - **MCA** is NOT the same as multiple single element acquisition or wildcard access.
    - Can play a BIG roll in data archiving !
- Stock Properties
  - All Servers have them!
  - Some are *FEC specific*
    - SRVSTARTTIME, SRVCMDLINE, etc.
  - Some are *EQM specific*
    - ALARMS, CONTRACTS, etc.
    - ACCESSLOCK (quite a useful one!).

Properties

- Have '*meta*' information !
  - Access most information via a property query structure (STRUCT + tag = "PRPQSr4") or
  - Individually via *meta property decorations*:
    - .EGU, .XEGU, .MAX, .MIN, .DESC
  - *Interface* meta properties:
    - .HIST is the interface to retrieve the local history of the property (*if it is being taken*).
    - Also: .BIT.x, .MASK.x, .GATE.x
      - Apply to integer types only
  - *Hidden from general browsing* !

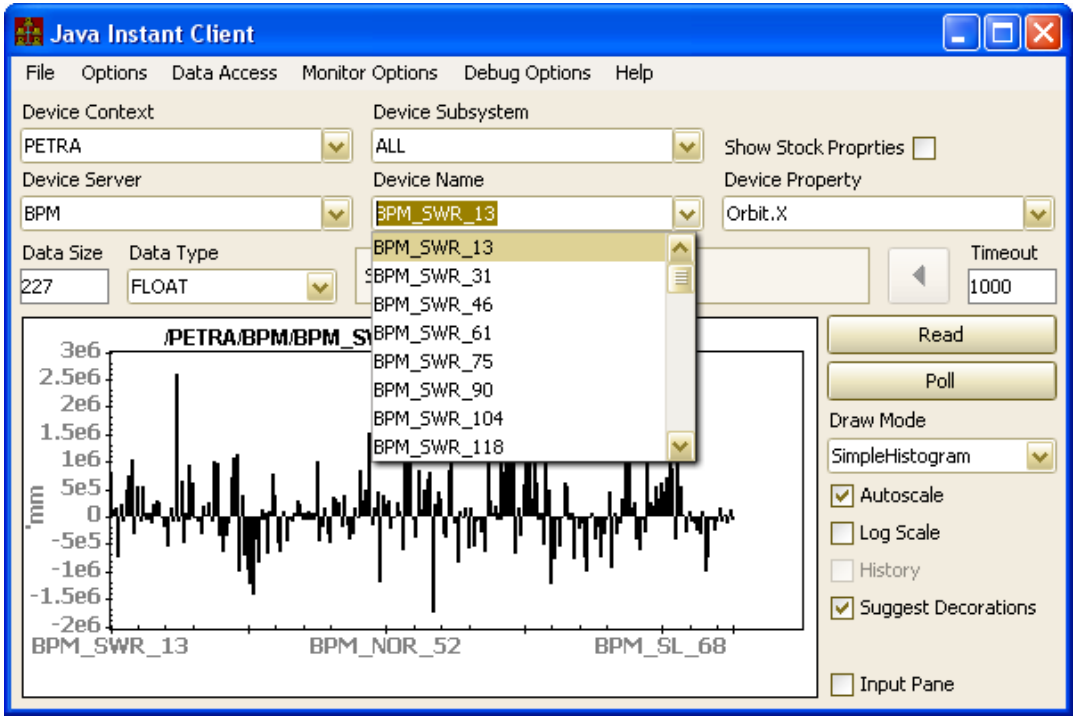
Properties

A property query :



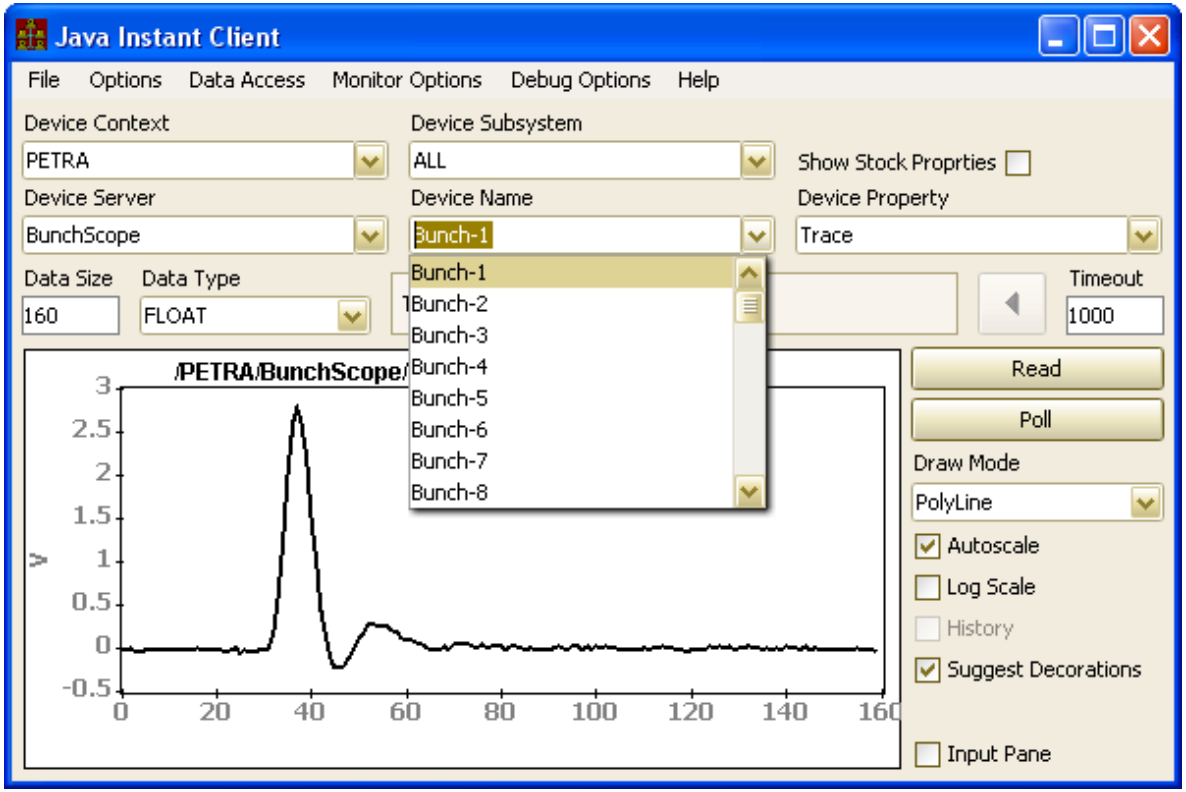
Devices

- *Input device name checked at the EQM* (property handler) !
- Can signify particular *instances* of a hardware module
  - device server view
  - e.g. BPM has devices BPM_SWR_13, BPM_SWR_31, etc.



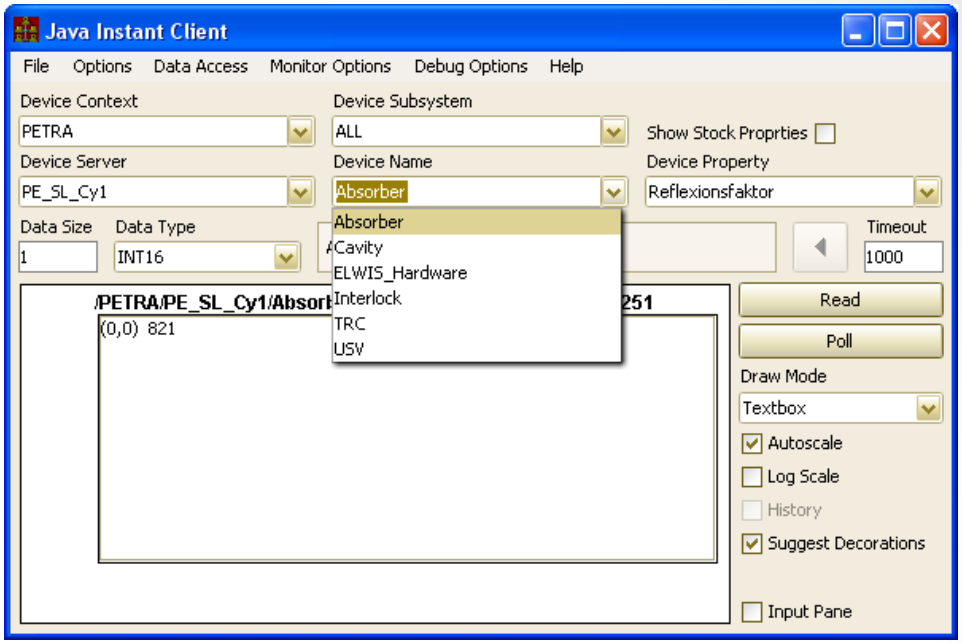
Devices

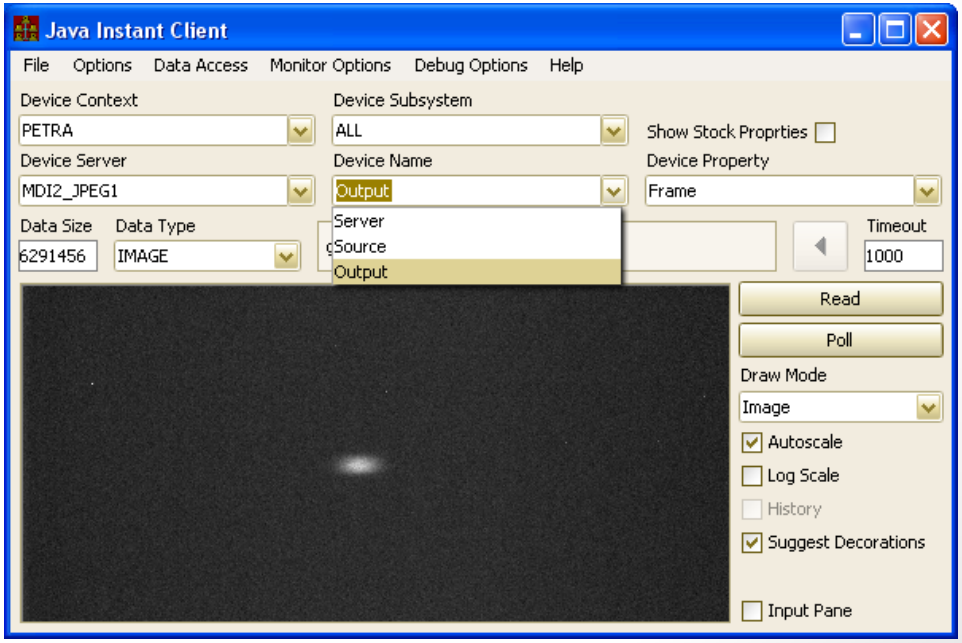
- Can signify particular readout *elements*
  - e.g. BunchScope has devices Bunch-1, Bunch-2, etc.



Devices

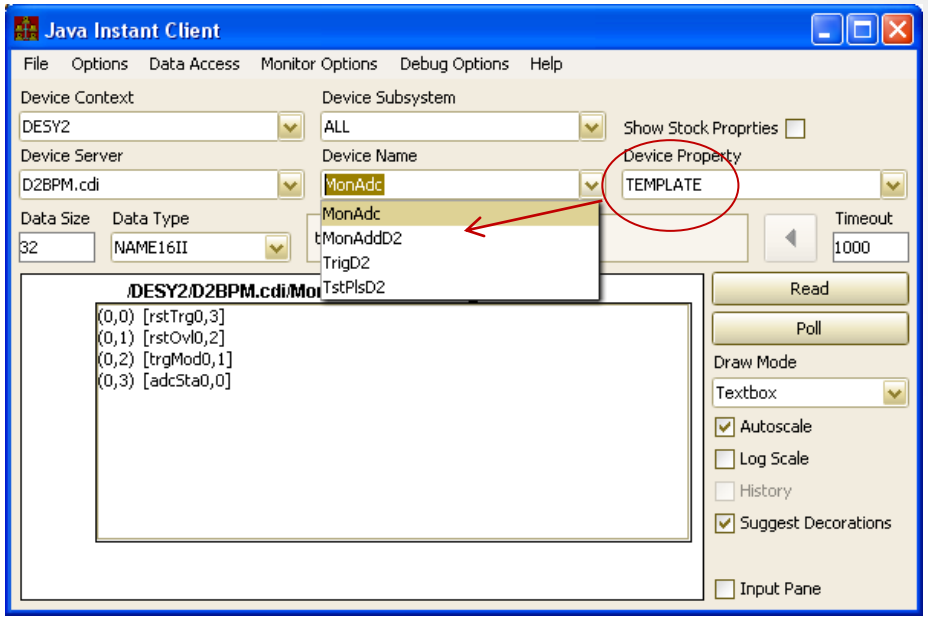
- Can signify different readout *categories*
  - e.g. ELWISes have devices “Absorber”, “Cavity”, “Interlock”, etc.
  - e.g. Video has devices “Server”, “Source”, “Output”, “Adjustments”, etc.

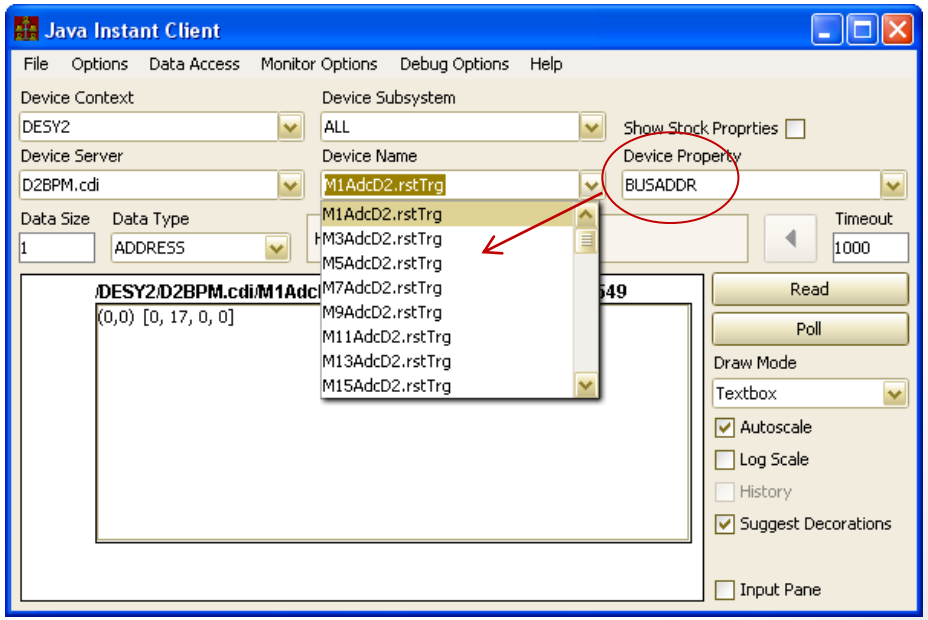




Devices

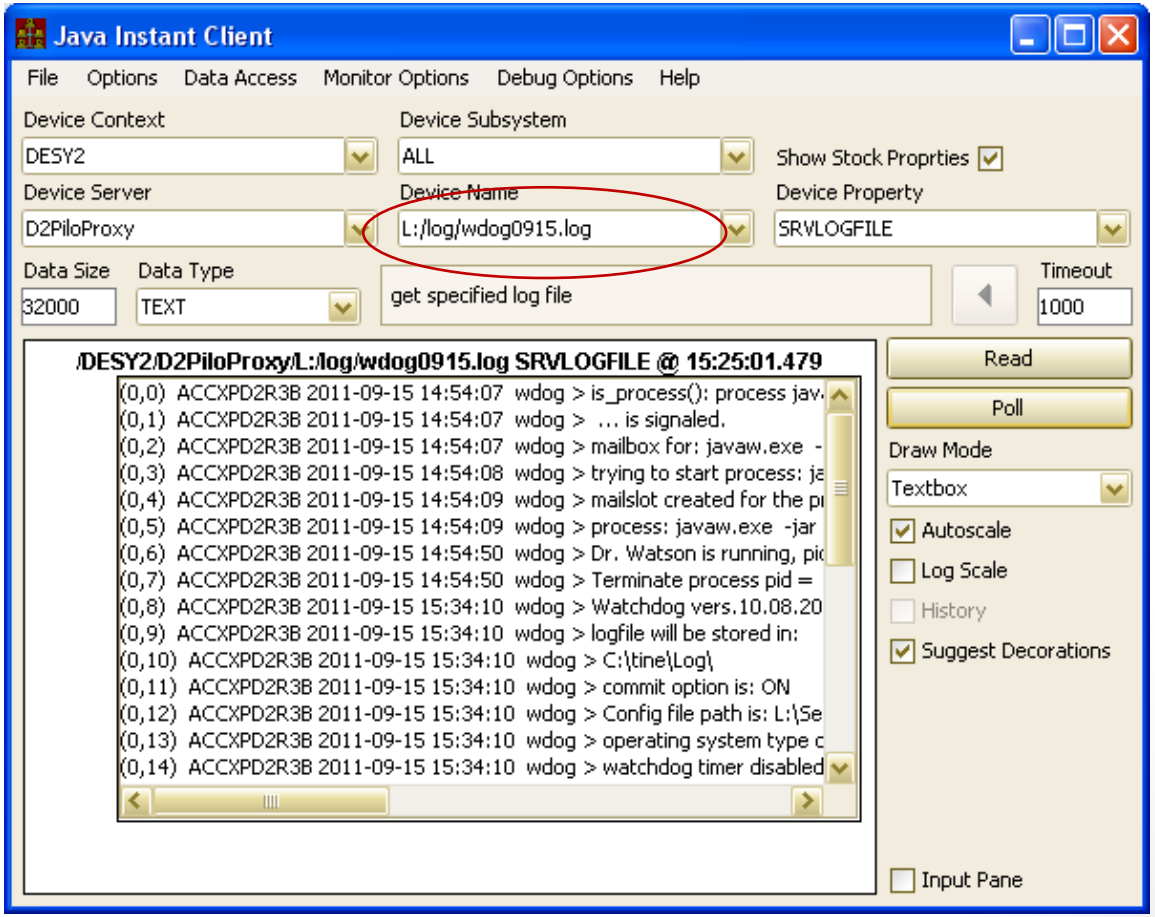
- Can signify property related '*keywords*'
  - Property server view
  - e.g. CDI lists
    - cdi devices for BUS properties (SEND, RECV, etc.)
    - registered Templates for properties TEMPLATE, INSTANCES
    - 'split' device locations for extended properties, etc.





Devices

- A ‘free’ string used as input.
  - e.g. stock property “SRVLOGFILE” takes device ‘name’ as file path and name.



Security

- Based on *User Name* and/or *Network address*
- *Fec Name = User Name* when servers are clients !
- Applies to *WRITE* access and to *exclusive READ* access.
- Usually assigned at the server level, but:
  - Can be applied to *individual Properties*
  - Can be applied to *individual Devices*
- Stock Property "ACCESSLOCK" can be used to assign an *access token* to a single specific client process.
  - Inherent in API calls 'SetAccessLock()', 'FreeAccessLock()', etc.
  - Exclusive READ can also be registered to take effect only when an Access Lock is in play !

Error (status) Codes

- *Are in general NOT exceptions*!
- *Systematic codes* (< 512)
  - Several used at the protocol level
    - '*illegal_protocol*', '*get_subscritption_id*', '*property_is_mca*', …
- *User defined* (>= 512)
- *Can send data + status* !
  - return (CE_SENDATA | status)
  - e.g. 'has_query_function' status is used to signal either property or device query precedence.
  - e.g. 'information_static' applied when polling a property whose data will not change.
  - "*here's the data, but there's something else you should probably know* …."

Part II: Services and Utilities

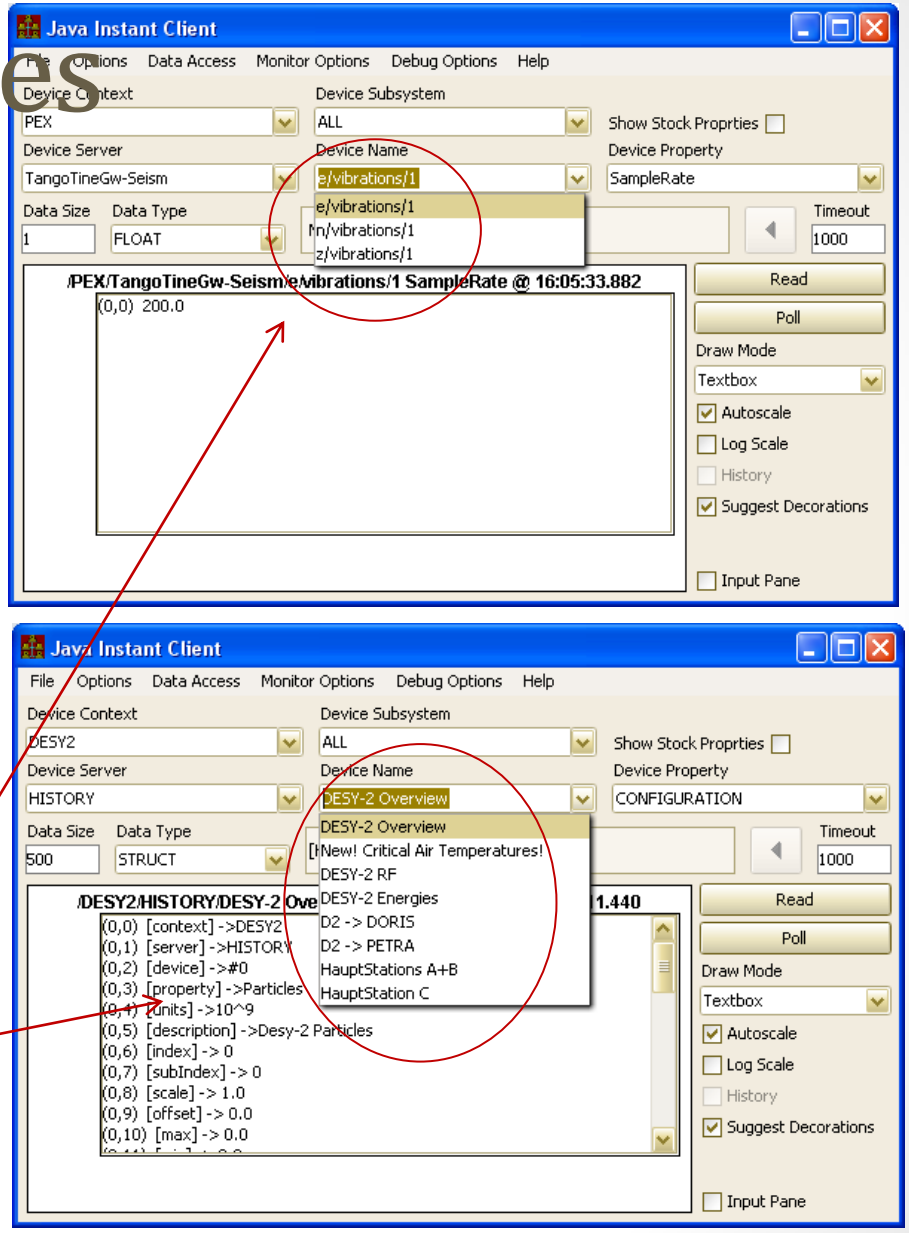
Naming Services

Naming Services

- Hierarchical Naming convention
  - "/*context*/*server*/*device*[*property*]"
  - *subsystem* provide extra (browseable) information
  - *context* and *server* names :
    - <= *32 characters*
    - must begin with *alpha-numeric* character
    - cannot contain '\', '/', or '*'
    - *But please avoid blanks and exotic characters and names like "12345678"* !
  - *context* :
    - can be omitted (if no ambiguity)
    - n.b. "DEFAULT" is NOT a context !
  - *device* name, *property* name :
    - <= *64 characters*
    - No character restrictions
    - *But please avoid blanks and exotic characters* !

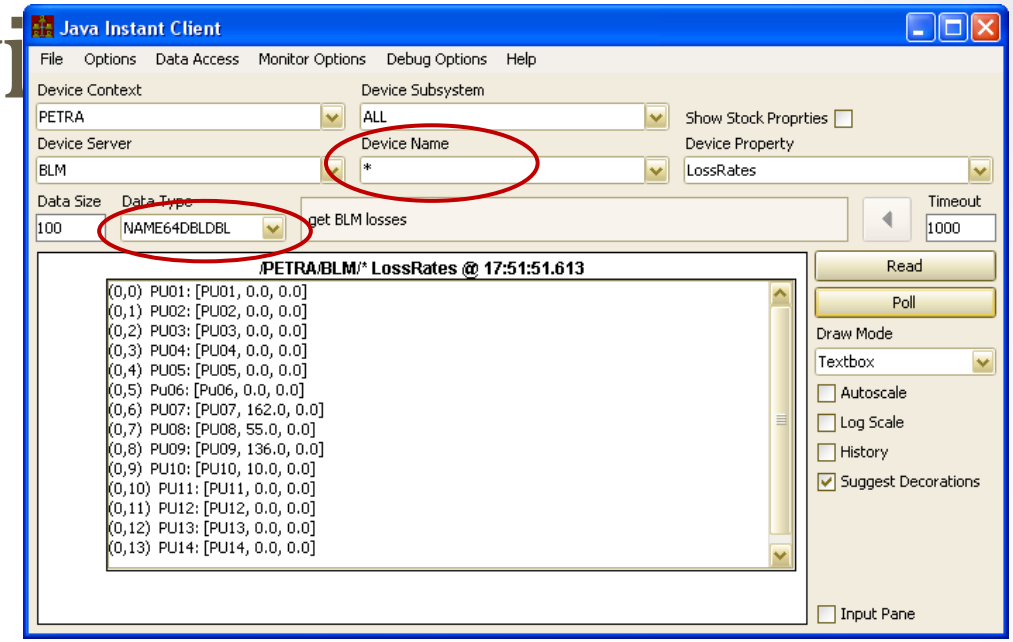
Naming Services

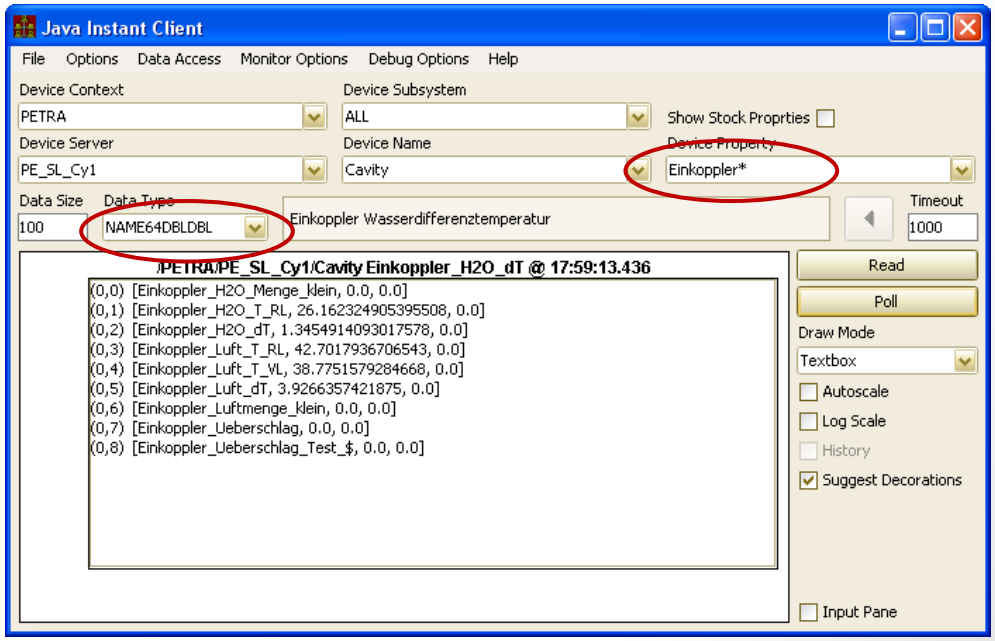
- Hierarchical Naming convention
  - *device name* (further information)
    - *NOT* required to be registered or supplied in call !
    - 64 character *official* limit (for queries, redirection, etc.)
    - can contain *1024 characters* ! (any individual contract)
      - e.g. device = “cdiDev1,cdiDev2,cdiDev3, …”
    - A ‘/’ can sometimes be used to *extend* the hierarchy !
    - ‘blanks’ sometimes not a bad option, after all !



Naming Servi

- **wild cards**
  - e.g. *device* or *property* = “*”, “ABC*”, “*DEF”, ”*CD*”
  - BUT:
    - *don’t know what will come back* !
    - requested data type *must* be able to carry ‘device name’, value, status
      - NAME64DBLDBL, NAME16FLTINT, USTRING, etc.
  - an *MCA property* handles the call as such
  - *else* loops through all devices or properties!





Naming Services

- subsystems
  - <= 16 characters
  - Registered subsystem compared with an *allowed* list !
    - *Not it the list* ? -> no subsystem information !
- *decorated context* + no subsystem (at registration):
  - decoration is used as the subsystem and removed (!) from the context !
  - e.g.

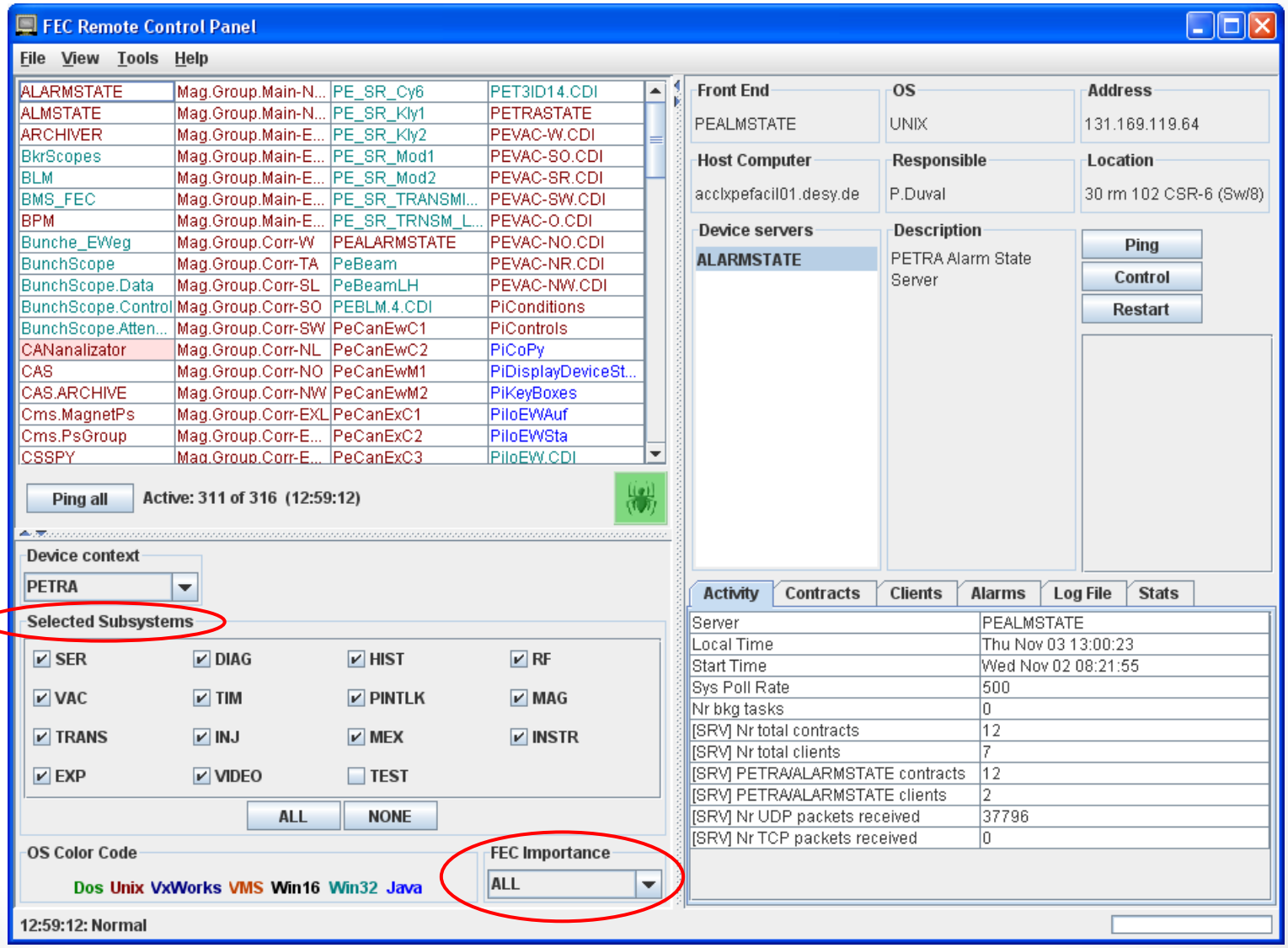
    context = “TTF2.RF”, server = “KLY.INTERLOCK”, subsystem = “”

    -> context = “TTF2”, server = “KLY.INTERLOCK”, subsystem = “RF”
  - Name resolution will honor a request for “/TTF2.RF/KLY.INTERLOCK”
    - “TTF2.RF/KLY.INTERLOCK” and “/TTF2/KLY.INTERLOCK” both map to the same server !
  - Exceptions: context.TEST, context.SIM always allowed.
    - (=> Maybe we can cleanup context.TEST and context_TEST, etc. ?)

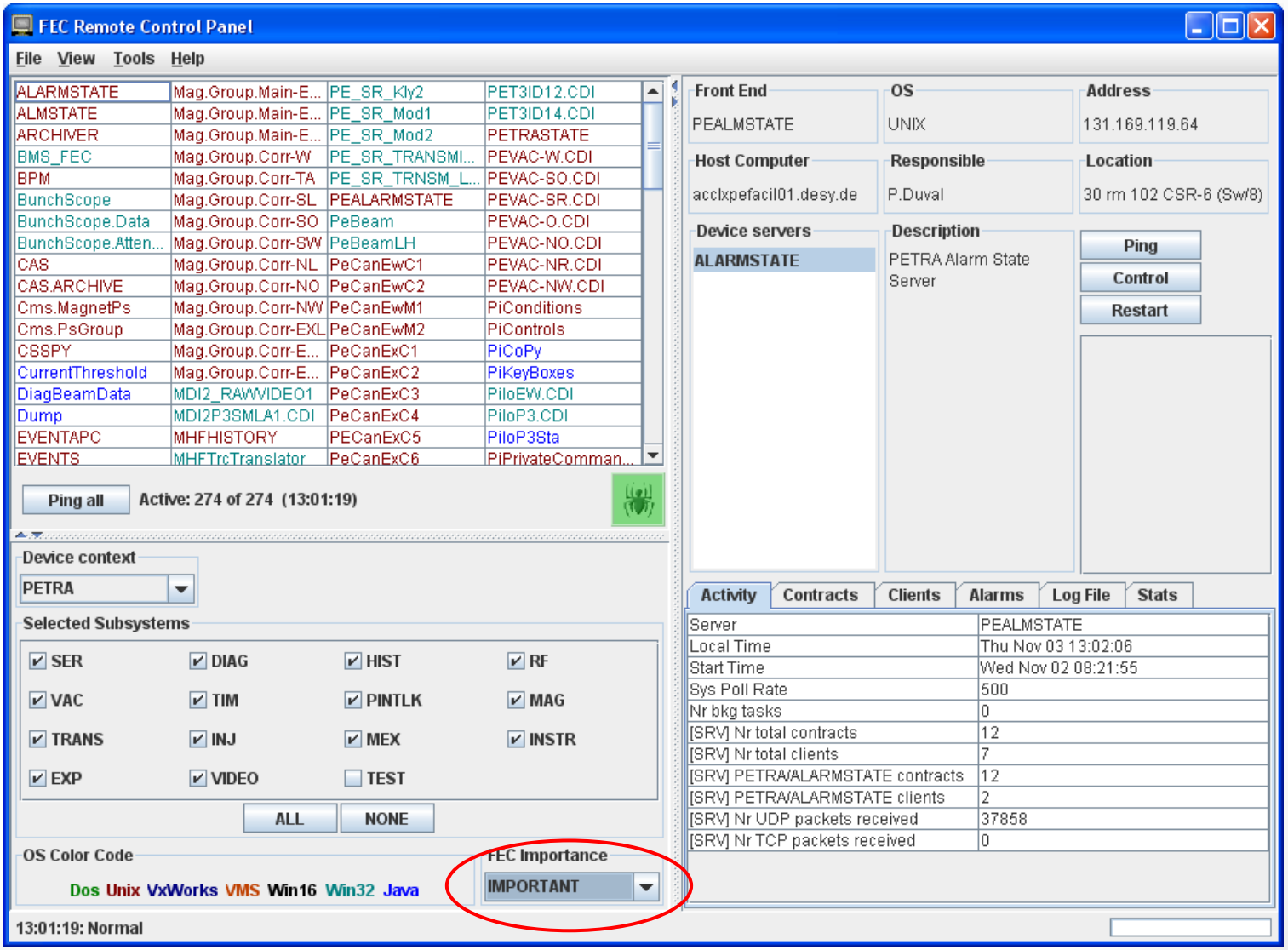
Naming Services

- Clients/Servers use *plug-and-play* !
- Equipment Name Server (ENS) manages control system server database.
  - Context + server -> EQM name + FEC
  - FEC -> address (IP and port)
  - (Properties and devices managed at the server)
  - ANY new server is allowed to plug into the system.
    - (Optionally restrict allowed 'root' contexts)
  - BUT *importance* is administered !
  - On-line status is regularly checked
    - If allowed dead time (default = 3 months) exceeded -> *server is removed* !
- Clients find an address by asking the ENS
  - *ENS down or doesn't know* -> ask local address cache.

Naming Services



Naming Services

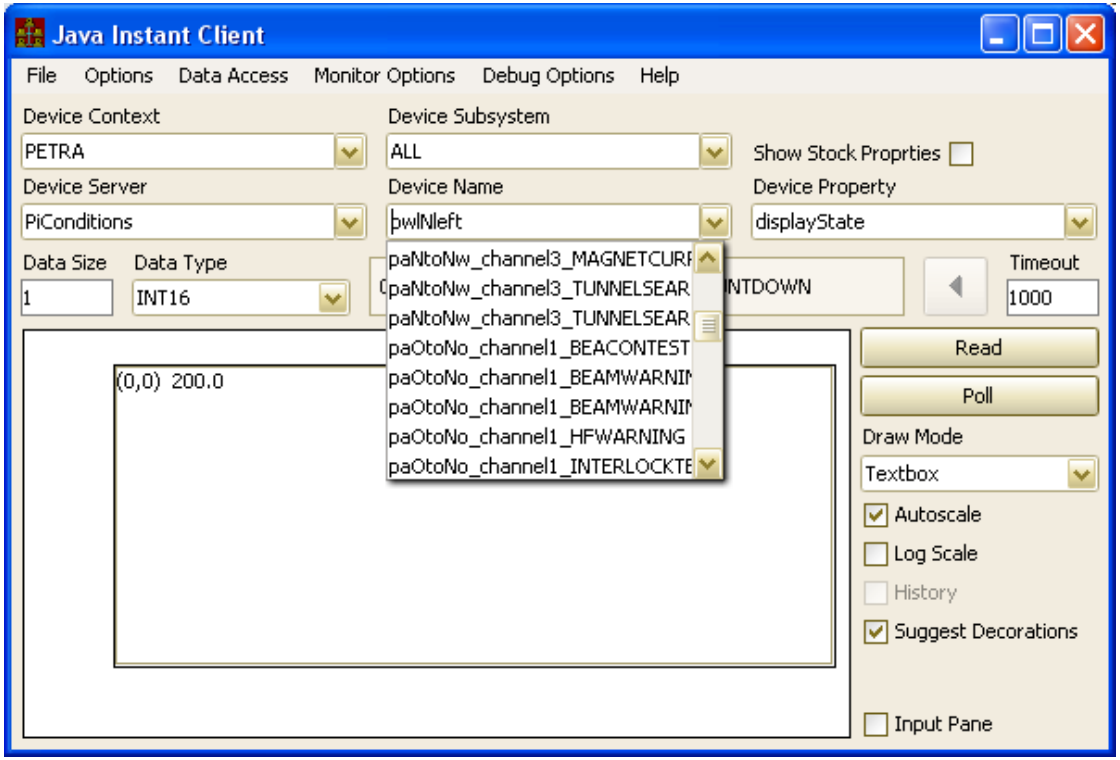


Naming Services

- *How does a client know where the ENS is* ?
  - API call sets the ENS address (takes precedence).
  - environment variable TINE_HOME points to location of file cshosts.csv (list of installed name servers).
  - Ask DNS for address of "tineens" in local domain.
  - Issue a multicast asking for an ENS to respond.

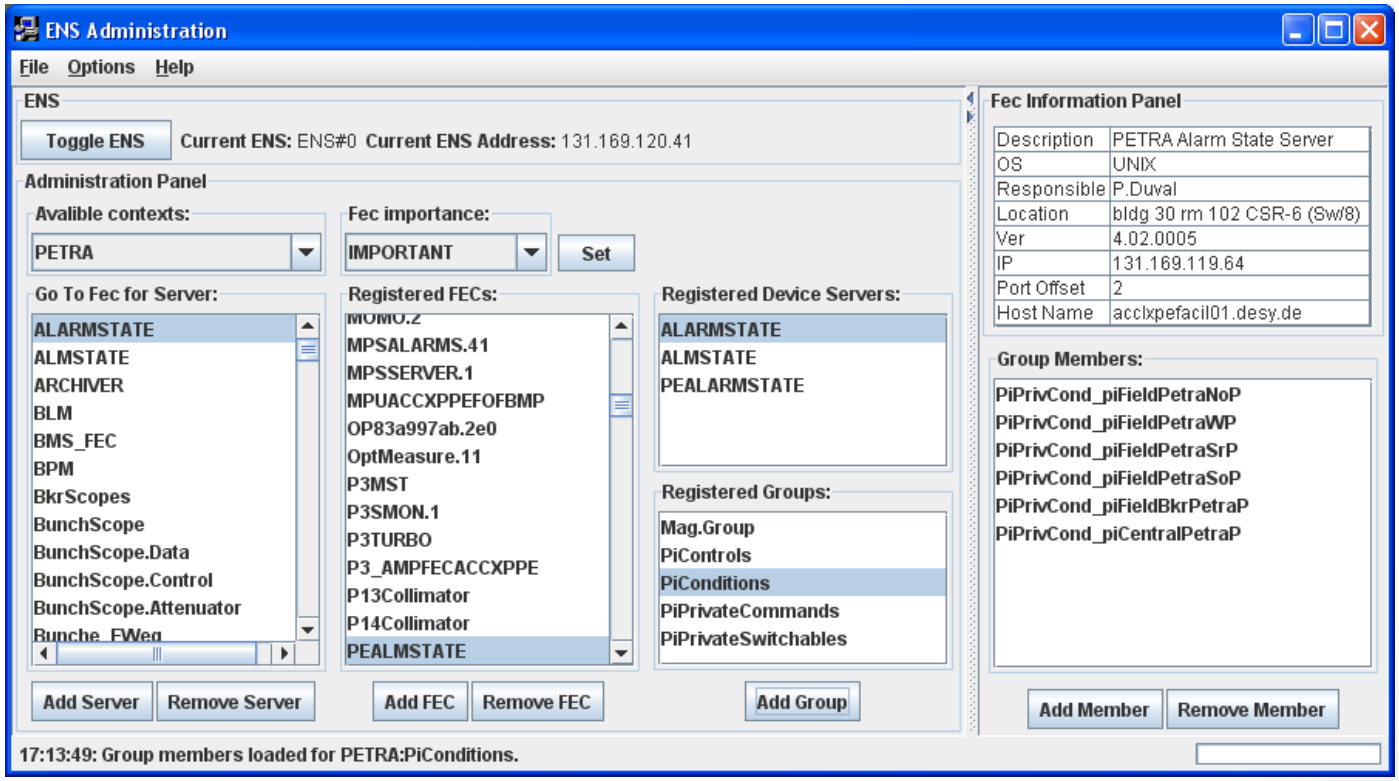
Naming Services

- Group Equipment Name Server (GENS)
  - Companion server to the ENS
  - Manages groups, group members, group device lists.



Naming Services

- ENS administration (who is allowed ?)
  - Registered administrators
  - The FEC's 'responsible' party is allowed to remove the associated FEC.
    - Login name (user name) must match 'responsible'.



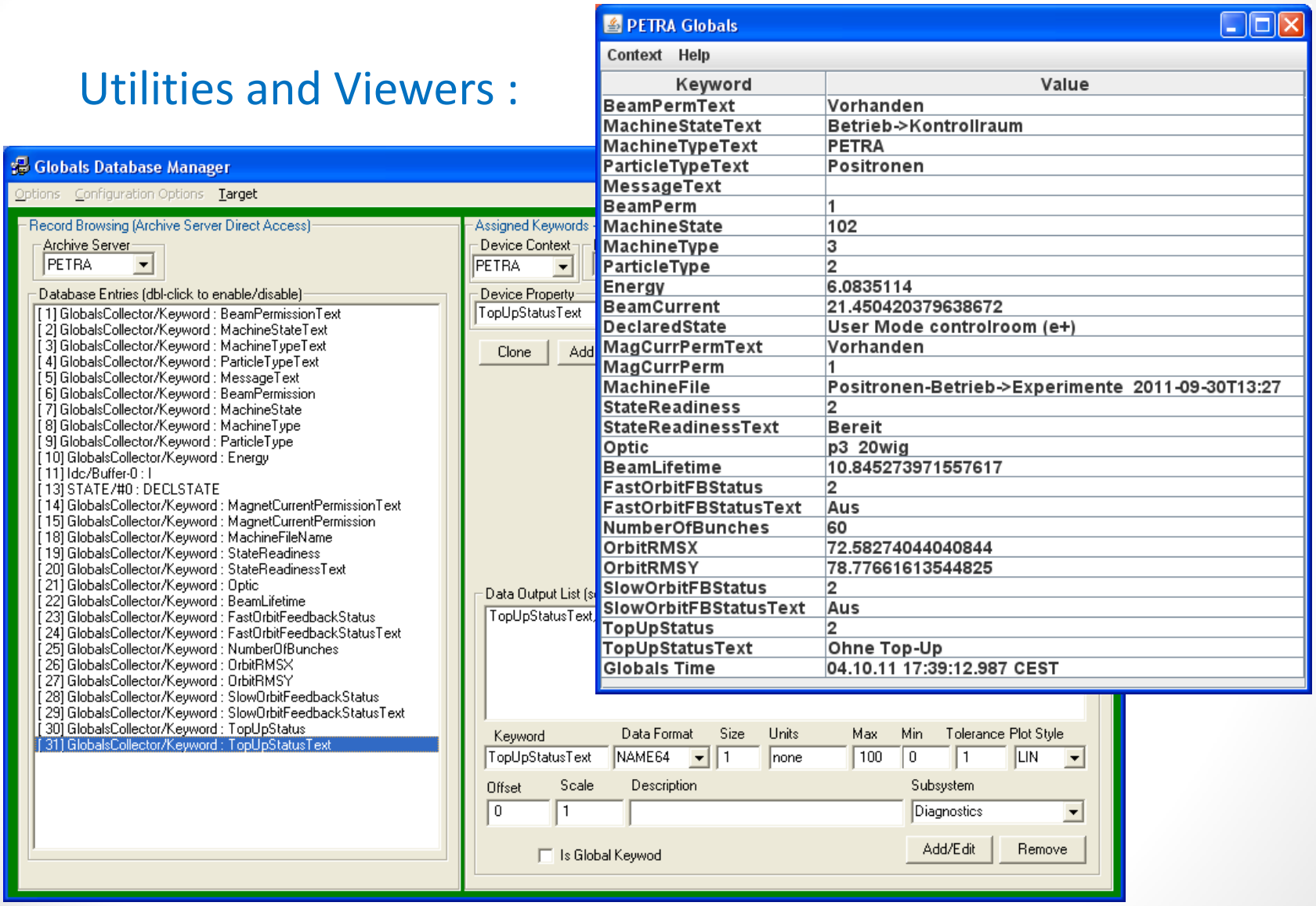
Network Globals

Network Globals

- Keyword parameter set multicasted (producer-consumer)
  - Default rate = 1 Hz
  - Keyword oriented (no device names)
  - Server = "**GLOBALS**" (in the given context).
  - Attempts to attach() to e.g. "/context/GLOBALS[BeamCurrent]" are coerced into listening for globals multicast !
    - 'receive()' (java) or recvNetGlobal() (C) are the preferred API methods.
    - n.b. any server can 'produce' data via 'sendNetGlobal()'.

Network Globals

Utilities and Viewers :



TIME Synchronization

TIME Synchronization

- Server "/SITE/**TIMESRV**" multicasts the reference time at 1 Hz.
- A server automatically syncs to this incoming time (*if available*)
  - *Does NOT adjust local clock* !
  - Applies an *offset* to the current clock when applying data timestamps or log entries.
  - *5 consecutive updates* must give a consistent offset !
    - Considers both jumps and slopes in various NTP correction strategies which might be running in parallel.
  - Offset must be > 100 msec.
- Note:
  - a server schedules activity *based on its clock*
  - a client accepts or rejects incoming data *based on its timestamp* (among other things).

Central Logger

Central Logger

- Server “/SITE/CLOG” is happily logging entries sent its way from any context on site.
  - API: clslog() will send a log entry to the central logger (see http://tine.desy.de) for details.
  - Event Server automatically logs incoming events
  - Watchdog automatically logs restarts
  - *Not seeing much action at the moment ….*

Context	Tag	Logger	Text	Priorität	Status	
DORIS	Services	EVENTSTORE	DOTRCRFSR.1 : TRIGGER dotrcrfsr (set nr 2)	CLOG_PRIORITY_IMPORTANT	CLOG_STATUS_WARN	C
PETRA	Services	EVENTSTORE	PE_SL_TRANSMITTR : TRIGGER mhf_sl1cav_trc (set nr 3)	CLOG_PRIORITY_IMPORTANT	CLOG_STATUS_WARN	C
PETRA	Services	EVENTSTORE	PE_SL_TRANSMITTR : TRIGGER mhf_sl1cav_trc (set nr 3)	CLOG_PRIORITY_IMPORTANT	CLOG_STATUS_WARN	1
FLASH	Services	EVENTSTORE	KLY6ARC.2 : TRIGGER kly6_hv (set nr 13)	CLOG_PRIORITY_IMPORTANT	CLOG_STATUS_WARN	1
PETRA	Services	EVENTSTORE	PE_SL_TRANSMITTR : TRIGGER mhf_sl1cav_trc (set nr 3)	CLOG_PRIORITY_IMPORTANT	CLOG_STATUS_WARN	1
DORIS	Services	EVENTSTORE	DOTRCRFSR.1 : TRIGGER dotrcrfsr (set nr 2)	CLOG_PRIORITY_IMPORTANT	CLOG_STATUS_WARN	1
DORIS	Services	EVENTSTORE	DOTRCRFC.1 : TRIGGER dotrcrfc (set nr 5)	CLOG_PRIORITY_IMPORTANT	CLOG_STATUS_WARN	1
DORIS	Services	EVENTSTORE	DOTRCRFNL.1 : TRIGGER dotrcrfnl (set nr 3)	CLOG_PRIORITY_IMPORTANT	CLOG_STATUS_WARN	1
SERVICE	HASP05VIL	Watchdog	stopping all processes	CLOG_PRIORITY_NONE	CLOG_STATUS_NONE	1
SERVICE	HASP05VIL	Watchdog	CDI-Server stopped.	CLOG_PRIORITY_NONE	CLOG_STATUS_NONE	1
SERVICE	HASP05VIL	Watchdog	See: screen shot	CLOG_PRIORITY_NONE	CLOG_STATUS_NONE	1
DORIS	Services	EVENTSTORE	DOTRCRFC.1 : TRIGGER dotrcrfc (set nr 5)	CLOG_PRIORITY_IMPORTANT	CLOG_STATUS_WARN	1
DORIS	Services	EVENTSTORE	DOTRCRFSR.1 : TRIGGER dotrcrfsr (set nr 2)	CLOG_PRIORITY_IMPORTANT	CLOG_STATUS_WARN	1
DORIS	Services	EVENTSTORE	DOTRCRFNL.1 : TRIGGER dotrcrfnl (set nr 3)	CLOG_PRIORITY_IMPORTANT	CLOG_STATUS_WARN	1
PETRA	Services	EVENTSTORE	PE_SR_Control : TRIGGER mhf_sr1cav_err (set nr 11)	CLOG_PRIORITY_IMPORTANT	CLOG_STATUS_WARN	1
PETRA	Services	EVENTSTORE	PE_SL_Control : TRIGGER mhf_sl1cav_err (set nr 9)	CLOG_PRIORITY_IMPORTANT	CLOG_STATUS_WARN	1
PETRA	Services	EVENTSTORE	LBRESRV : TRIGGER bpm_intlk (set nr 16)	CLOG_PRIORITY_IMPORTANT	CLOG_STATUS_WARN	1

Alarm System

Alarm System

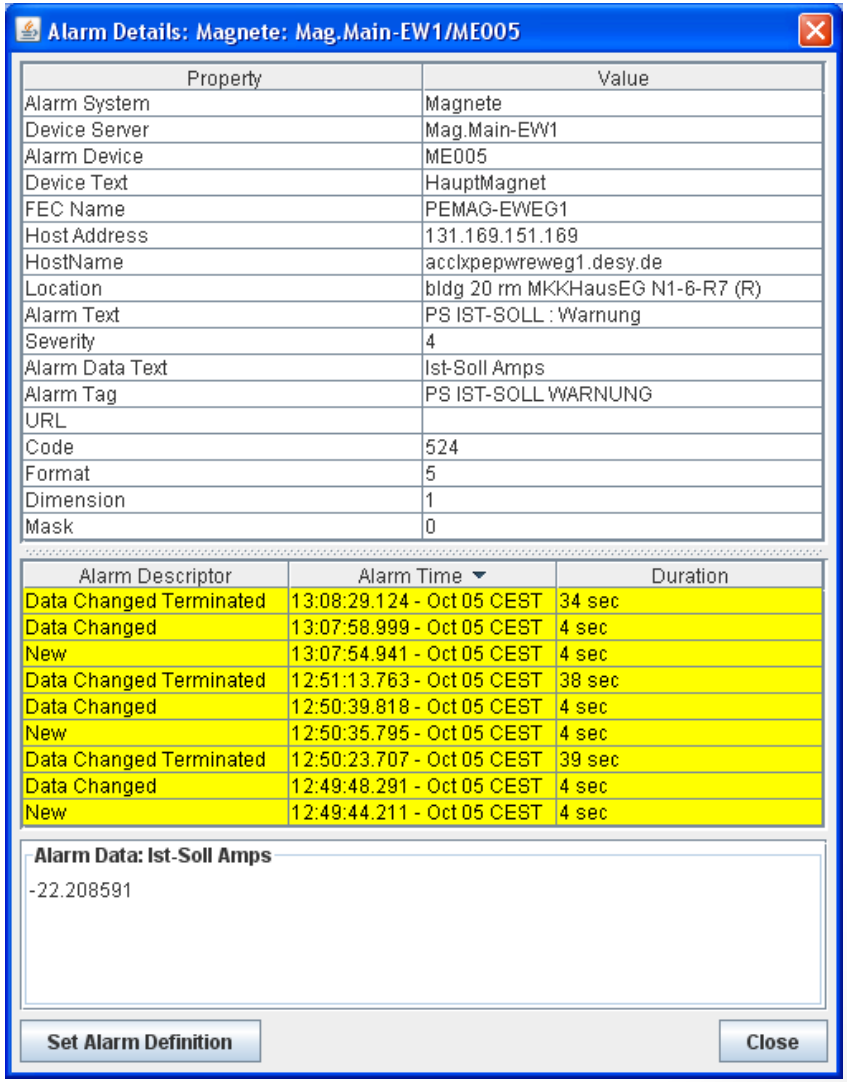
- Alarms
  - Belong to a *registered device* !
  - Defined by: /*context*/*server*/*device* + alarm code + starttime
  - Have a *history*
    - Tagged as '*new*', '*transient*', '*oscillating*', '*data changed*', '*heartbeat*', '*terminated*'.
  - Have '*static*' information defined by alarm code
    - alarm tag, various descriptive texts, url, severity, data format
  - Have '*dynamic'* information
    - descriptor, alarm time, alarm start time, alarm data
  - Alarm Message: *dynamic part* + cross-reference to *static part*.

Alarm System

- e.g. a magnet alarm :

static information:

dynamic information:

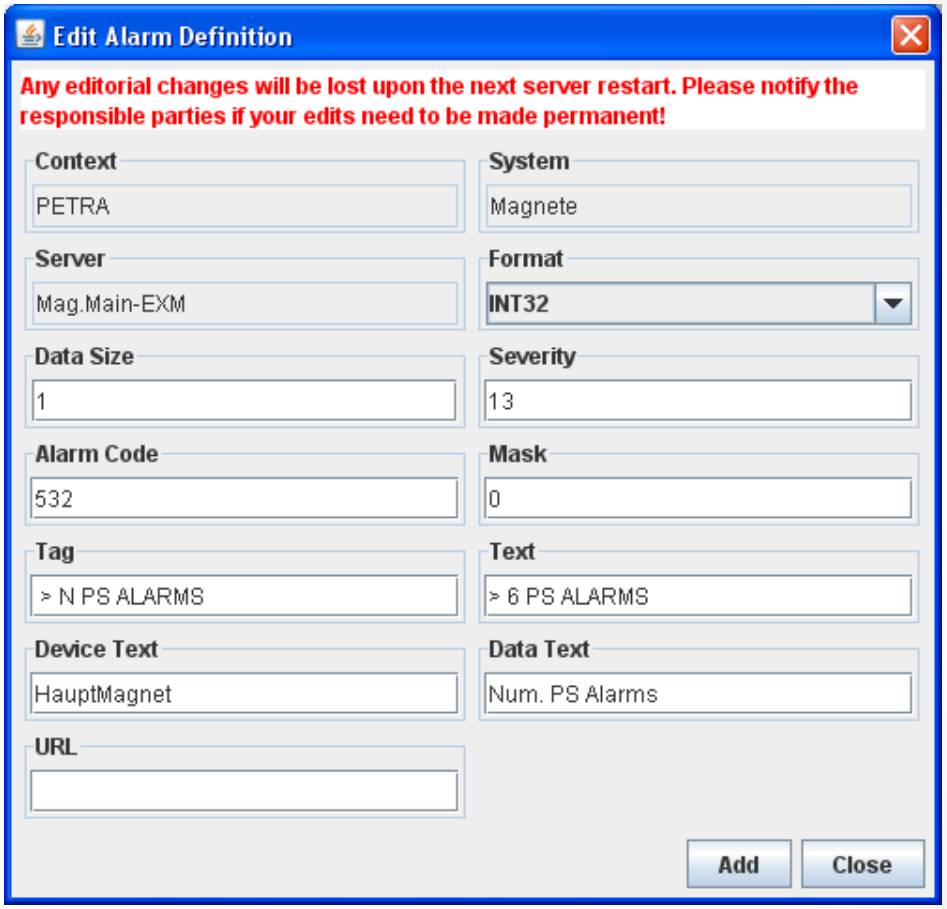


Alarm System

- *static* alarm definitions:
  - Given by alarms.csv (or fec.xml or via API)
  - Can be edited '*on-the-fly*'

    -> but changes are volatile !
  - Best to leave '*alarm system*' = 0

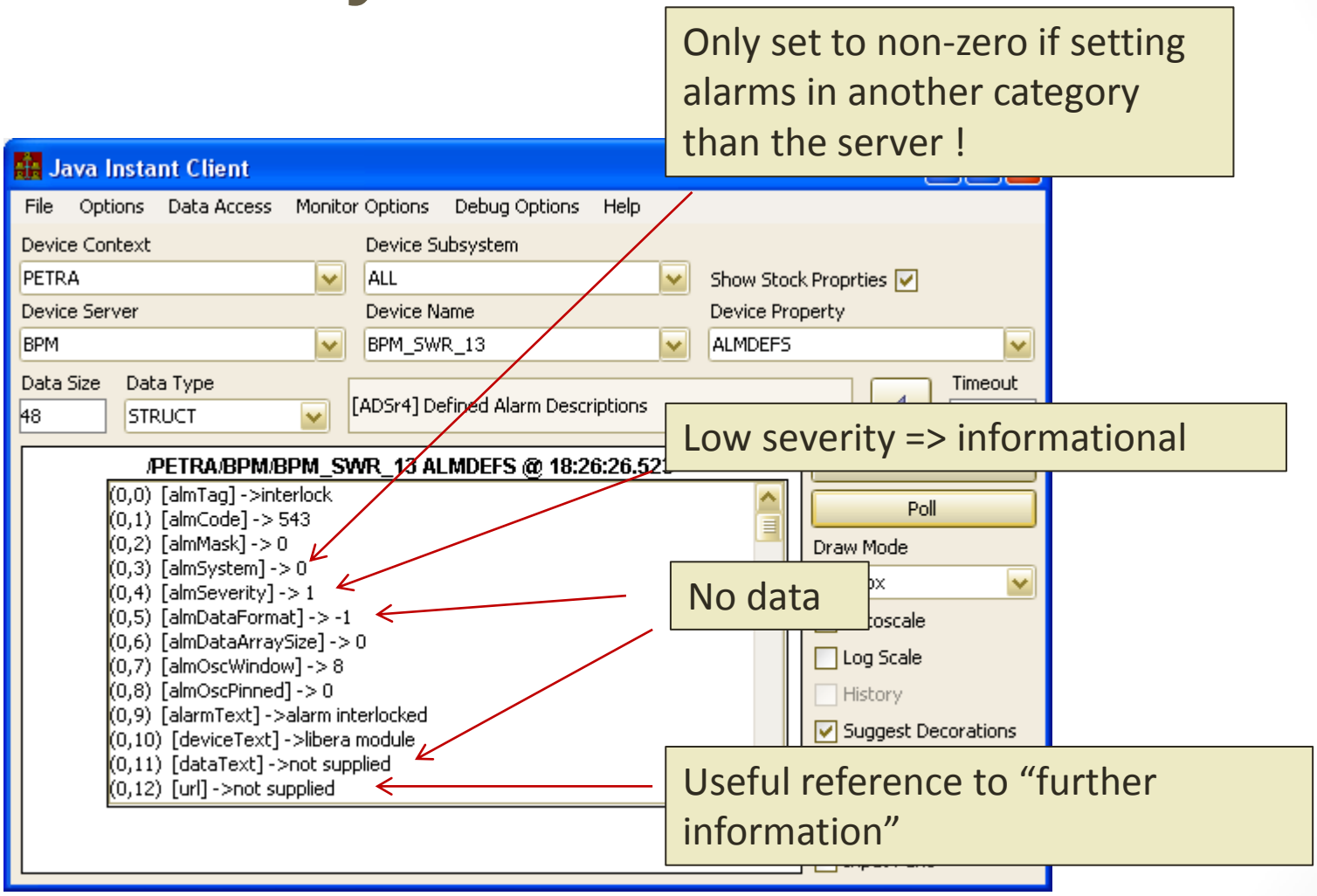
    (signals the CAS to apply the alarm system registered for the server issuing the alarm).



Alarm System

- Alarm Code:
  - Systematic TINE error codes < 512
    - e.g. hardware_error = 79
    - Have default severity (most have ‘8’)
    - Have default alarm system = 0
  - User defined 512 and above (require corresponding static alarm definition in order to apply non-zero severity).
- Alarm System Numbers :
  - No particular systematics used
  - Alarm system ‘tags’ are more relevant in displays
  - e.g. alarm system 100 = ‘Magnets’
- Alarm Severity:
  - 0 = none -> do not display
  - 1 -> 3 = information
  - 4 -> 7 = warning
  - 8 -> 11 = error
  - 12 -> 14 = impending doom
  - 15 = operations not possible
- Alarm Data:
  - 64 bytes to include ‘other relevant information’

Alarm System



Alarm System

- Every server has a *Local Alarm System*
  - Does *nothing* unless:
    - Alarms are *defined* (see alarms.csv, fec.xml, or API call, or code < 512)
      - code, severity, tag, + …
    - Alarms are *set* (and cleared)
      - via API : setAlarm(), clearAlarm() (terminateAlarm())
      - via alarm watch table (see almwatch.csv, fec.xml, or API call)
      - Special cases: Link error alarms, disk space alarms
  - *Manages* the local alarm list to first order
    - Sets the alarm 'descriptor' bits
      - '*new*', '*heartbeat*', '*oscillating*', '*data change*', '*transient*', '*terminated*'
    - Sets the alarm timestamps (start time, alarm time)
    - Collapses alarm *storms* to a single alarm.
  - *Offers* alarms list to any interested clients (e.g. the CAS)
  - *Clears* alarm list at the Central Alarm Server (CAS) upon start up.

Alarm System

- Alarm descriptors
  - '*new*' applied to initial entry into the alarm list.
    - start time = alarm time = time of setAlarm().
  - '*heartbeat*' applied every 20 minutes
    - alarm time updated
  - '*oscillating*' applied when a 'cleared' alarm has been reset prior to termination
    - alarm time updated
    - Note: clearAlarm() augments a clear counter but by itself does not mark an alarm as terminated
    - '*oscillation window*' (default = 8) gives clear-termination threshold.
  - '*data change*' applied when alarm data have changed
    - alarm time updated
    - '*data change window*' (default = 30 sec) gives elapsed time before a new alarm time is applied.

Alarm System

- Alarm descriptors
  - '*terminated*' applied when the alarm is declared as terminated.
    - alarm time = time of termination.
    - clear counter > oscillation window
    - removeAlarm() has been called.
  - '*transient*' applied when setAlarm() declares the alarm as transient.
    - start time = alarm time = time of setAlarm()
    - '*new*' and '*terminated*' applied simultaneously !
    - does not have a duration !
  - '*test*' (= '*suppress*') is ignored by the CAS
  - '*disabled*' is set by the CAS

Alarm System

- setAlarm() strategies (alarm system managed):

  Let the system check for oscillating alarms !

```
public void update()
{
  clearAlarms();

  int[] v = new int[numberValues];

  getReadbackValues(v);

  for (int i = 0; i < numberValues; i++)
  {
    if (v[i] > numberThreshold) setAlarm(512,v[i]);
  }
}
```

clear all alarms at start of update cycle

get the new data

set alarm if necessary

- setAlarm() strategies (user managed):

  'remove' marks an alarm for termination immediately !

```
public void update()
{
  int[] v = new int[numberValues];

  getReadbackValues(v);

  for (int i = 0; i < numberValues; i++)
  {
    if (v[i] <= numberThreshold)
    {
      removeAlarm(512);
    }
    else
    {
      setAlarm(512,v[i]);
    }
  }
}
```

get the new data

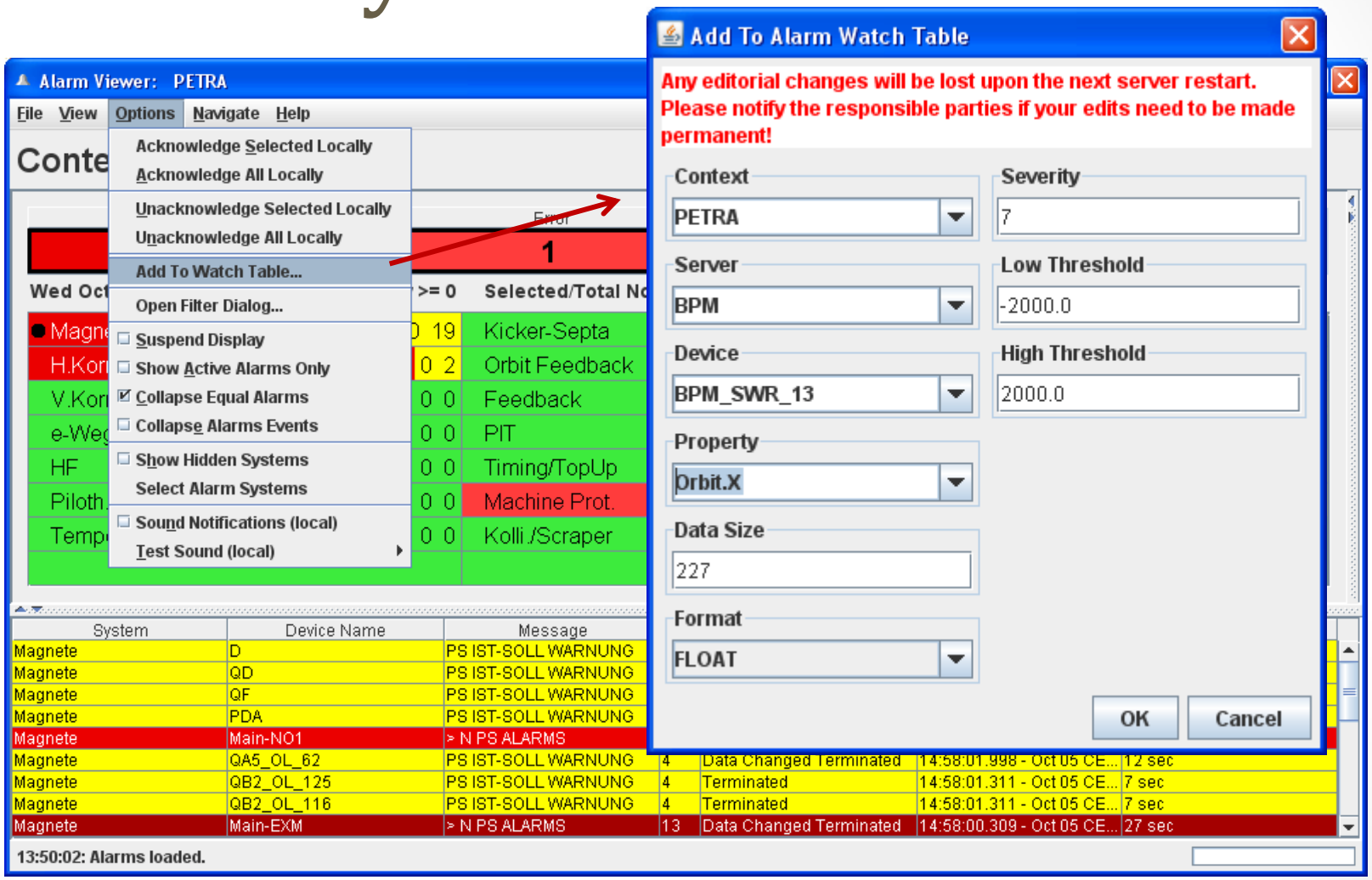
either remove alarm

or set alarm

Alarm System

- Automatic Alarms: *Alarm Watch Table*
  - *Monitors* a specific property via a local call to the associated equipment module.
  - *Checks* readback value against low and high *thresholds*
    - value_too_high, value_too_low
  - *or Checks* a readback value against a valid *pattern*
    - invalid_data
  - Supply monitor parameters via almwatch.csv, fec.xml, or API.
    - Can supply '*on-the-fly*' (but new information is volatile).
  - All 'setAlarm()' and 'clearAlarm()' logic is done for you.

Alarm System



Alarm System

- Automatic Alarms:
  - ‘link_error’ alarms (middle layer servers).
    - can suppress if desired
- Disk space alarms:
  - ‘low_disk_space’ if given path does not have the registered minimum disk space.
    - Yes, there’s an API call.

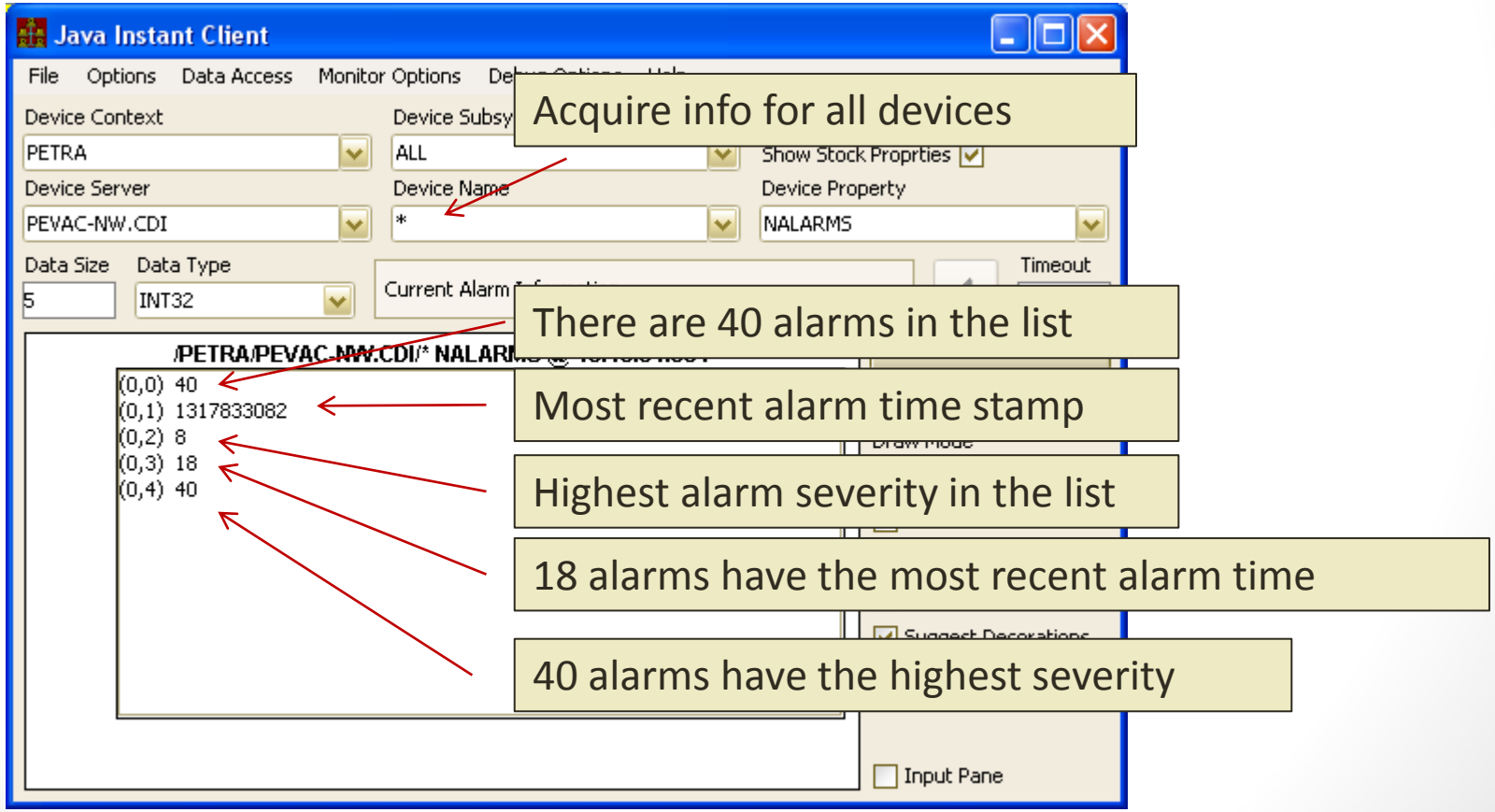
Alarm System

- Central Alarm Server (CAS)
  - Uses a '*pull*' strategy to acquire alarms
    - NOT a '*push*' strategy from the server!
  - Has a *configuration database* giving which servers to listen to.
    - A server knows if the CAS is listening to it !
    - Servers 'clear' their alarm lists at the CAS when the start up.
      - /*context*/**CAS**/*server* + "**REMOVEALARMS**"
  - Can take '*actions*' upon specific alarms
    - Trigger events
    - Send emails (SMS)

- Alarms pulled via stock properties "NALARMS" and "ALARMS"

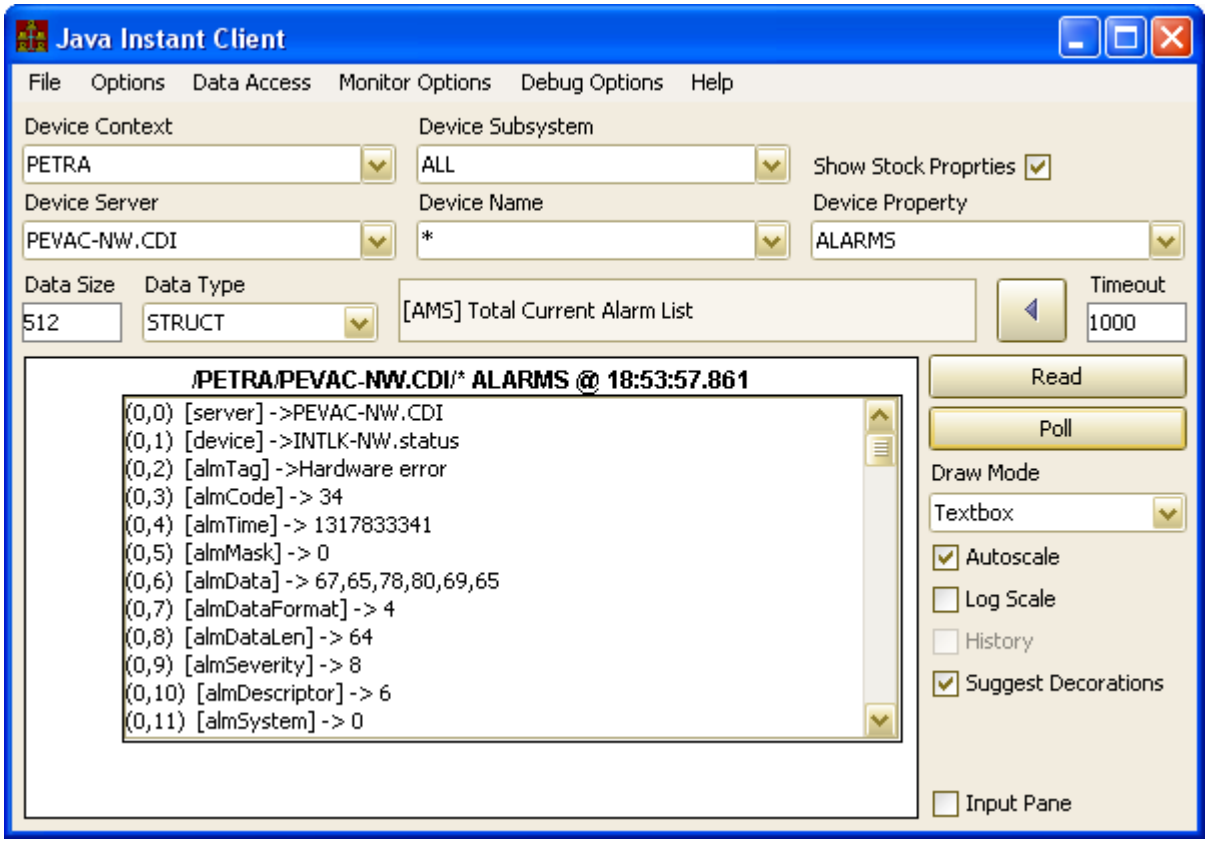
Alarm System

- "NALARMS"
  - Provides a 'snapshot' of the current alarm situation at the server.



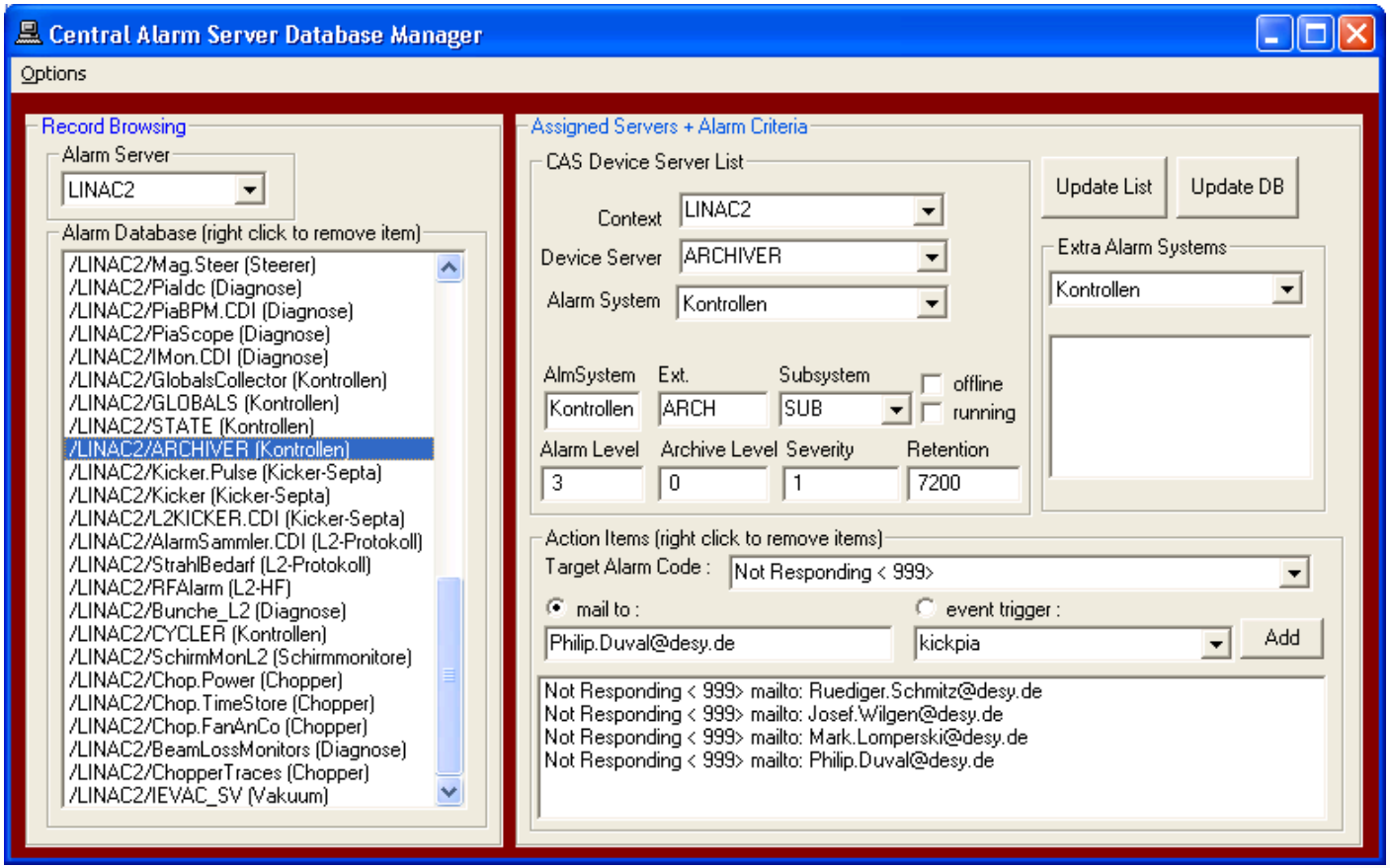
Alarm System

- "ALARMS"
  - Can provide range as input (default = all alarm times)
  - And minimum severity as input (default = 0)



Alarm System

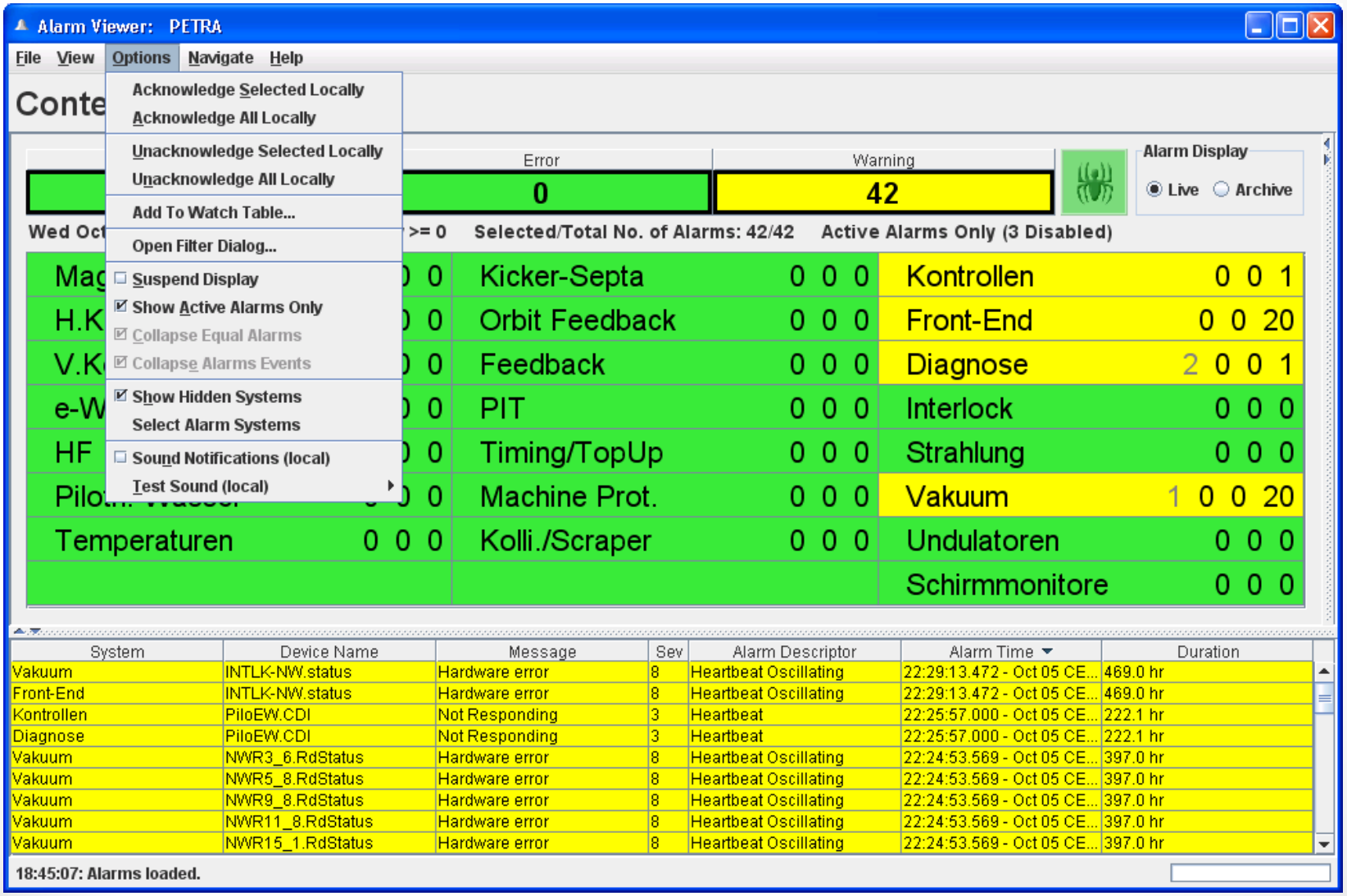
- Database manager:



Alarm System

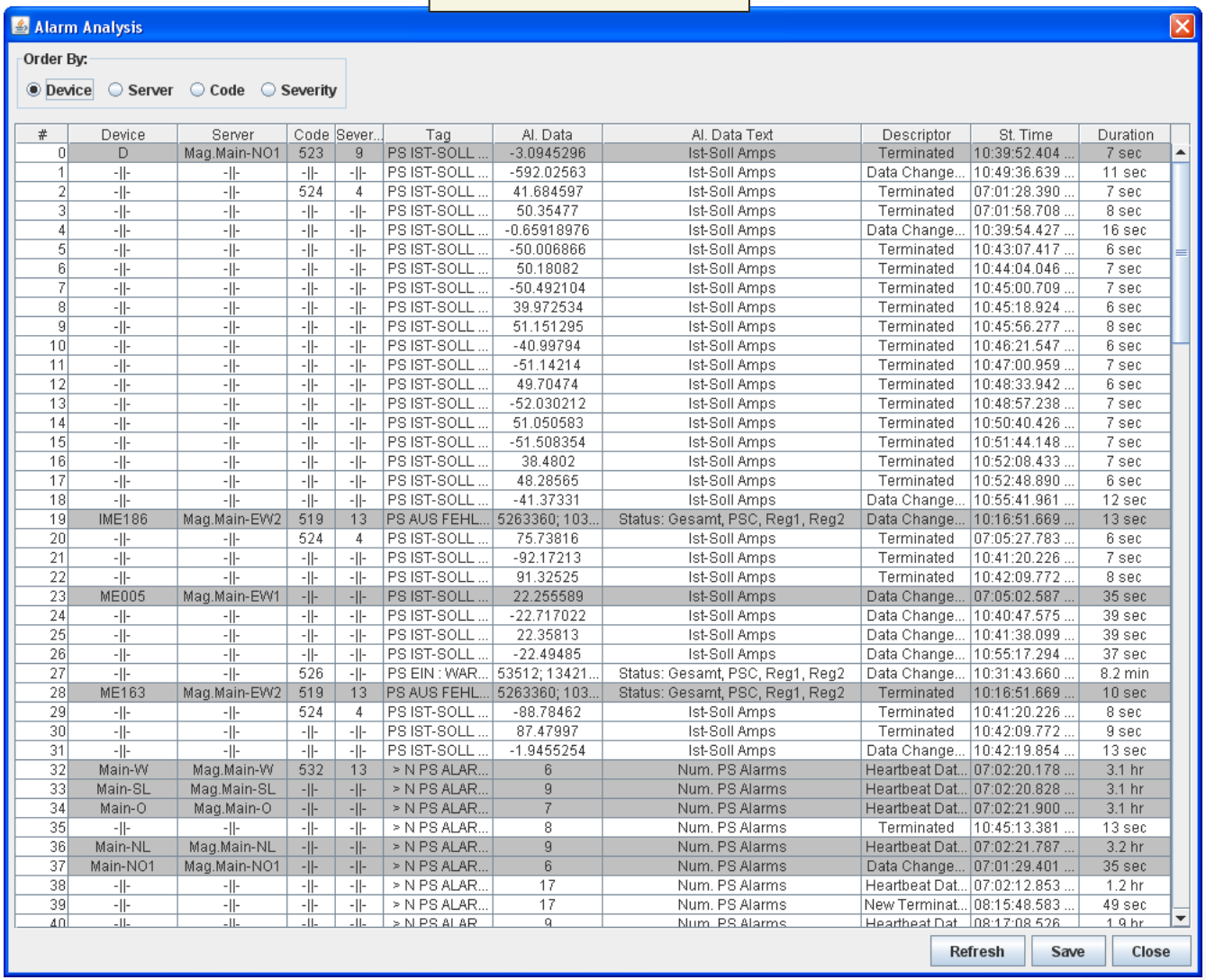
- Alarm Systems manager:

Alarm System



Alarm System

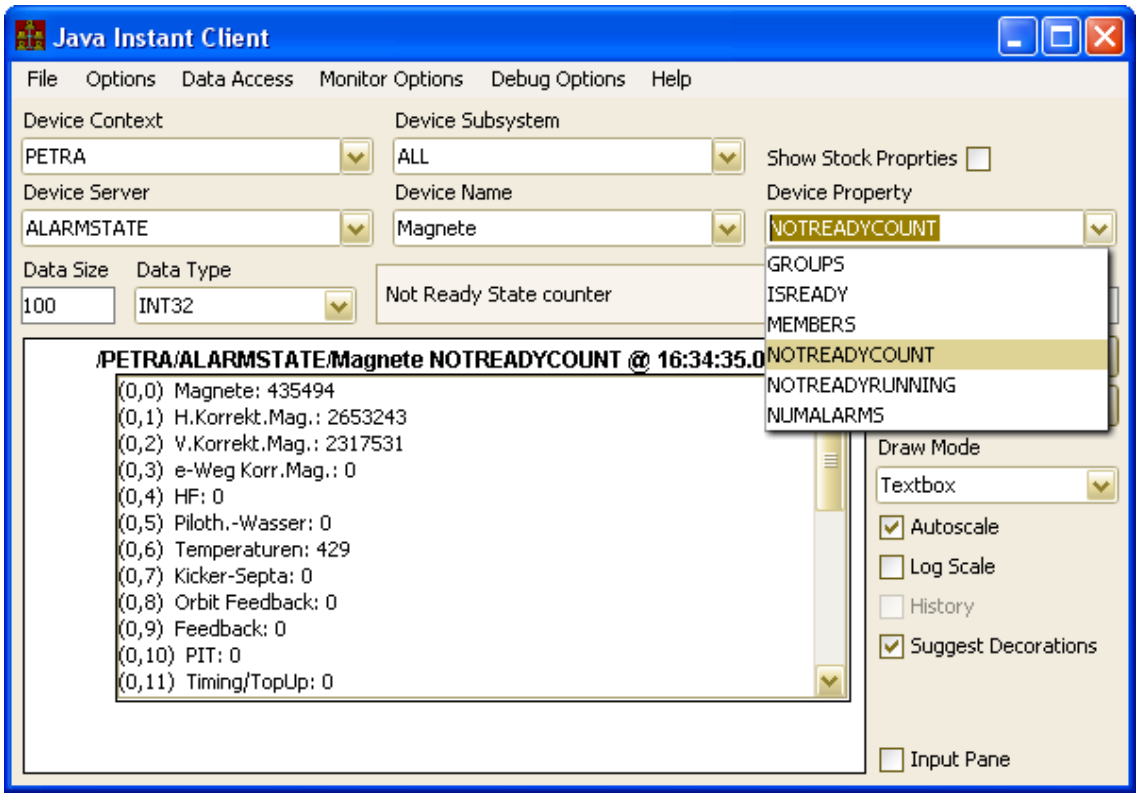
Alarm Analysis



#	Device	Server	Code	Sever...	Tag	Al. Data	Al. Data Text	Descriptor	St. Time	Duration
0	D	Mag.Main-NO1	523	9	PS IST-SOLL ...	-3.0945296	Ist-Soll Amps	Terminated	10:39:52.404 ...	7 sec
1	-||-	-||-	-||-	-||-	PS IST-SOLL ...	-592.02563	Ist-Soll Amps	Data Change...	10:49:36.639 ...	11 sec
2	-||-	-||-	524	4	PS IST-SOLL ...	41.684597	Ist-Soll Amps	Terminated	07:01:28.390 ...	7 sec
3	-||-	-||-	-||-	-||-	PS IST-SOLL ...	50.35477	Ist-Soll Amps	Terminated	07:01:58.708 ...	8 sec
4	-||-	-||-	-||-	-||-	PS IST-SOLL ...	-0.65918976	Ist-Soll Amps	Data Change...	10:39:54.427 ...	16 sec
5	-||-	-||-	-||-	-||-	PS IST-SOLL ...	-50.006866	Ist-Soll Amps	Terminated	10:43:07.417 ...	6 sec
6	-||-	-||-	-||-	-||-	PS IST-SOLL ...	50.18082	Ist-Soll Amps	Terminated	10:44:04.046 ...	7 sec
7	-||-	-||-	-||-	-||-	PS IST-SOLL ...	-50.492104	Ist-Soll Amps	Terminated	10:45:00.709 ...	7 sec
8	-||-	-||-	-||-	-||-	PS IST-SOLL ...	39.972534	Ist-Soll Amps	Terminated	10:45:18.924 ...	6 sec
9	-||-	-||-	-||-	-||-	PS IST-SOLL ...	51.151295	Ist-Soll Amps	Terminated	10:45:56.277 ...	8 sec
10	-||-	-||-	-||-	-||-	PS IST-SOLL ...	-40.99794	Ist-Soll Amps	Terminated	10:46:21.547 ...	6 sec
11	-||-	-||-	-||-	-||-	PS IST-SOLL ...	-51.14214	Ist-Soll Amps	Terminated	10:47:00.959 ...	7 sec
12	-||-	-||-	-||-	-||-	PS IST-SOLL ...	49.70474	Ist-Soll Amps	Terminated	10:48:33.942 ...	6 sec
13	-||-	-||-	-||-	-||-	PS IST-SOLL ...	-52.030212	Ist-Soll Amps	Terminated	10:48:57.238 ...	7 sec
14	-||-	-||-	-||-	-||-	PS IST-SOLL ...	51.050583	Ist-Soll Amps	Terminated	10:50:40.426 ...	7 sec
15	-||-	-||-	-||-	-||-	PS IST-SOLL ...	-51.508354	Ist-Soll Amps	Terminated	10:51:44.148 ...	7 sec
16	-||-	-||-	-||-	-||-	PS IST-SOLL ...	38.4802	Ist-Soll Amps	Terminated	10:52:08.433 ...	7 sec
17	-||-	-||-	-||-	-||-	PS IST-SOLL ...	48.28565	Ist-Soll Amps	Terminated	10:52:48.890 ...	6 sec
18	-||-	-||-	-||-	-||-	PS IST-SOLL ...	-41.37331	Ist-Soll Amps	Data Change...	10:55:41.961 ...	12 sec
19	IME186	Mag.Main-EW2	519	13	PS AUS FEHL...	5263360; 103...	Status: Gesamt, PSC, Reg1, Reg2	Data Change...	10:16:51.669 ...	13 sec
20	-||-	-||-	524	4	PS IST-SOLL ...	75.73816	Ist-Soll Amps	Terminated	07:05:27.783 ...	6 sec
21	-||-	-||-	-||-	-||-	PS IST-SOLL ...	-92.17213	Ist-Soll Amps	Terminated	10:41:20.226 ...	7 sec
22	-||-	-||-	-||-	-||-	PS IST-SOLL ...	91.32525	Ist-Soll Amps	Terminated	10:42:09.772 ...	8 sec
23	ME005	Mag.Main-EW1	-||-	-||-	PS IST-SOLL ...	22.255589	Ist-Soll Amps	Data Change...	07:05:02.587 ...	35 sec
24	-||-	-||-	-||-	-||-	PS IST-SOLL ...	-22.717022	Ist-Soll Amps	Data Change...	10:40:47.575 ...	39 sec
25	-||-	-||-	-||-	-||-	PS IST-SOLL ...	22.35813	Ist-Soll Amps	Data Change...	10:41:38.099 ...	39 sec
26	-||-	-||-	-||-	-||-	PS IST-SOLL ...	-22.49485	Ist-Soll Amps	Data Change...	10:55:17.294 ...	37 sec
27	-||-	-||-	526	-||-	PS EIN : WAR...	53512; 13421...	Status: Gesamt, PSC, Reg1, Reg2	Data Change...	10:31:43.660 ...	8.2 min
28	ME163	Mag.Main-EW2	519	13	PS AUS FEHL...	5263360; 103...	Status: Gesamt, PSC, Reg1, Reg2	Terminated	10:16:51.669 ...	10 sec
29	-||-	-||-	524	4	PS IST-SOLL ...	-88.78462	Ist-Soll Amps	Terminated	10:41:20.226 ...	8 sec
30	-||-	-||-	-||-	-||-	PS IST-SOLL ...	87.47997	Ist-Soll Amps	Terminated	10:42:09.772 ...	9 sec
31	-||-	-||-	-||-	-||-	PS IST-SOLL ...	-1.9455254	Ist-Soll Amps	Data Change...	10:42:19.854 ...	13 sec
32	Main-W	Mag.Main-W	532	13	> N PS ALAR...	6	Num. PS Alarms	Heartbeat Dat...	07:02:20.178 ...	3.1 hr
33	Main-SL	Mag.Main-SL	-||-	-||-	> N PS ALAR...	9	Num. PS Alarms	Heartbeat Dat...	07:02:20.828 ...	3.1 hr
34	Main-O	Mag.Main-O	-||-	-||-	> N PS ALAR...	7	Num. PS Alarms	Heartbeat Dat...	07:02:21.900 ...	3.1 hr
35	-||-	-||-	-||-	-||-	> N PS ALAR...	8	Num. PS Alarms	Terminated	10:45:13.381 ...	13 sec
36	Main-NL	Mag.Main-NL	-||-	-||-	> N PS ALAR...	9	Num. PS Alarms	Heartbeat Dat...	07:02:21.787 ...	3.2 hr
37	Main-NO1	Mag.Main-NO1	-||-	-||-	> N PS ALAR...	6	Num. PS Alarms	Data Change...	07:01:29.401 ...	35 sec
38	-||-	-||-	-||-	-||-	> N PS ALAR...	17	Num. PS Alarms	Heartbeat Dat...	07:02:12.853 ...	1.2 hr
39	-||-	-||-	-||-	-||-	> N PS ALAR...	17	Num. PS Alarms	New Terminat...	08:15:48.583 ...	49 sec
40	-||-	-||-	-||-	-||-	> N PS ALAR...	9	Num. PS Alarms	Heartbeat Dat...	08:17:08.526 ...	1.9 hr

Alarm System

- Availability
  - server : "/<*context*>/**ALARMSTATE**"
  - monitor *fatal alarms* for all alarm systems from **CAS**.
  - At least 1 alarm => system *NOT AVAILABLE*



Archive System

Archive System

- **Central Archive** (2 processes)
  - "ARCHIVER" : Netmex Gateway / Archiver
  - "HISTORY" : archive retriever / viewer configurations
  - data always available
  - many filter options to reduce unnecessary data storage
  - '*property*' server
- **Local History** (server sub system)
  - *short term* storage (ring buffer with defined depth)
  - *long term* storage (stored on disk with defined depth in months)
  - data disappears when '*depth*' is surpassed
    - Can optionally and purposefully '*keep*' data if desired.
  - filter on tolerance only (absolute or relative).
  - '*device*' server
    - history data accessed via <*property*>.HIST meta property

Archive System

- *Archive Record*
  - Refers to data set take from some *property* and *device(s)*.
  - Make use of multi-channel arrays (MCAs) as often a possible.
    - *=> record is an array*
- *Archive Retrieval*
  - Single array *element* or *scalar* over a time range
  - *Entire record* at a specified time.
  - Can Specify
    - Range : *start time* to *stop time* (default: now – depth to now)
    - Array element index : (default: given by device name)
    - Raster : (default: automatic)
      - -> YES, we can return ALL stored data in the doocs manner as well.
    - Filter : min value, max value (default: ALL)

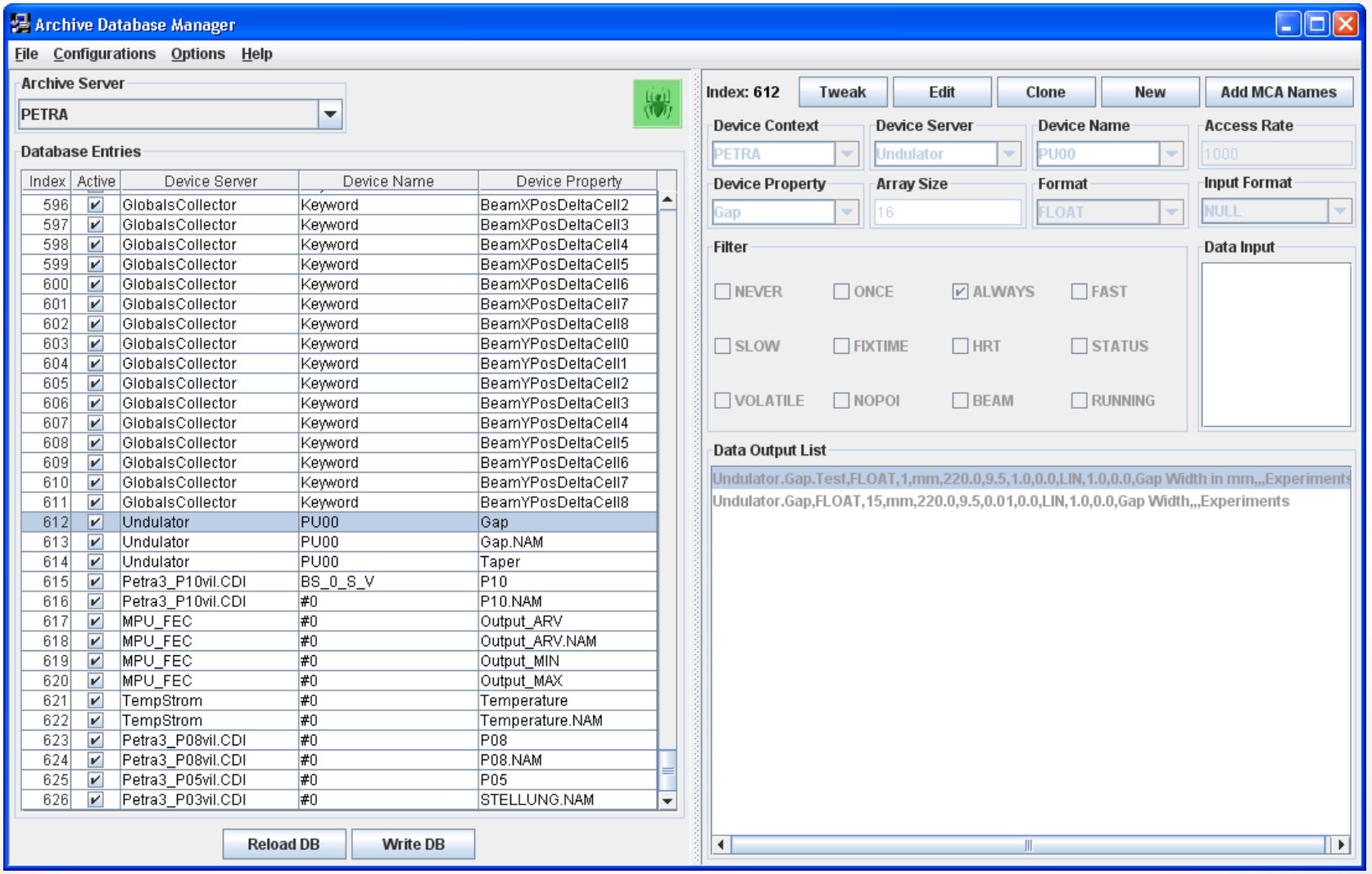
Archive System

- Notes about rastering
  - By default: the *server decides on a raster*
    - Specifying start and stop times and buffer size defines the raster
    - e.g. last 24 hours, 4000 points
  - Lightning fast look ups !
    - Skip through the stored data at the raster points !
    - No data massaging or the like
      - ***What was taken is what was stored is what you get !***
  - Points of Interest are included in the returned data set if the raster does NOT = 1.
    - *Peaks and dips will appear in the displayed data set !*
    - *(this isn't always perfect).*
  - Archive Viewer uses an '*optical zoom*' approach
    - Any zoom will re-acquire data sets at the new zoom boundaries
    - And will have a smaller raster !

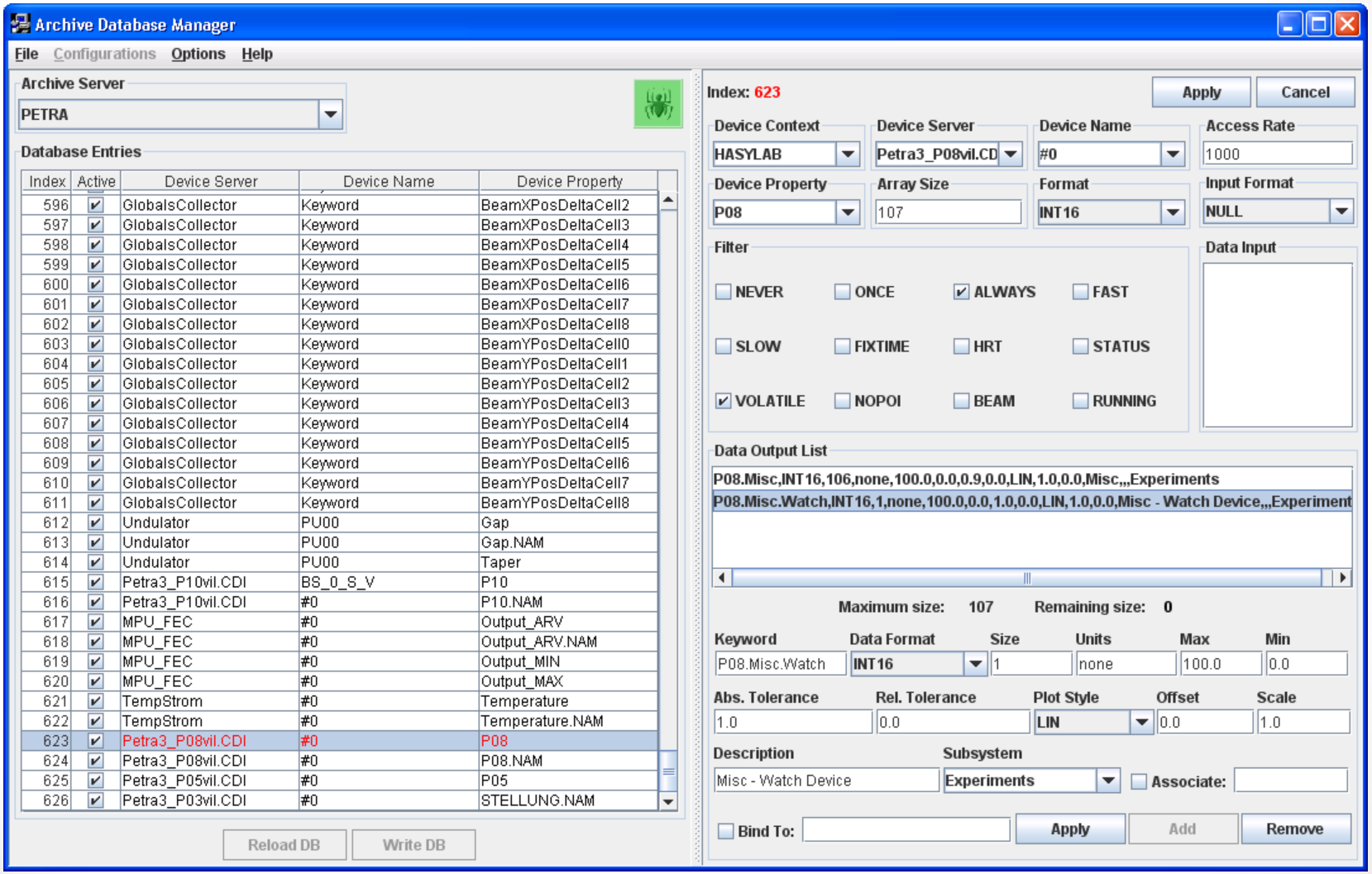
Archive System

- Central Archive Server
  - Breaks down acquired data records into properties and keywords
    - Data from “/PETRA/SomeServer/#0[ALLDATA]” can be given sensible record names !
    - e.g. “WidgetTemperature”
  - Has a broad and definable range of filters.
  - Registers property aliases referring to originating property call.
    - e.g. “SomeServer.ALLDATA” is a property alias for “WidgetTemperature”
  - Has a database manager

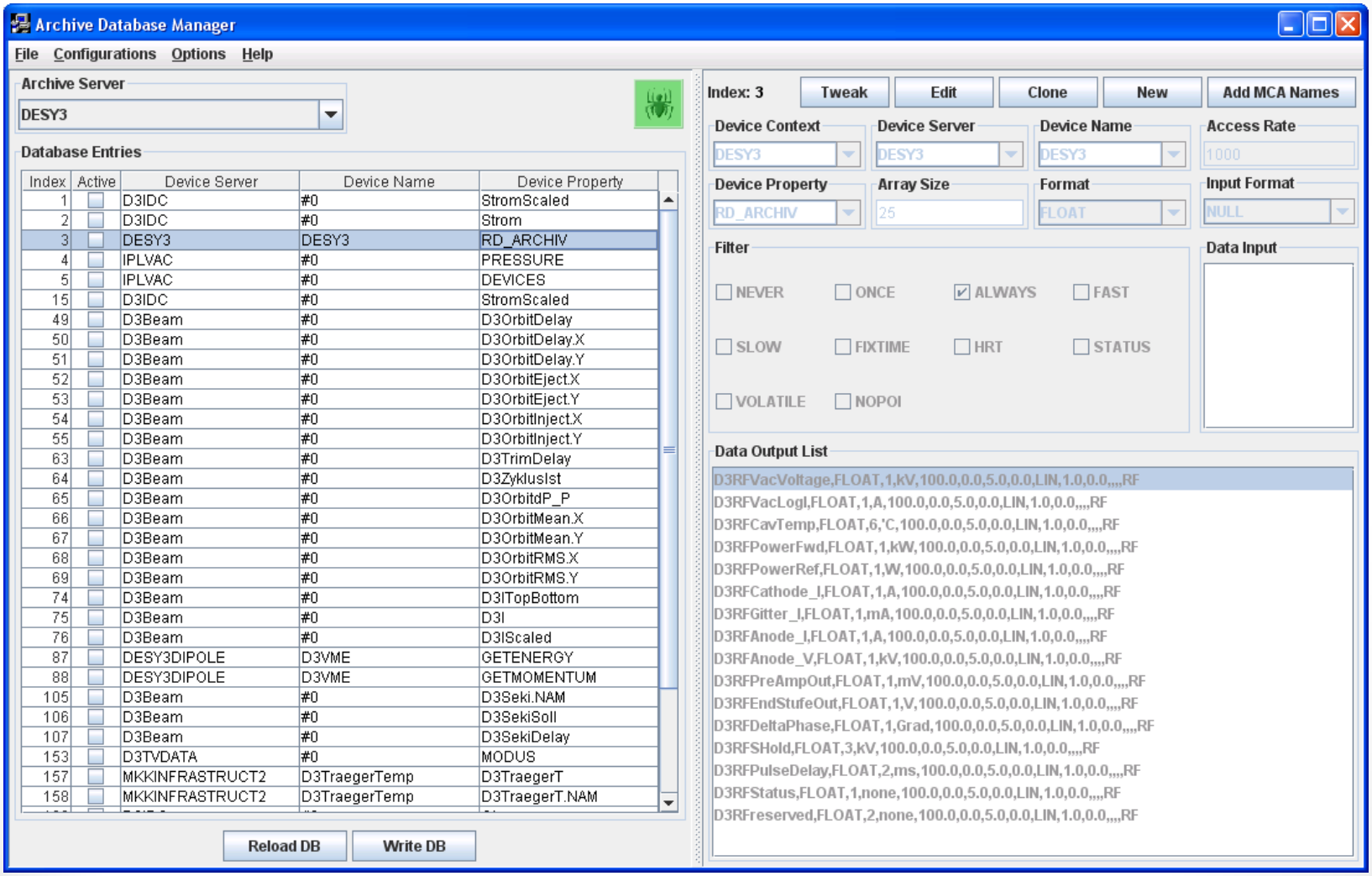
Archive System



Archive System

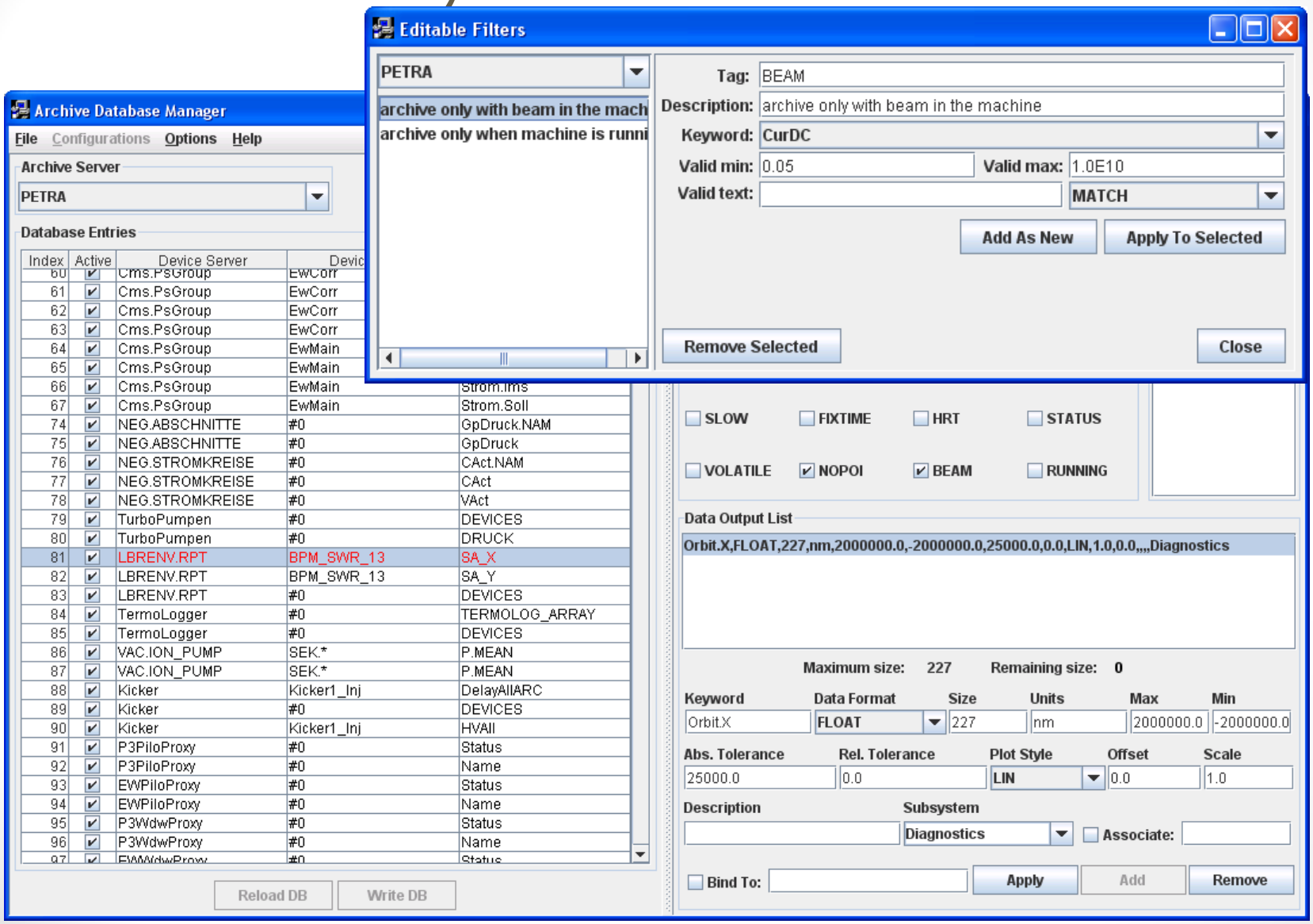


Archive System



Archive System

Archive Filters



Archive System

- *Filters + Archiving Rules*
  - call status <> 0 => *do not archive*
    - *But call status is now archived separately !*
  - data timestamp does NOT change => *do not archive*
  - FILTER = ONCE -> archive a single record at
    - archive server start
    - reconnect to server (following server link timeouts)
    - Midnight
    - e.g. MCA array device names
  - archive at least once per archive heartbeat (15 minutes)
  - data change *within tolerance* ? -> *do not archive* (unless heartbeat)
  - FILTER = FAST -> archive at polling interval
  - FILTER = SLOW -> archive no more often than once/minute
  - FILTER = NEVER -> *do not archive*
    - record available as NETMEX entry only (data pump)
  - else : archive no more often than once/two seconds
  - other defined filters also apply ….

Archive System

- Central Archive Server
  - Manages all viewer configurations
    - Archive Viewer
    - MCA Viewer
    - Scope Trace Viewer
  - Now stores call status/record if NOT = 0
  - Now stores user provided annotations/record

Archive System

- Local Archive System
  - *Record* given by a call to the local equipment module
    - must ensure *unique record index* !
      - Must have consistent data size and format !
    - no data kept if return code <> 0 !
    - access parameter contains the CA_HIST bit if call is coming from the local history sub-system.
      - *Note: This is useful !*
  - *Short term* ring buffer storage
    - depth of ring buffer => depth in time
      - Polling interval = 1 Hz -> depth is in seconds
    - filter on data time stamp only !
    - *volatile*: starts from scratch upon server restart !
  - *Long term* disk storage
    - depth in months
      - keeps 'daily' files by default
    - filter on tolerance (absolute or relative)
    - old data removed
      - Can also set minimum free disk space !
      - Can also explicitly move files to SAVED area !

Archive System

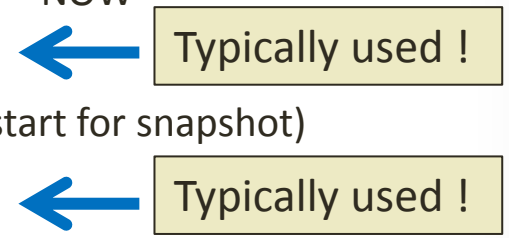
- Local Archive System
  - *local configuration database*
    - history.csv, fec.xml, or via API
    - contain call, storage, and filter information
    - history manifest (hstmf.csv) is dumped after server start.
  - HISTORY_HOME gives location of archive files
    - default = "../HISTORY"
  - Use 'mkhstfiles' utility to make the '*standard*' set of history files
    - '*worst-case*' non-fragmented file set
      - => fast lookups
      - rotated when necessary
    - very useful on windows systems !
      - e.g. pandora servers
  - *Note: java servers store data primitives as big-endian regardless of the platform architecture !*

Archive System

- Local Archive System
  - data retrieval based on stock meta properties
    - <*property*>.HIST (time range)
      - Note: *input = 1 integer value gives number points in range !*
    - <*property*>.HIST@ (snapshot at given time)
      - Note: *returns next record at or more recent than given time !*
    - <*property*>.ARCH (redirect to central archive)
  - If selected property is not being archived then these calls return an error !
    - A property query will return the short and long term depths
      - **= 0** *=> no history configured* !
  - Can add/edit local history information '*on-the-fly*'
    - *Changes are volatile !*

Archive System

- Local/Central Archive System
  - *input*:
    - CF_NULL
      - start time given by requested data array size; stop = NOW
    - 1 CF_INT32, CF_DOUBLE
      - UTC start time (stop = NOW for trend call; stop = start for snapshot)
    - 2 CF_INT32, CF_DOUBLE
      - UTC start and stop times
    - 3 CF_INT32, CF_DOUBLE
      - UTC start and stop times; desired array index (default = 0)
    - 4 CF_INT32, CF_DOUBLE
      - UTC start and stop times; desired array index (default = 0); sampling raster
    - 1 CF_FWINDOW (CF_INTINTFLTFLT)
      - UTC start and stop times; lower and upper data limits



Archive System

- Local/Central Archive System
  - requested data types
    - 1 CF_INT32, etc.
      - -> return number points in interval given
    - CF_FLOAT and other ‘simple’ types
      - snapshot of the stored data at time requested.
    - CF_DBLDBL, CF_FLTINT, CF_INTINT, CF_NAME64INT, etc.
      - -> array of value-timestamp pairs
    - CF_TDS (CF_INTFLTINT)
      - DOOCS style (UTC time, data, status (= 0))
    - CF_HISTORY
      - Array of ‘HISTORY’ instances
        - Carries any stored data type + timestamp, system stamp, user stamp

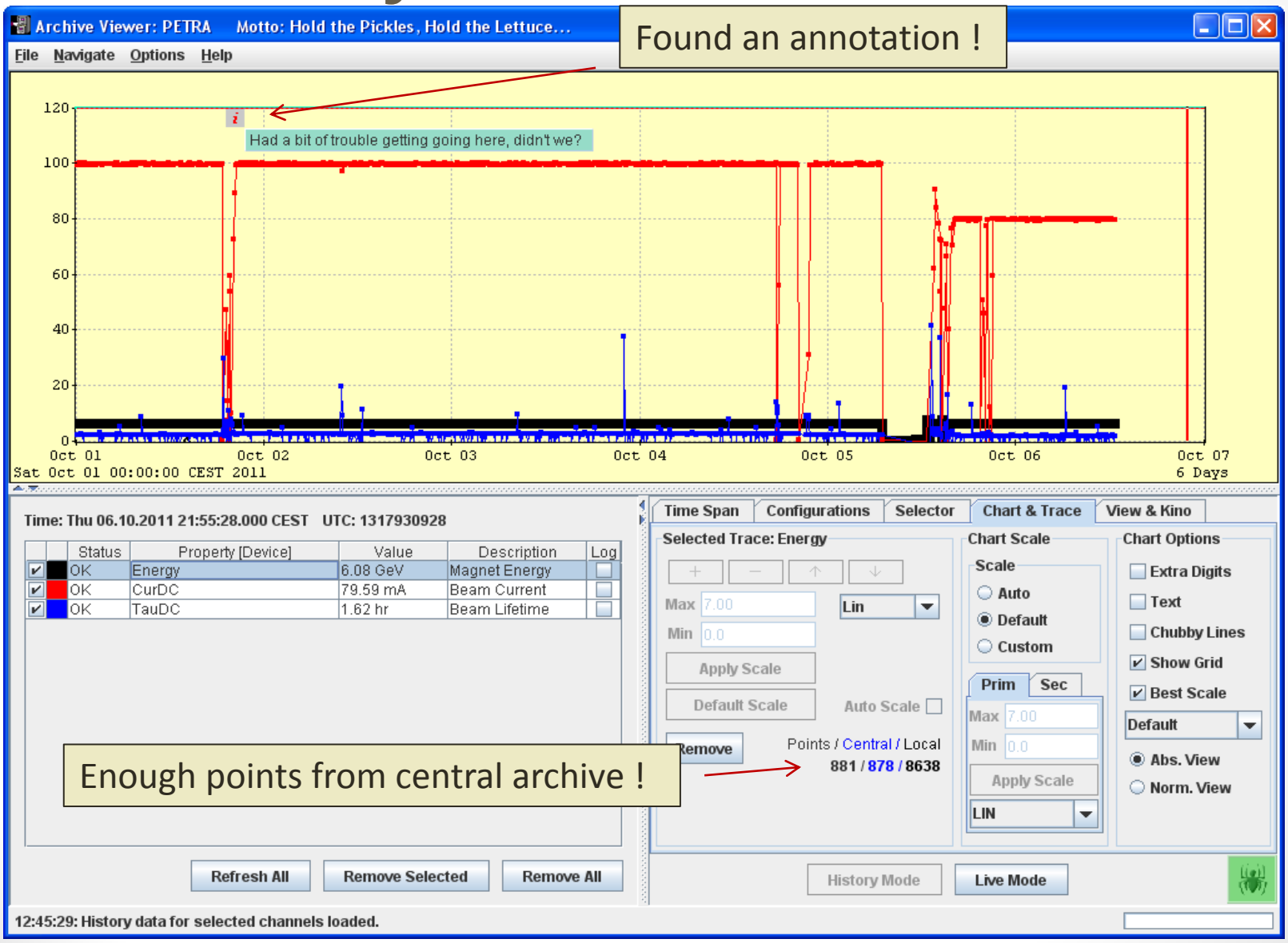
Archive System

- Local/Central Archive System user **APIs**
  - *don't try use the 'stock' and 'meta' properties unless you really know what you're doing* !
  - C-Lib:
    - GetArchivedDataAsAny()
    - GetArchivedDataAsText()
    - GetArchivedDataAsSnapshot()
    - GetArchivedDataAsFloat()
  - Java:
    - Thistory class with lots of static methods to retrieve data !
  - MatLab:
    - tine_history()
  - command line:
    - thistory

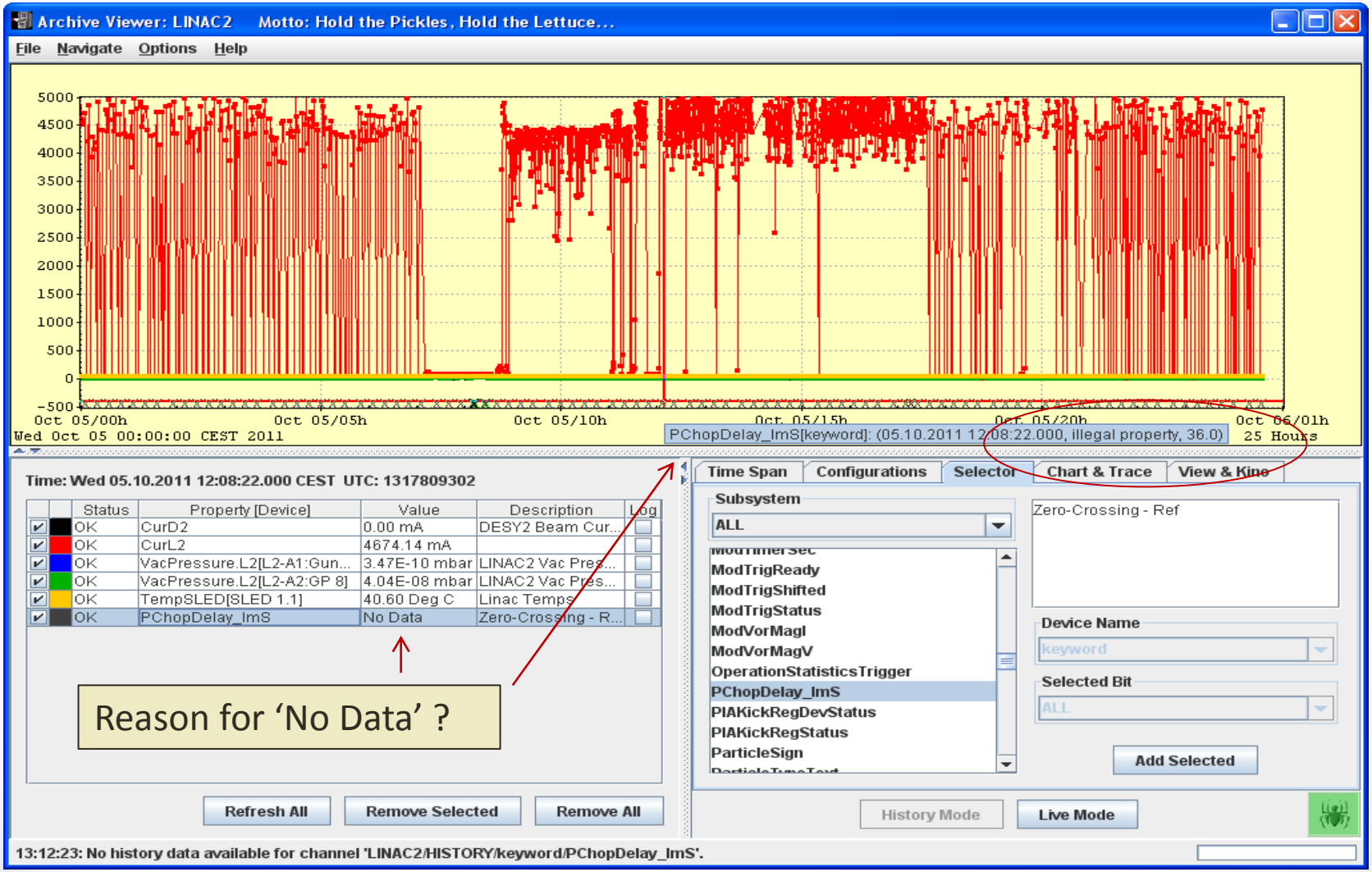
Archive System

- Archive Viewer
  - acquires and displays trend data
    - Normal mode
      - 2 sources ! (central archive and local history)
      - displays data from source with fewest number points in given range but with at least 500 !
      - checks for status information
      - checks for annotations

Archive System



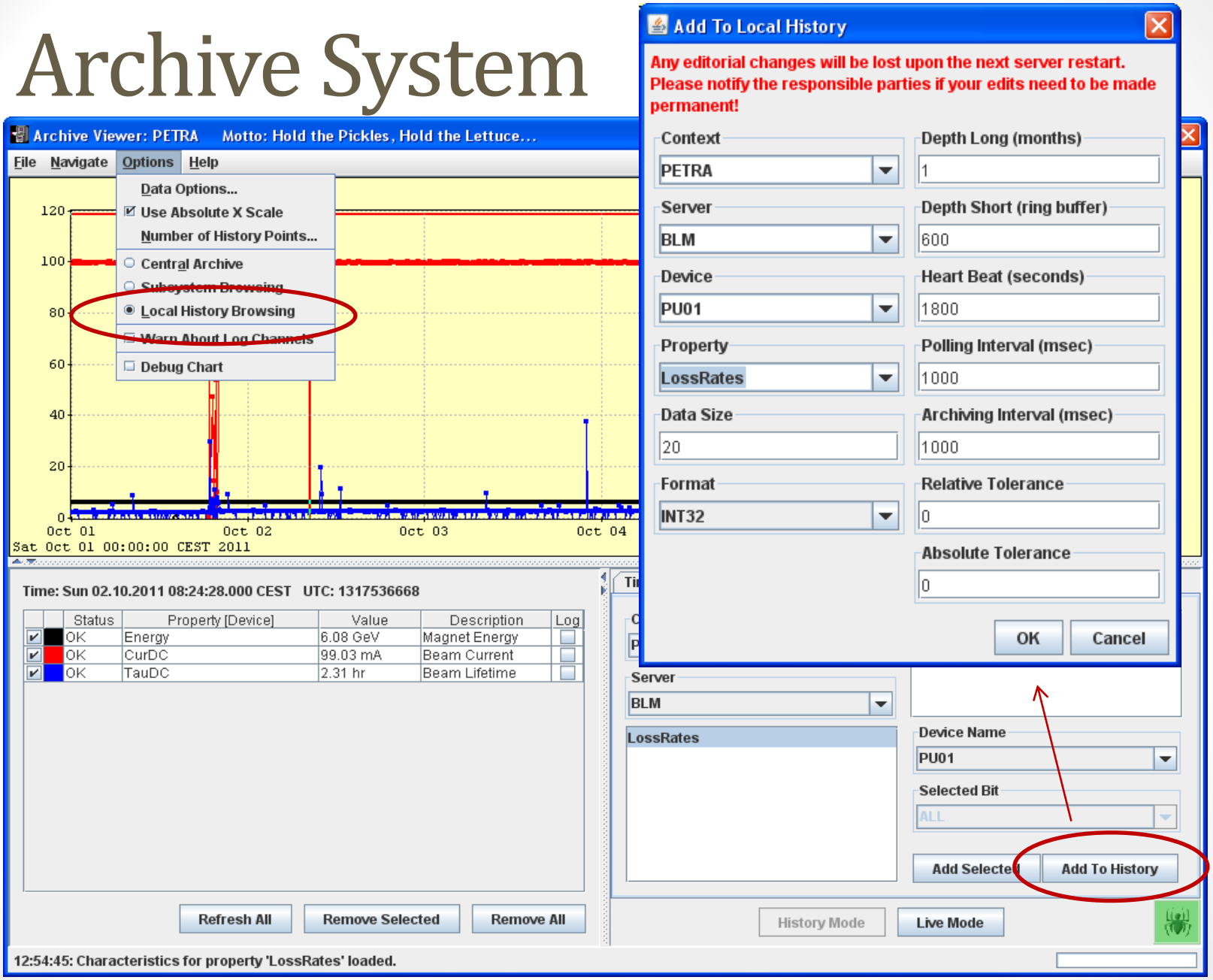
Archive System



Archive System

- Archive Viewer
  - Local history Modes
    - configured local history subsystems !
    - general browsing !

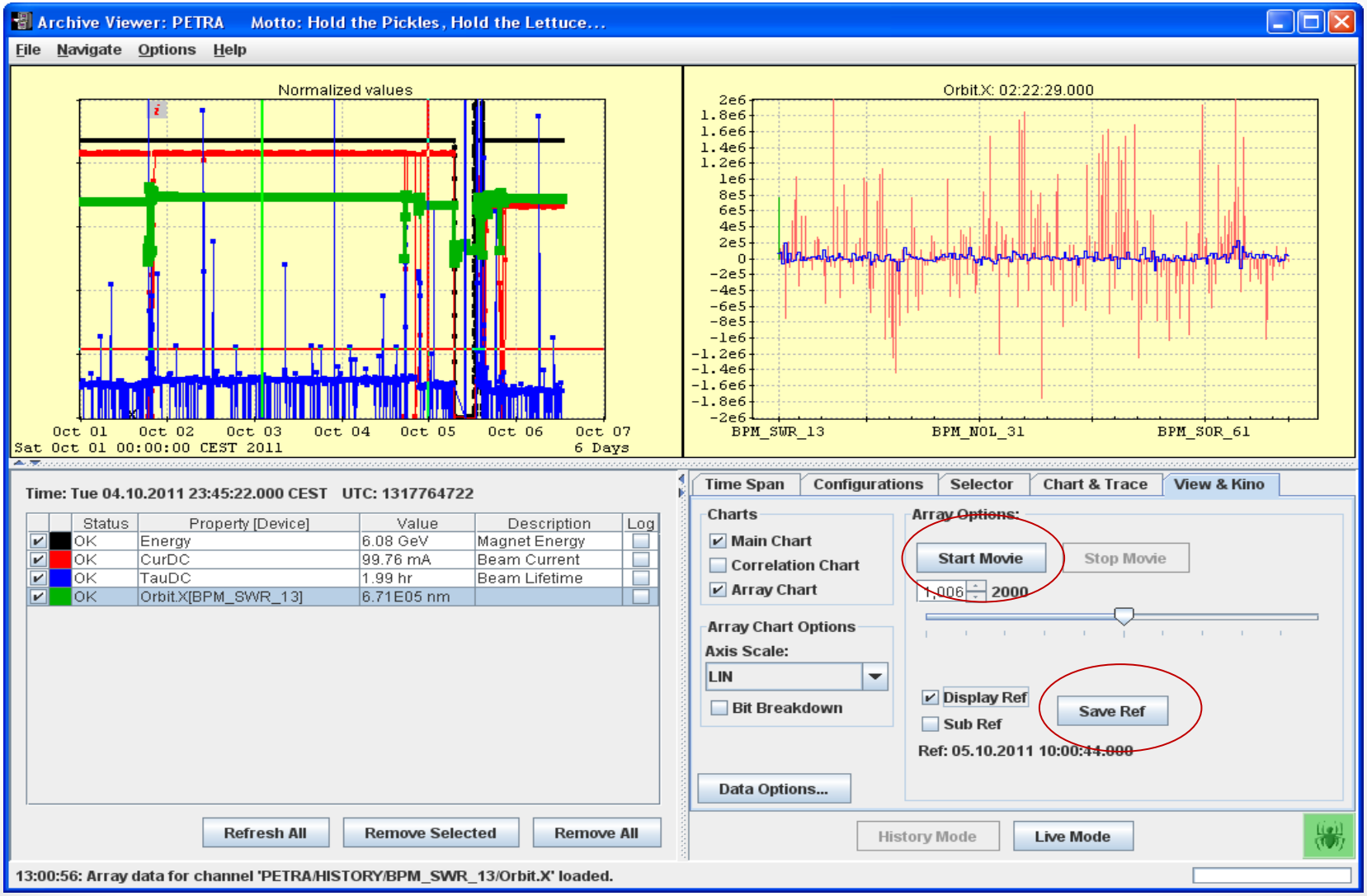
Archive System



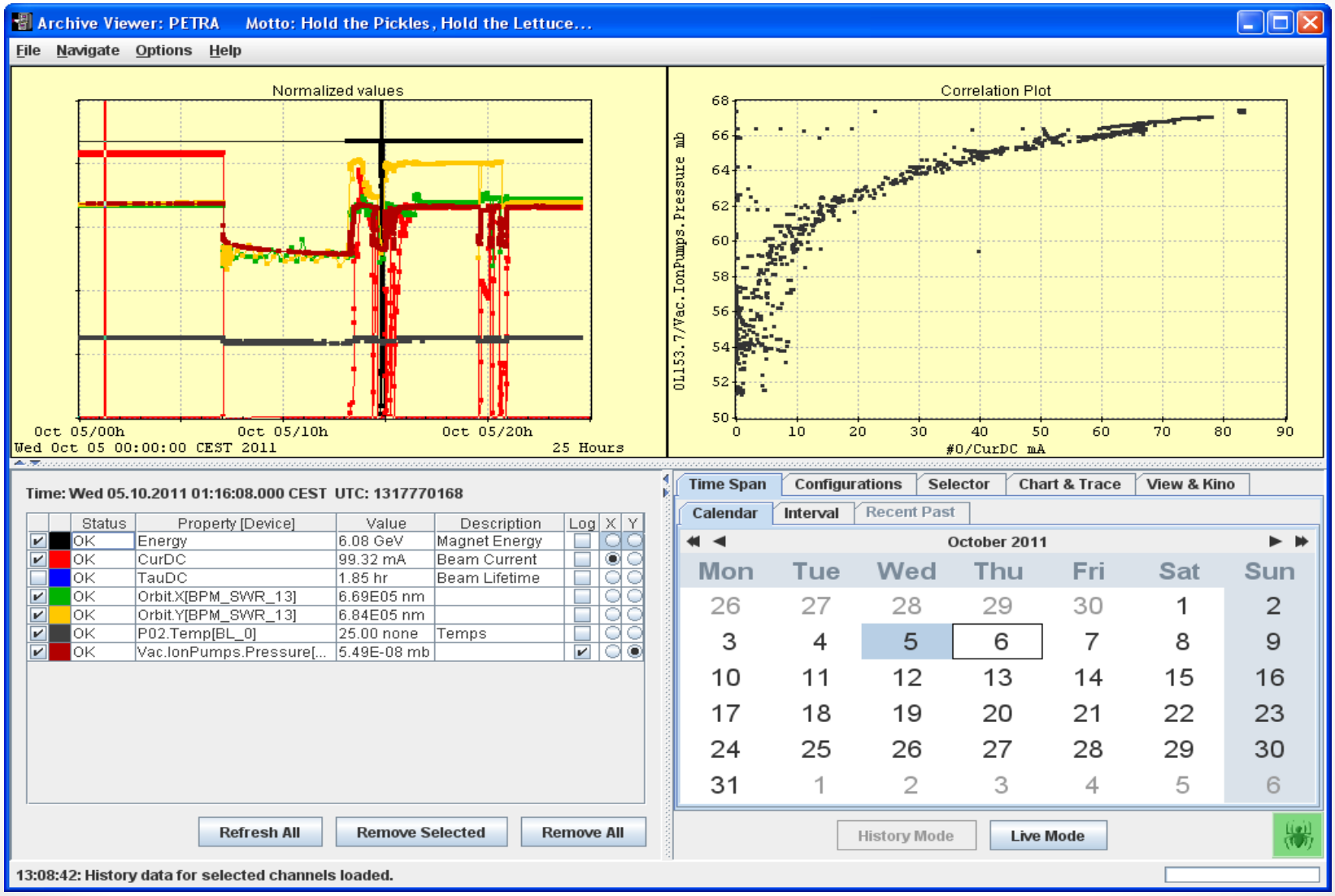
Archive System

- Archive Viewer
  - Array snapshots and movies
  - Correlation plots

Archive System



Archive System

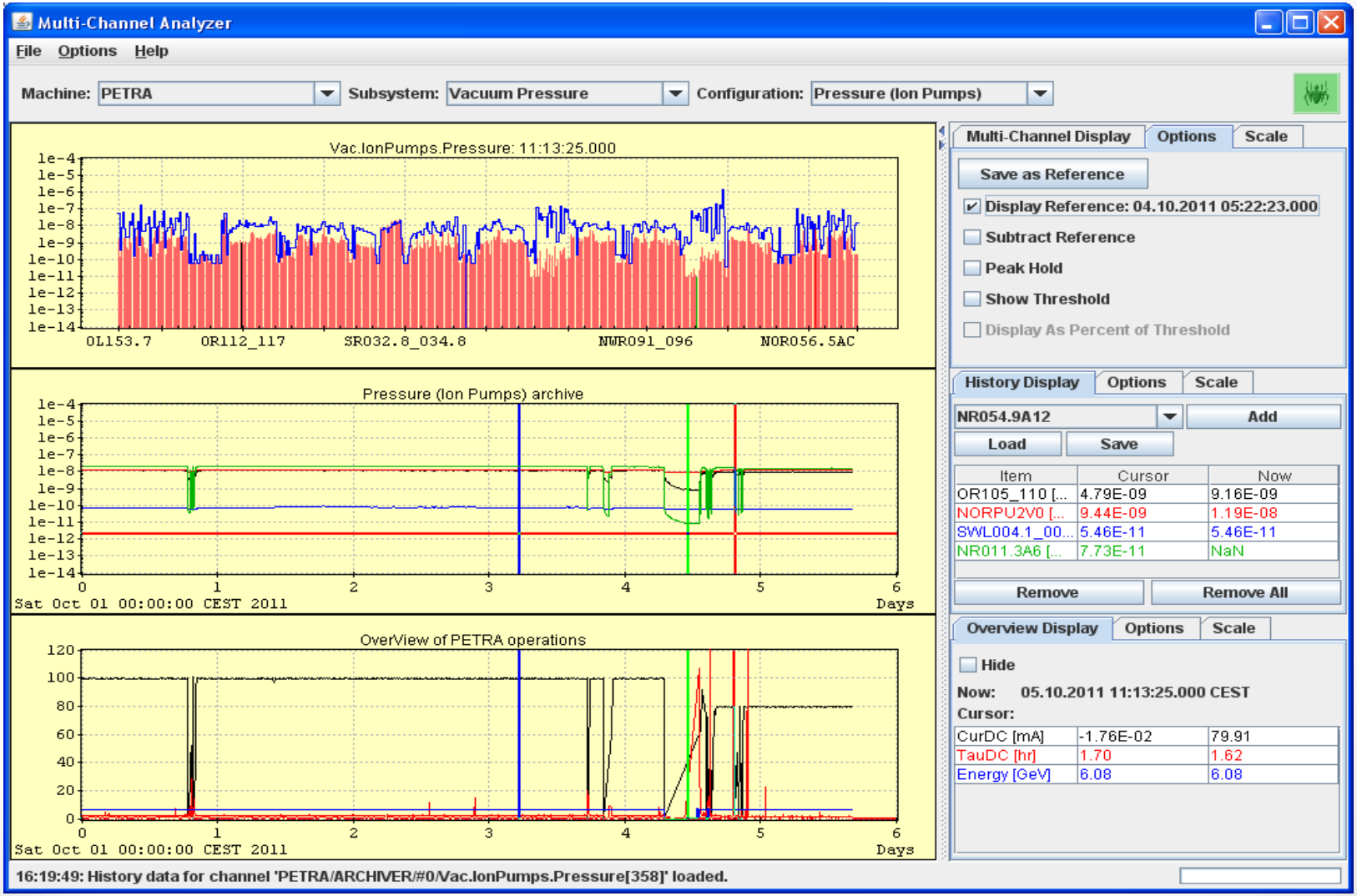


Archive System

- Archive Viewer designed to '*plot*' data
  - *trends*
  - *snapshots*
  - 'text' as string shown in data label and tool tip
- How to show 'complex' data elements ?
  - Can store / retrieve any data type (except CF_HISTORY)
  - e.g. data stored as NAME16FLTINT
    - => a range of these will be an 'array' of NAME16FLTINTs coupled with a set of time/data stamps.
    - how to display this in the archive viewer?
      - "trap and ask?"
        - i.e. ask the user which field he wants to display vs. time ?

Archive System

- Multi-Channel Analyzer uses archive system !



Event System

Event System

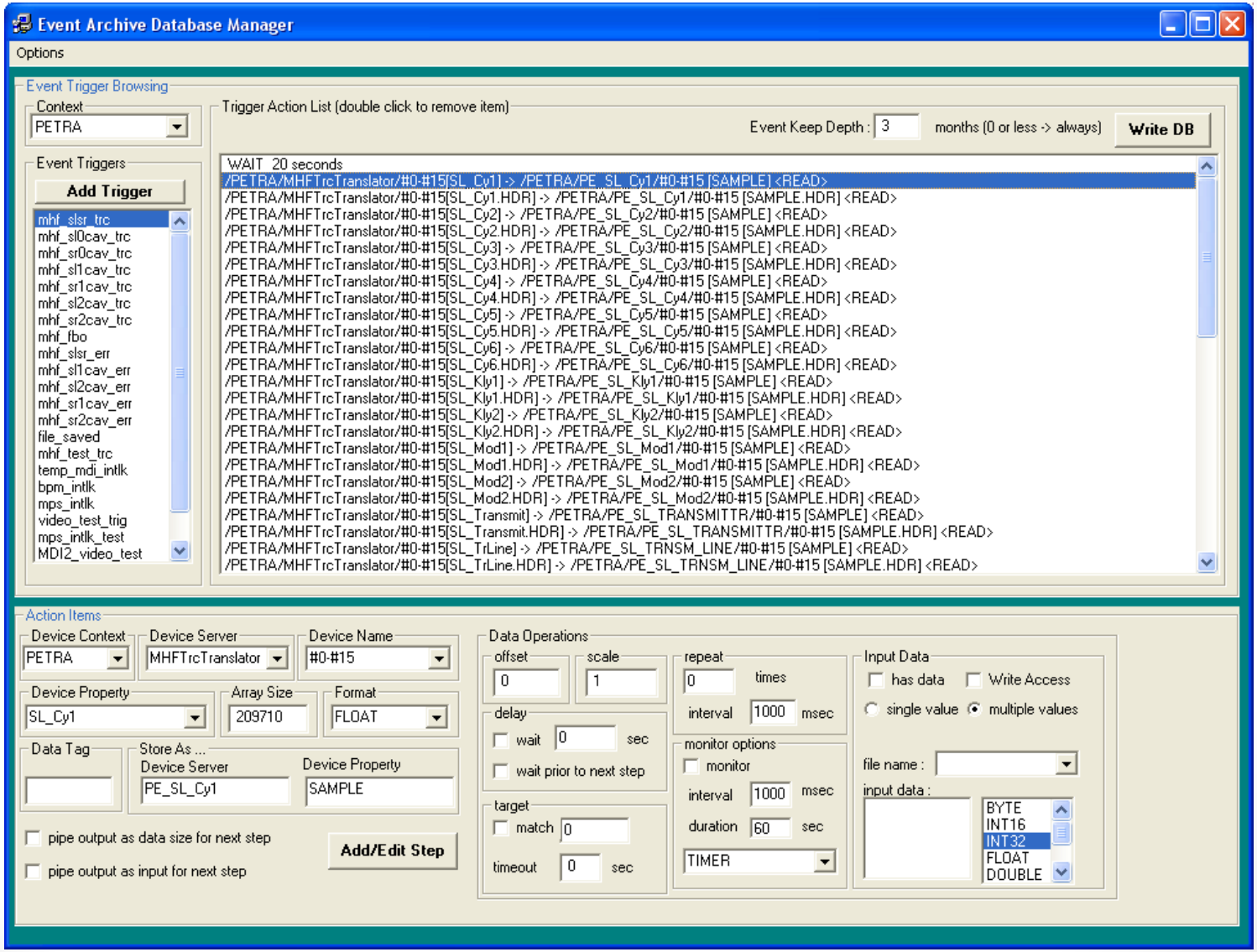
- **Event server**
  - "/SITE/EVENTMGR"
  - provides event numbers to event archive servers
    - defined event window : default 5 seconds
  - record event participants
    - which triggers are associated with an event number ?
- **Event Archive Server**
  - "/<*context*>/EVENTS" and "/<*context*>/EVENTSTORE"
  - reacts to event trigger
    - => event script ("get this, get that, wait here, sent this there", etc.)
  - generally acquires data from *1 or more sources*
  - *post mortem data* stored in hardware or short term local history
  - designed for *transient recorders*
  - Can store any type of event data
    - 'positive' events (e.g. "get orbit settings at injections", etc.)
    - store video train on demand, etc.

Event System

- data retrieval : must specify
  - *event trigger* : (what kind of event ?)
  - *event time* : (when ?)
  - *stored target* : /*context*/*server*/*property*/*device*

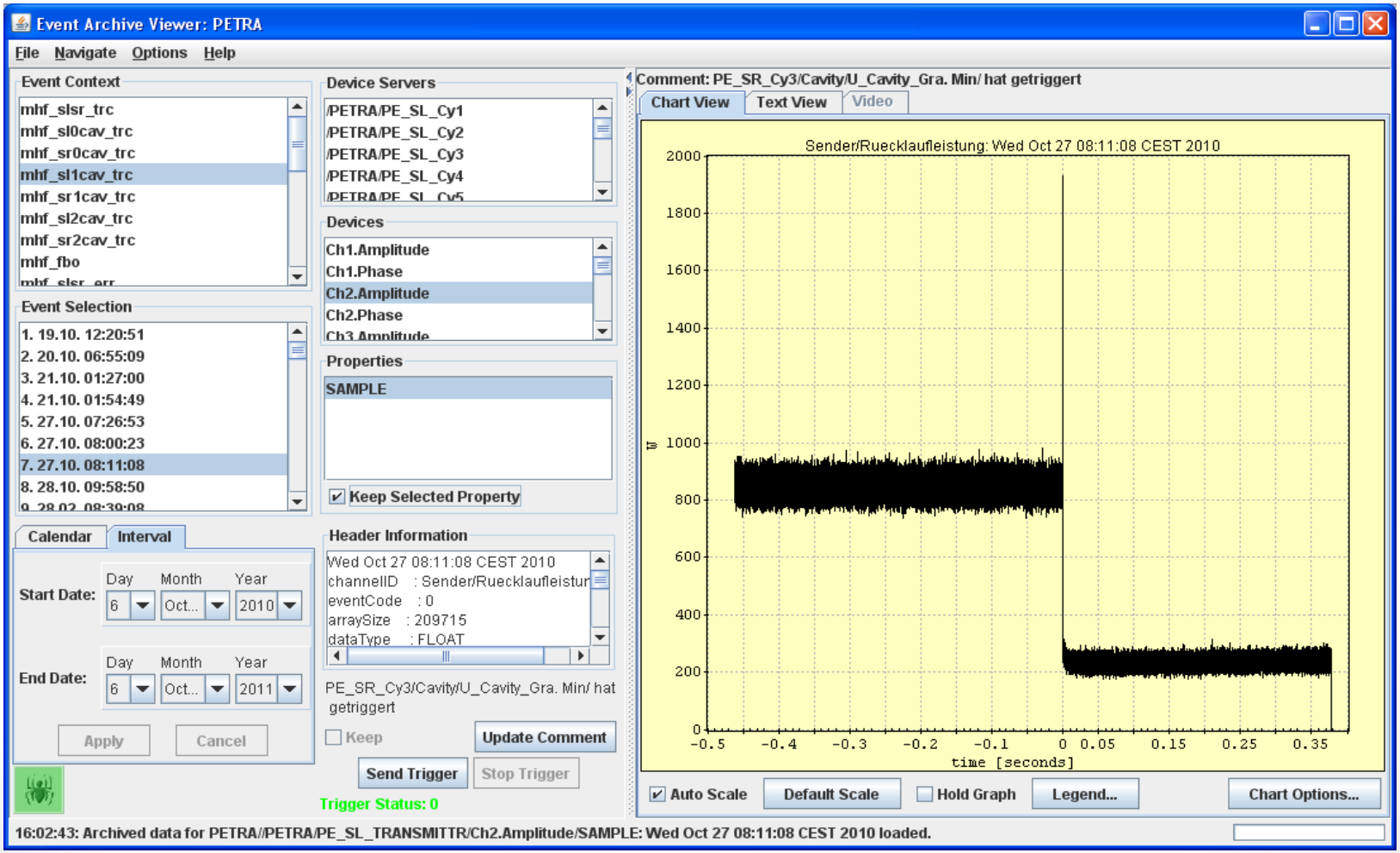
- events defined via event database manager
- event triggers sent via API
  - SendEventTrigger()
    - Sent to /*context*/**EVENTSTORE**/*trigger*[**TRIGGER**]
  - state change triggers are sent automatically from STATE server.
- events can be annotated

Event System



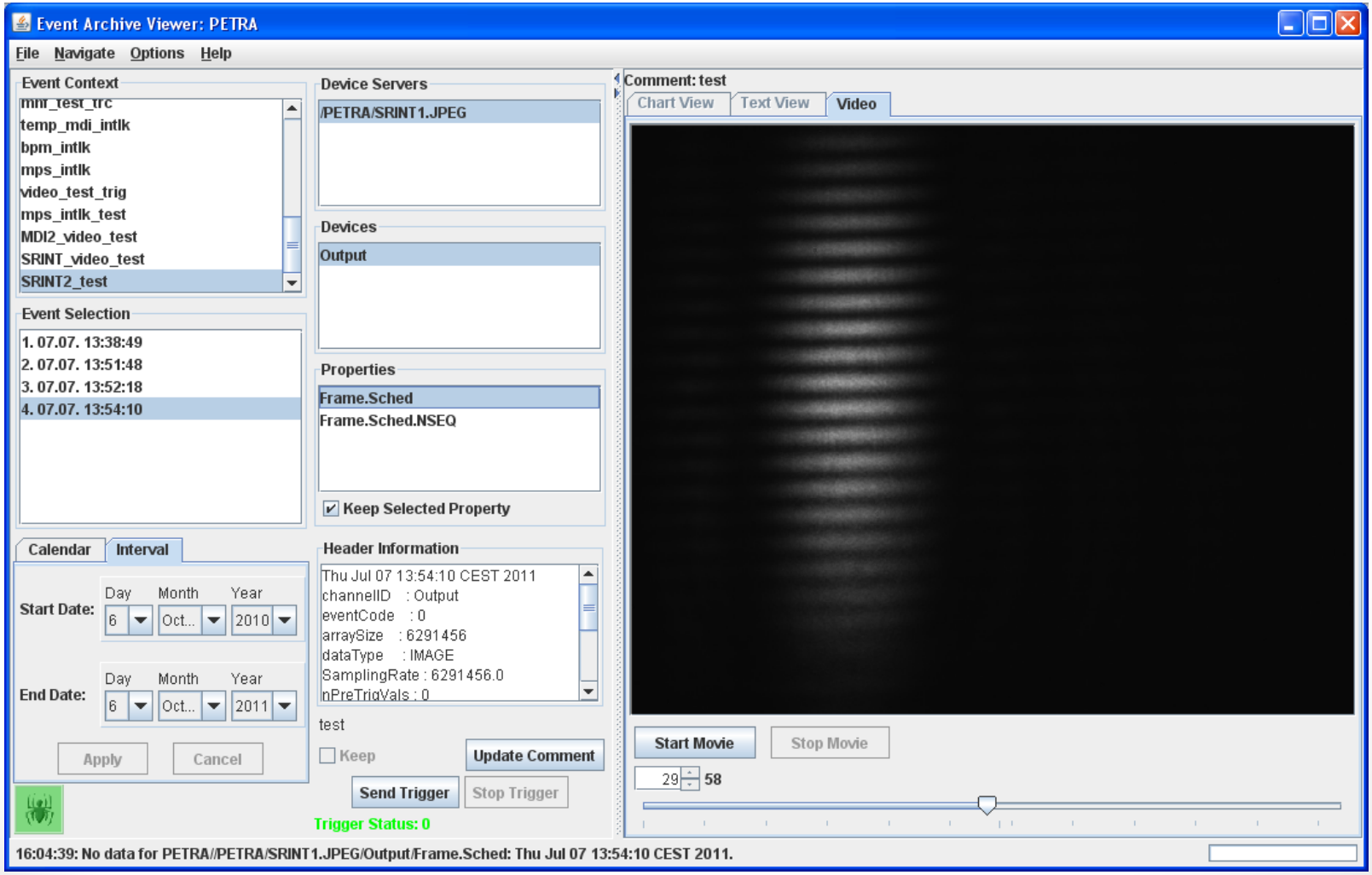
Event System

- Generic event viewer:



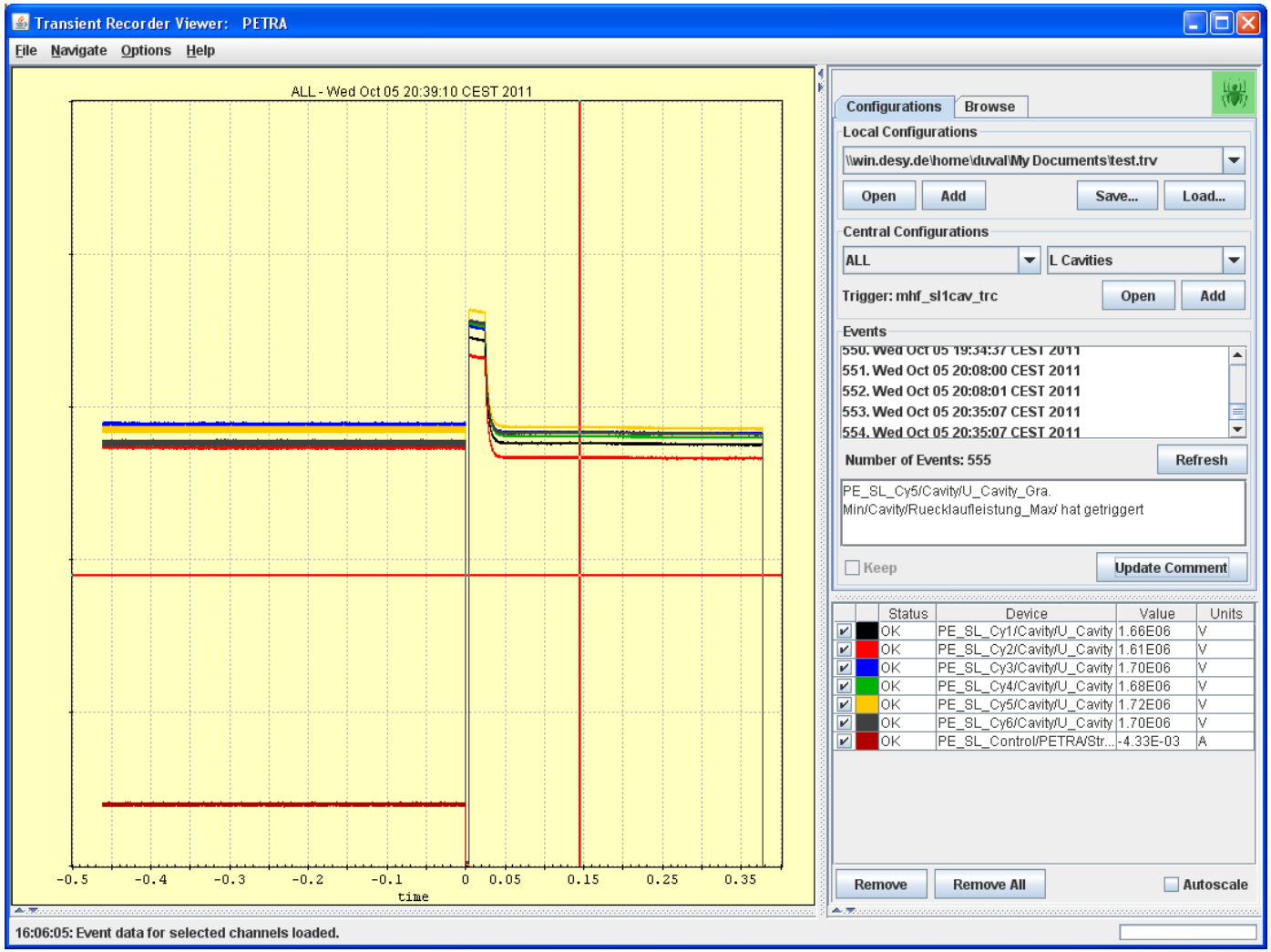
Event System

- Generic event viewer:



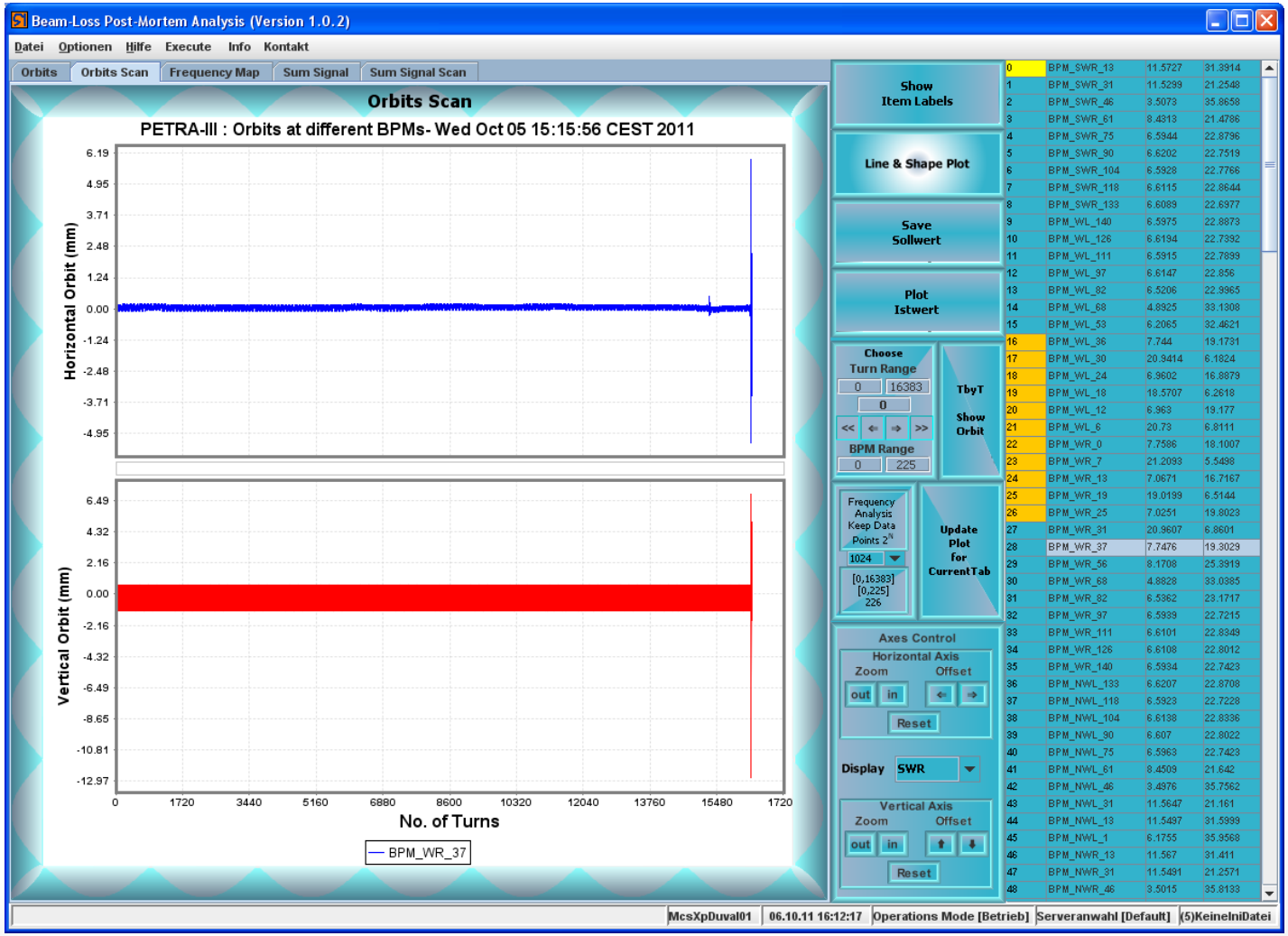
Event System

- Transient Recorder viewer



Event System

- BPM Event viewer



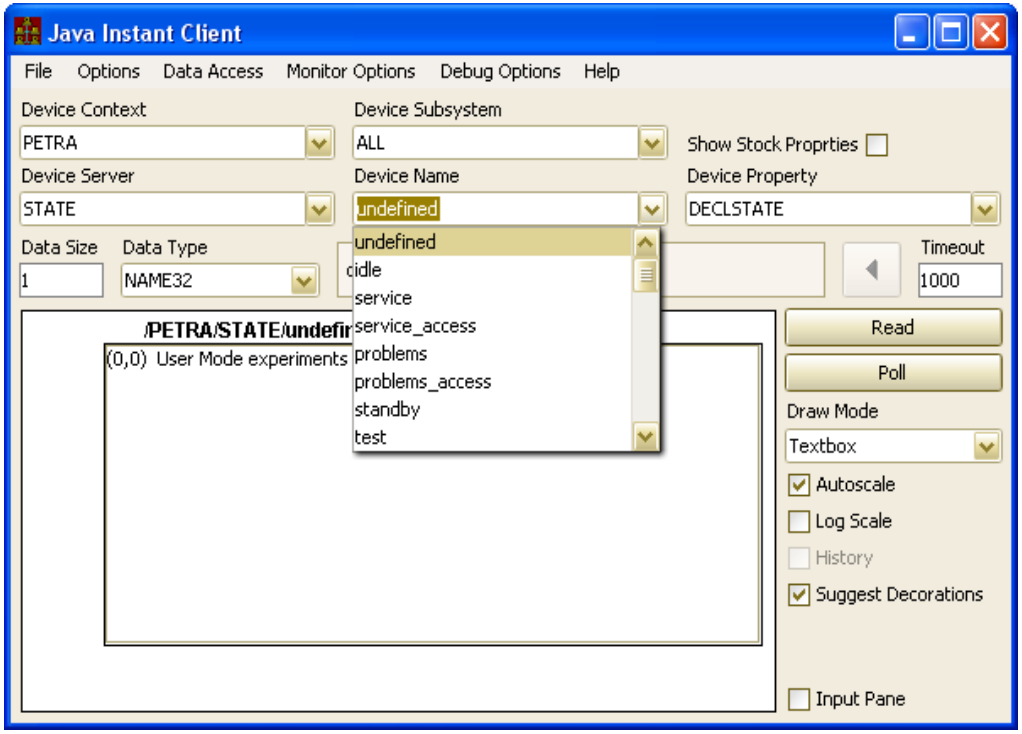
Event System

- Event System **APIs**
  - C-Lib
    - sendEventTrigger()
    - And from the ‘eventtools’ lib: (no documentation yet)
      - GetArchivedEventData()
      - GetArchivedEventList()
      - GetArchiveTriggers()
      - GetArchiveComment()
  - Java
    - Numerous calls from eventArchive.jar
  - MatLab
    - tine_eventdata
  - command line
    - teventdata

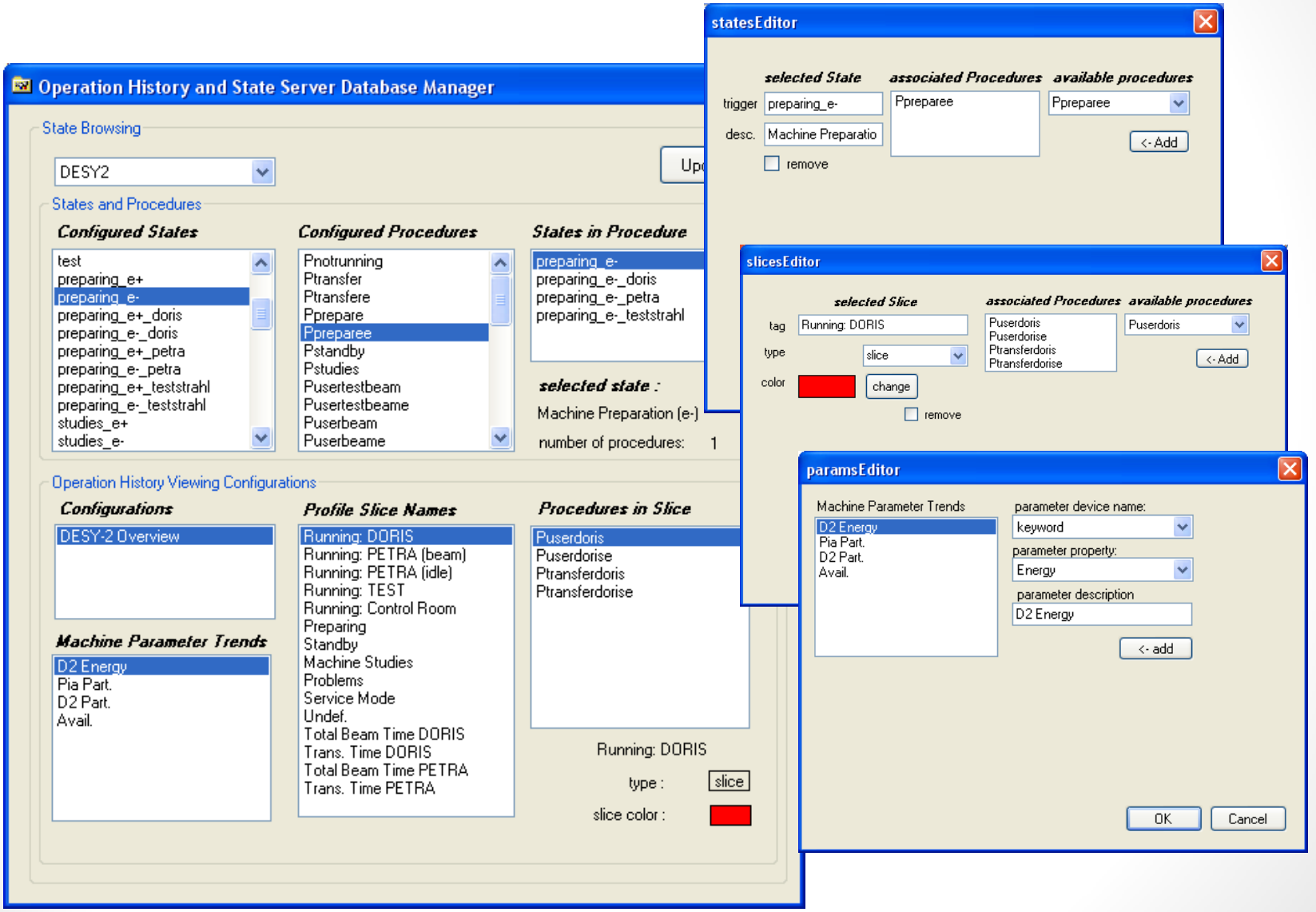
State Server

State Server

- "/<context>/**STATE**"
  - accepts state change commands
    - forwards trigger to EVENTSTORE
  - counts time spent in given state
  - Manages operation statistics profiles (i.e. those 'pie slices')

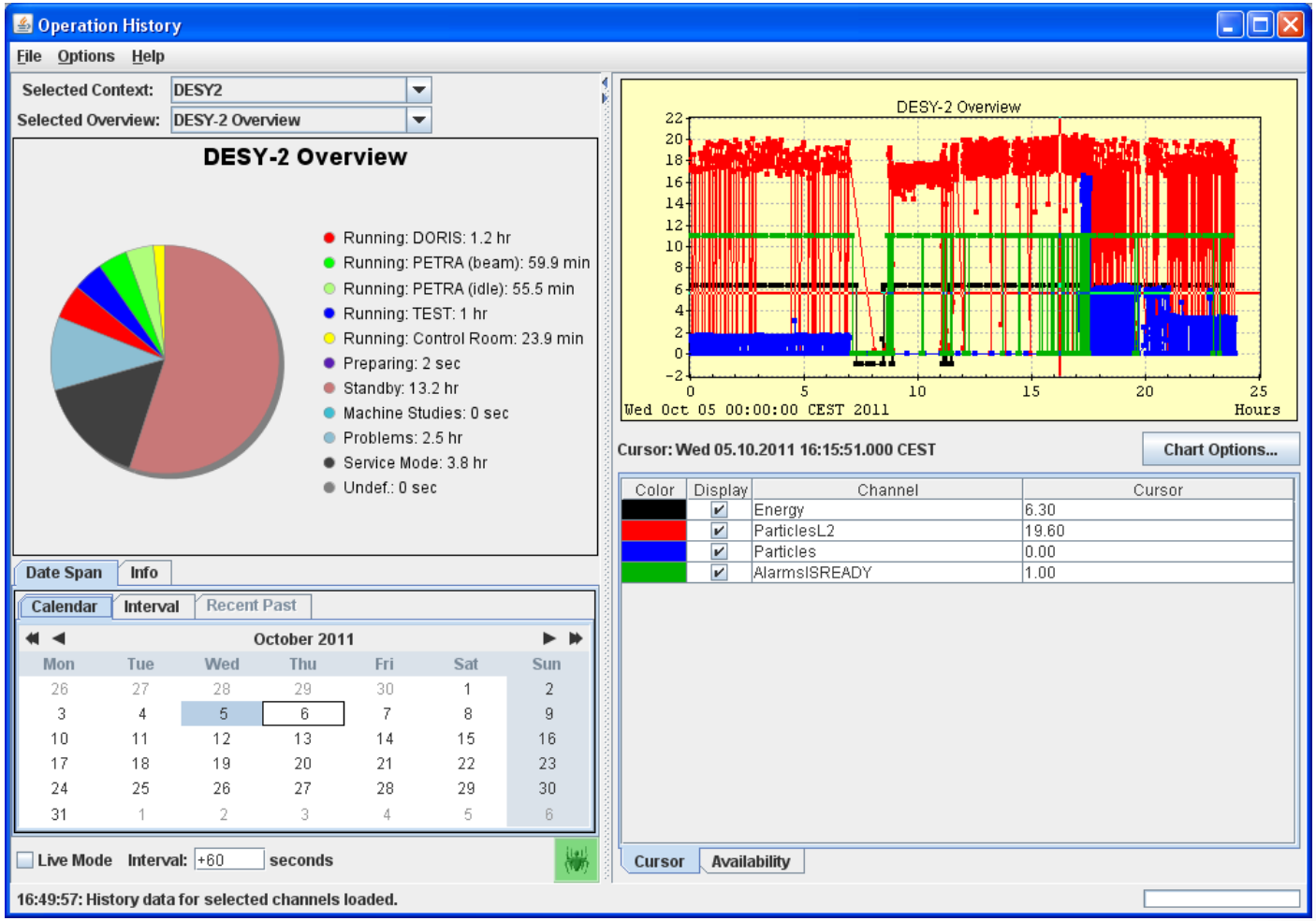


State Server



State Server

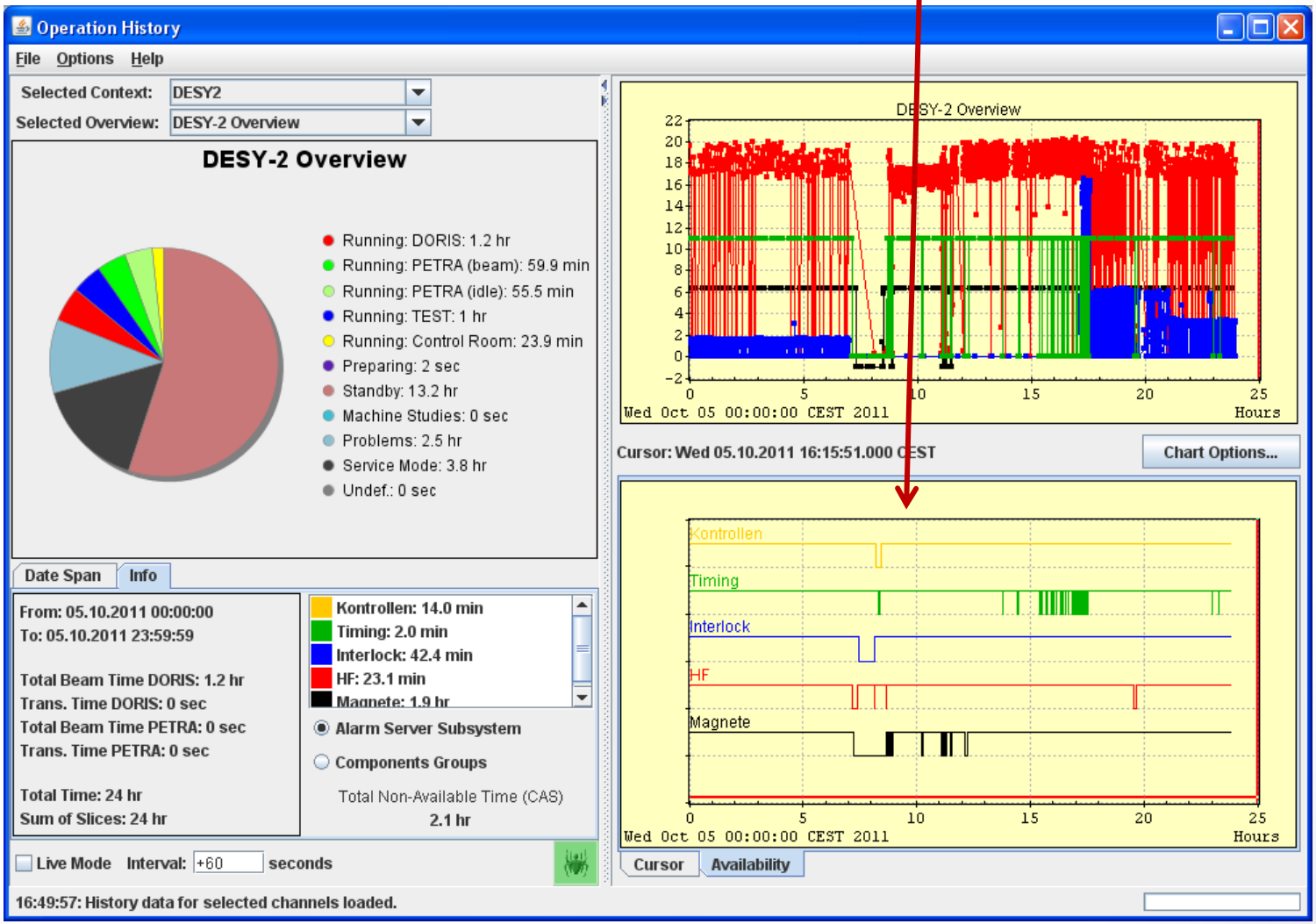
- Operation Statistics



State Server

- Operation Statistics

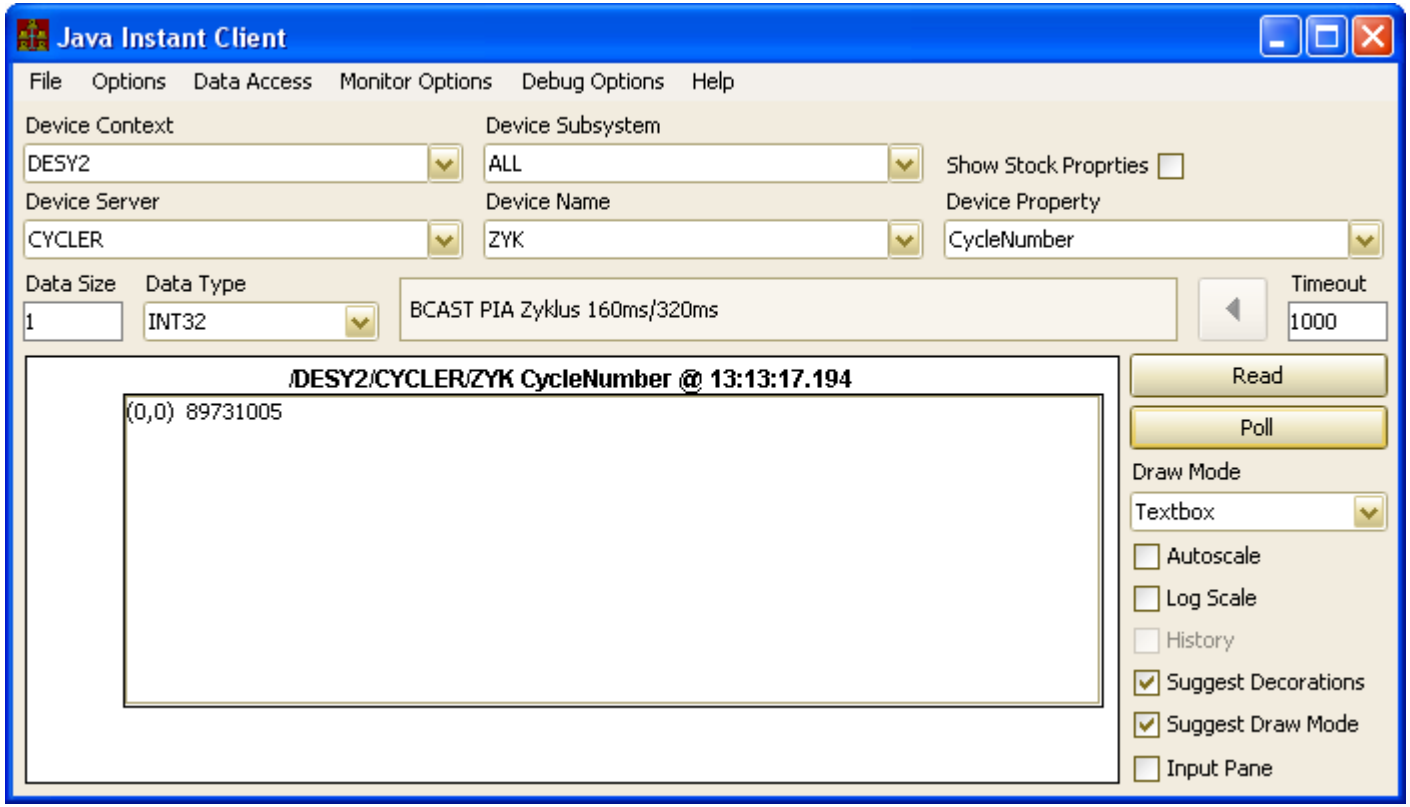
"ALARMSTATE" used for availability statistics !



The CYCLER

The CYCLER

- automatic system data stamps via server “CYCLER” and property “CycleNumber” :

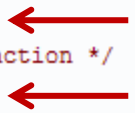


The CYCLER

- If a cycler exists for '*my*' context then a server will :
  - listen for cycle multicasts
  - apply cycle number to the data stamp of all property data being accessed.
  - support cycle trigger functions
    - initiate/synchronize hardware i/o
    - supply cycler delay offset (if necessary)

The CYCLER

```
typedef void (*HDWIOFCNP)(int);
void hdwIoCycle(int cycle)
{
  /* read relevant hardware (here we just print something out) */
  printf("read hardware for cycle %d\n",cycle);
}
void onCycleTrigger1(int cycle,int cc,void *ref)
{
  printf("received cycle %d <%d>\n",cycle,cc);
  cycleNumber = cycle;
}
void onCycleTrigger2(int cycle,int cc,void *ref)
{ /* call the referenced function */
  if (cc == 0) ((HDWIOFCNP)ref)(cycle);
}
void PreSystemInit(void)
{
  SetSystemUseDataProtection(TRUE);
  SetPacketMTU(64000);
  RegisterFecInformation("CYCCATCH.8","TST","TEST","Cycle catcher tester","My Office","none","me",8);
}
void PostSystemInit(void)
{
  /* register the equipment module: */
  RegisterEquipmentModule("CycleCatcher",EQMTAG,1,tsteqm,tstinit,tstbkg,100,NULL);
  /* register a cycle trigger function with no scheduling and no reference */
  RegisterCycleTriggerFunction(onCycleTrigger1,EQMTAG,NULL,NULL);
  /* register another cycle trigger function with a scheduled property and a reference to another function */
  RegisterCycleTriggerFunction(onCycleTrigger2,EQMTAG,"CycleNumber",(void *)hdwIoCycle);
}
```



The CYCLER

Cycle Delay and Cycle Offset :

I 'know' I'm going to be late getting my data ready …

void SetSystemStampDelay (int cycleDelay)

Establishes the system cycle delay.

If a server's context has a registered 'CYCLER' then all read data will be tagged with the incoming system cycle number. If it is known a priori that due to hard i/o latency the application of the cycle tag needs to be delayed by some value, then this routine may be used to establish such a cycle delay value (in milliseconds).

Parameters:

cycleDelay is the desired cycle delay (milliseconds), which will must elapse before the incoming cycle number from the registered CYCLER is to be applied to all readback data. (default = 0).

See also:

GetSystemStampDelay

void SetSystemStampOffset (int cycleOffset)

Establishes a system cycle offset.

If a server's context has a registered 'CYCLER' then all read data will be tagged with the incoming system cycle number. If it is known a priori that due to hard i/o latency the cycle tag needs to be offset by some value, then this routine may be used to establish such an offset.

Parameters:

cycleOffset is the desired cycle offset (counts) to be applied to the incoming cycle number from the registered CYCLER. (Default = 0).

See also:

GetSystemStampOffset

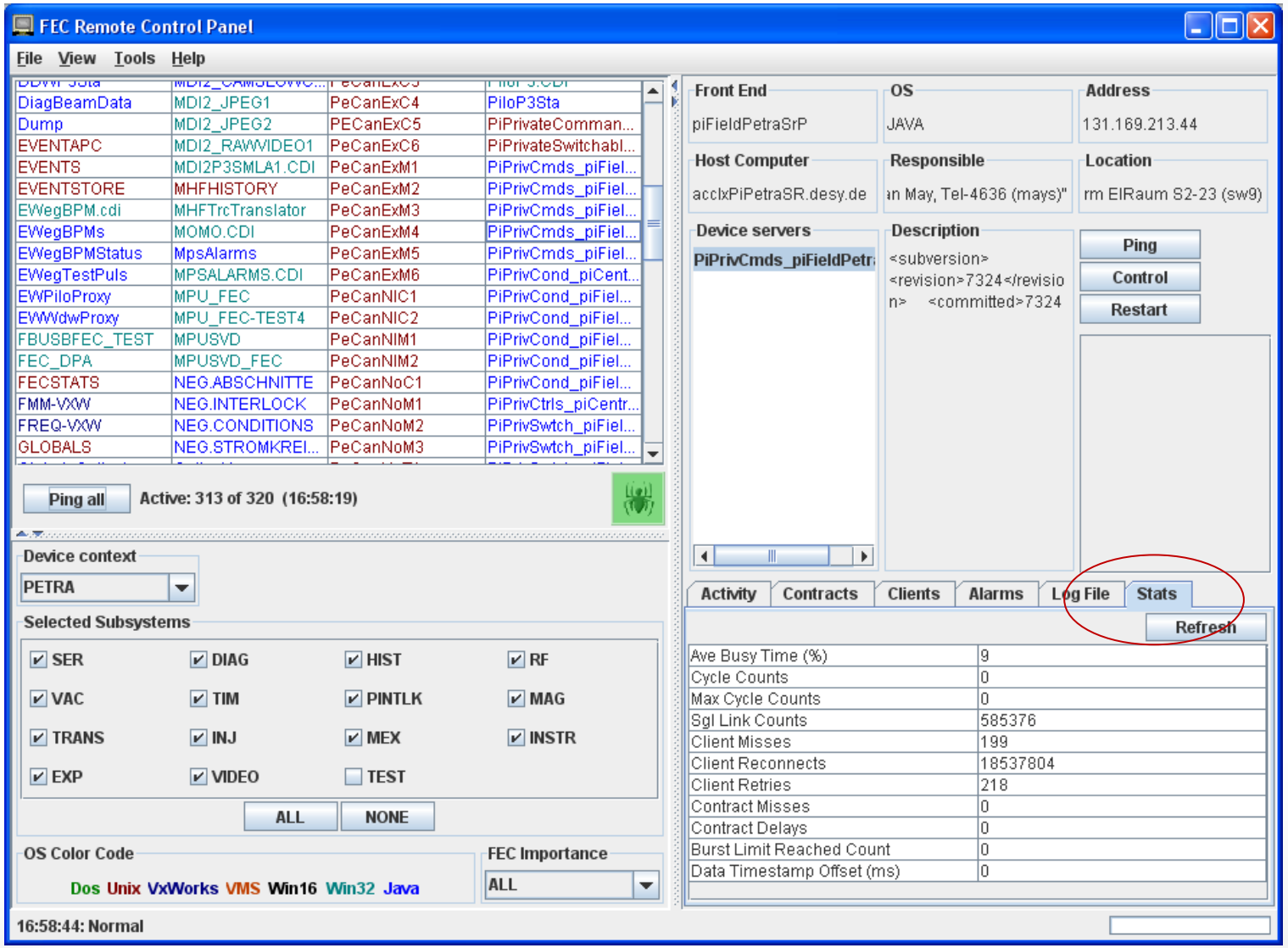
I 'know' the incoming cycle is 'off' by some number of counts …

FEC Statistics

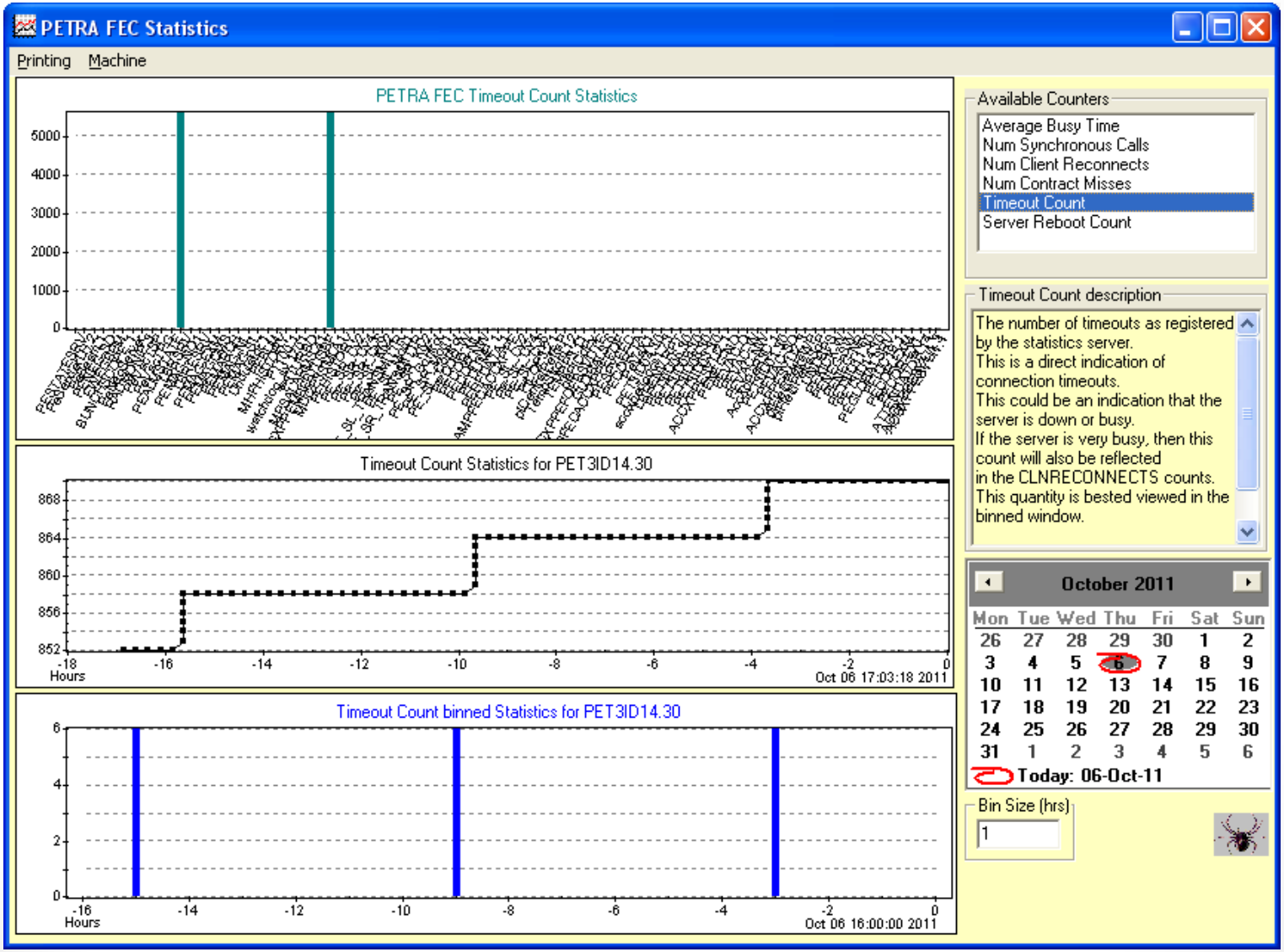
FEC Statistics

- "/<*context*>/**FECSTATS**"
  - maintains general statistics of all important servers in a context
  - stock property "SRVSTATS" from each server
  - reboot counts from ENS
  - timeouts (as determined by the FEC stats server itself)

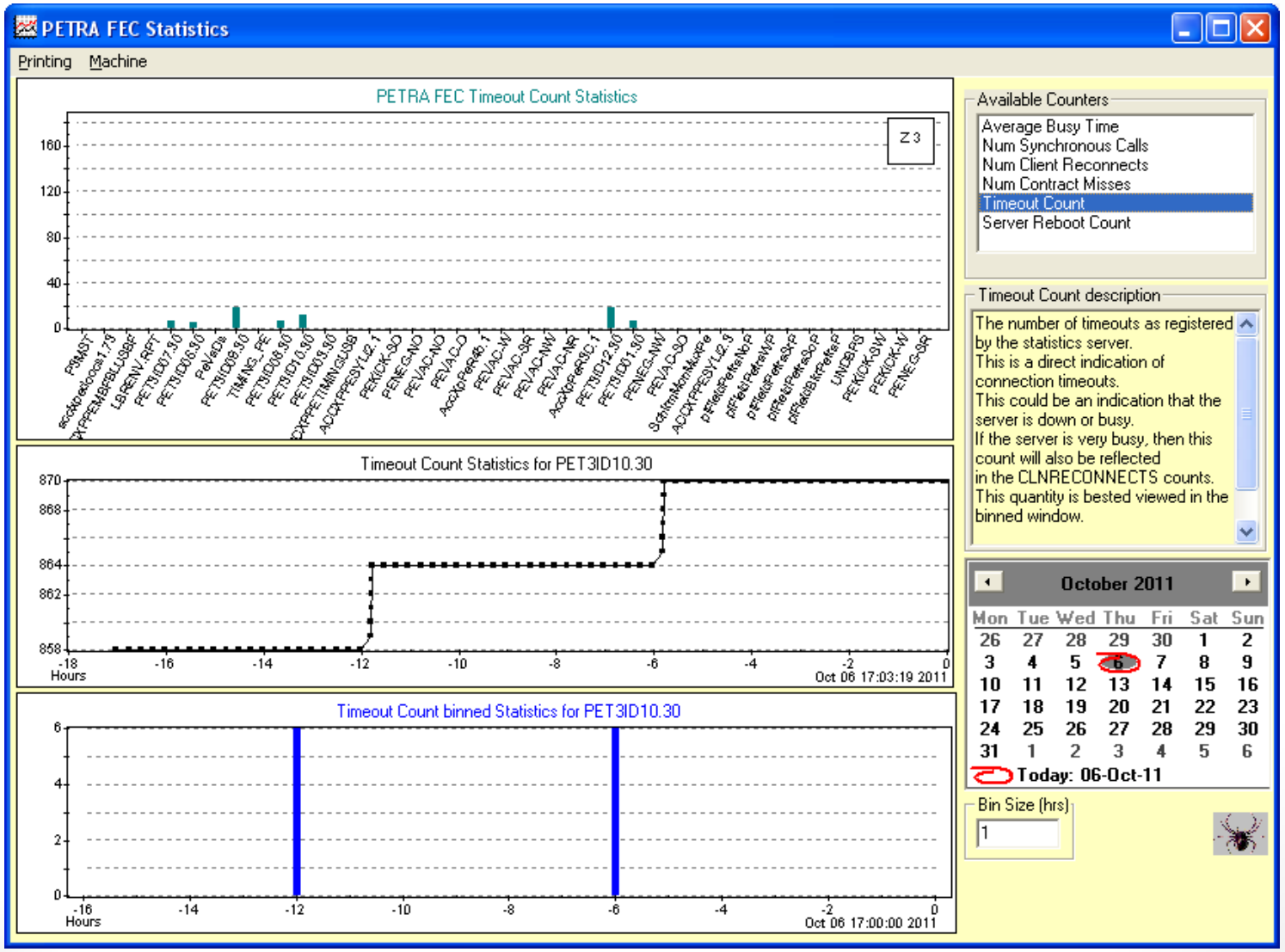
FEC Statistics



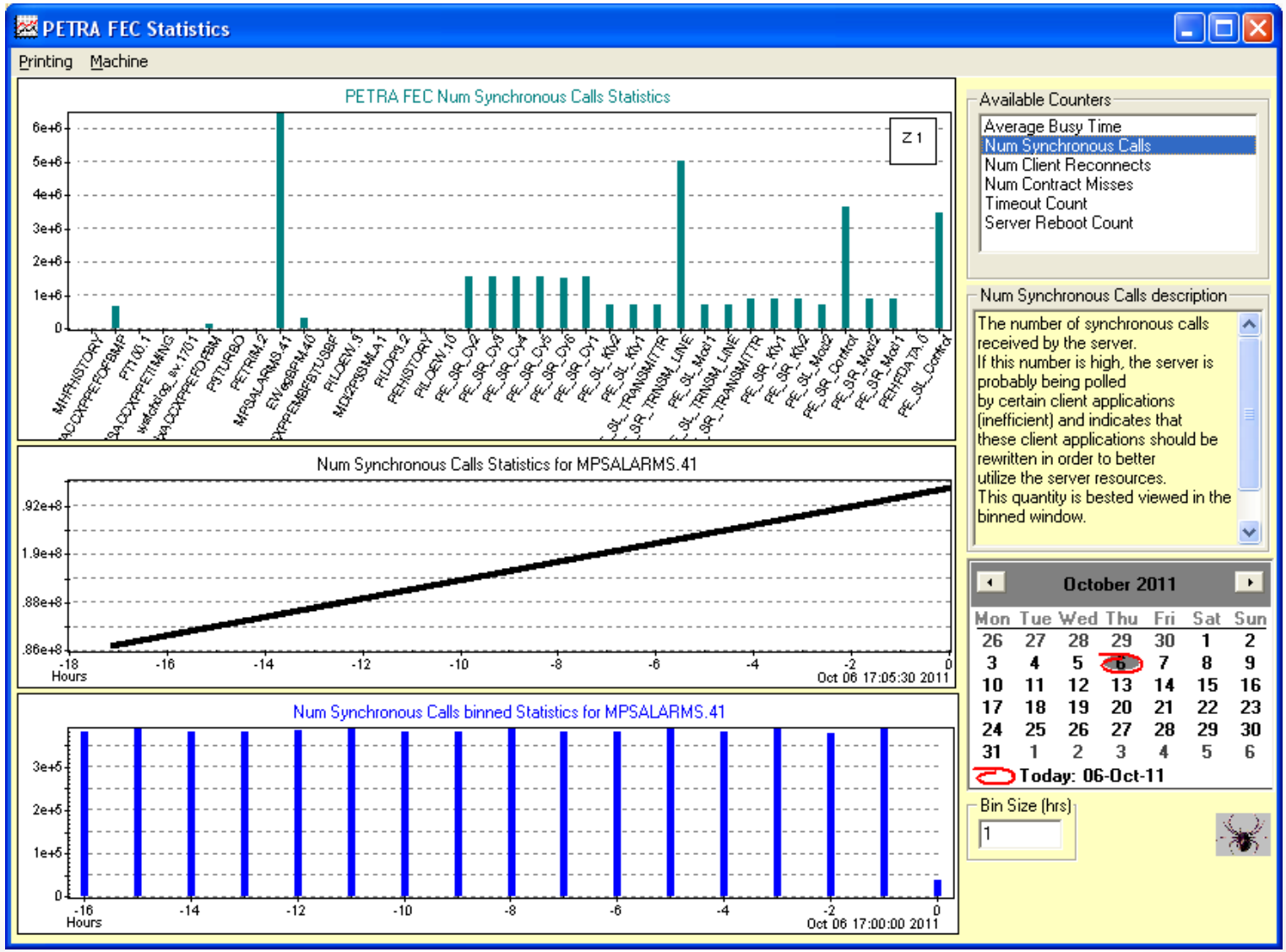
FEC Statistics



FEC Statistics



FEC Statistics

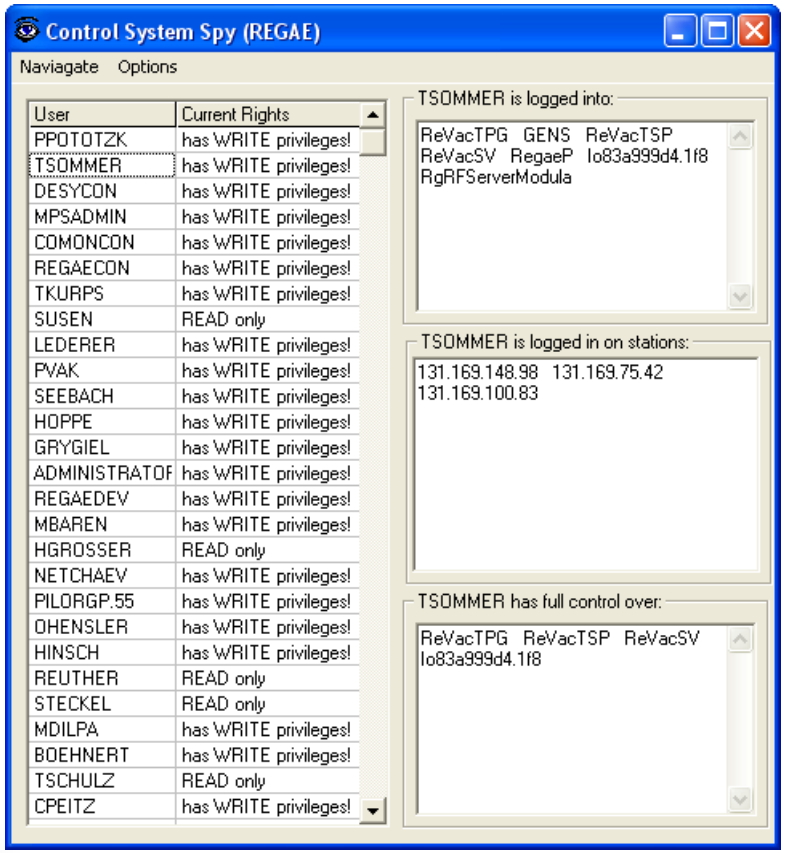


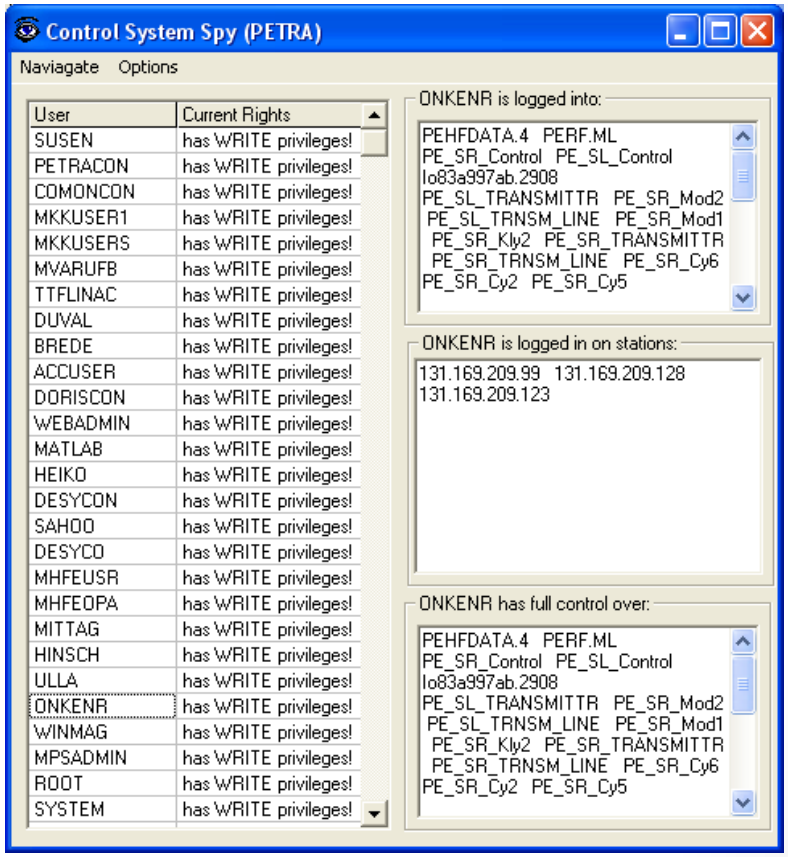
Control System SPY

Control System SPY

- "/<*context*>/**CSSPY**"
  - who's watching who (big-brother style).
  - scans all important servers in a context
  - makes use of stock properties:
    - "CLIENTS"
    - "SRVCOMMANDS"
    - "USERS"

Control System SPY





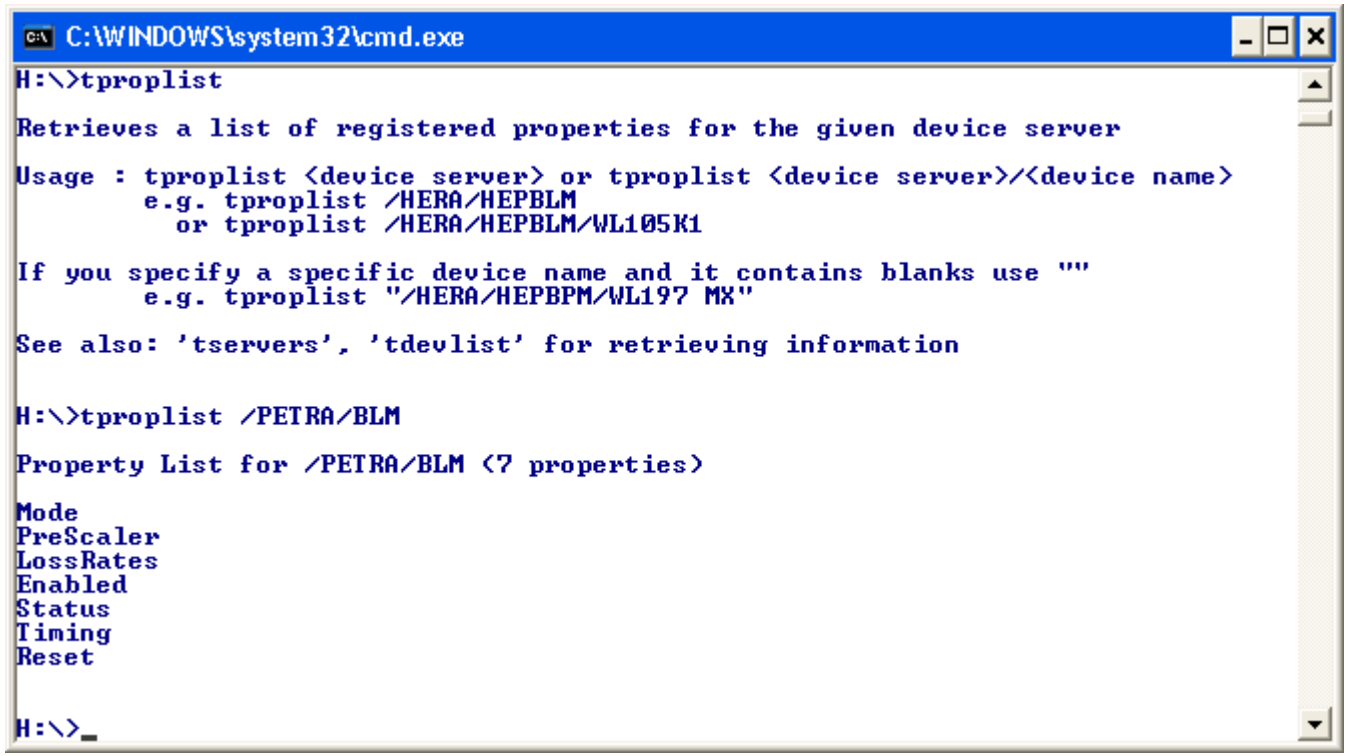
Command Line Utilities

Command Line Utilities

- See http://tine.desy.de (command line utilities)
- Query Utilities:
  - tservers
    - queries the Equipment Name Server ENS for registered servers
  - tdevlist
    - queries a server for its devices
  - tproplist
    - queries a server for its properties
  - tinfo
    - queries a server for property information

Command Line Utilities

e.g.

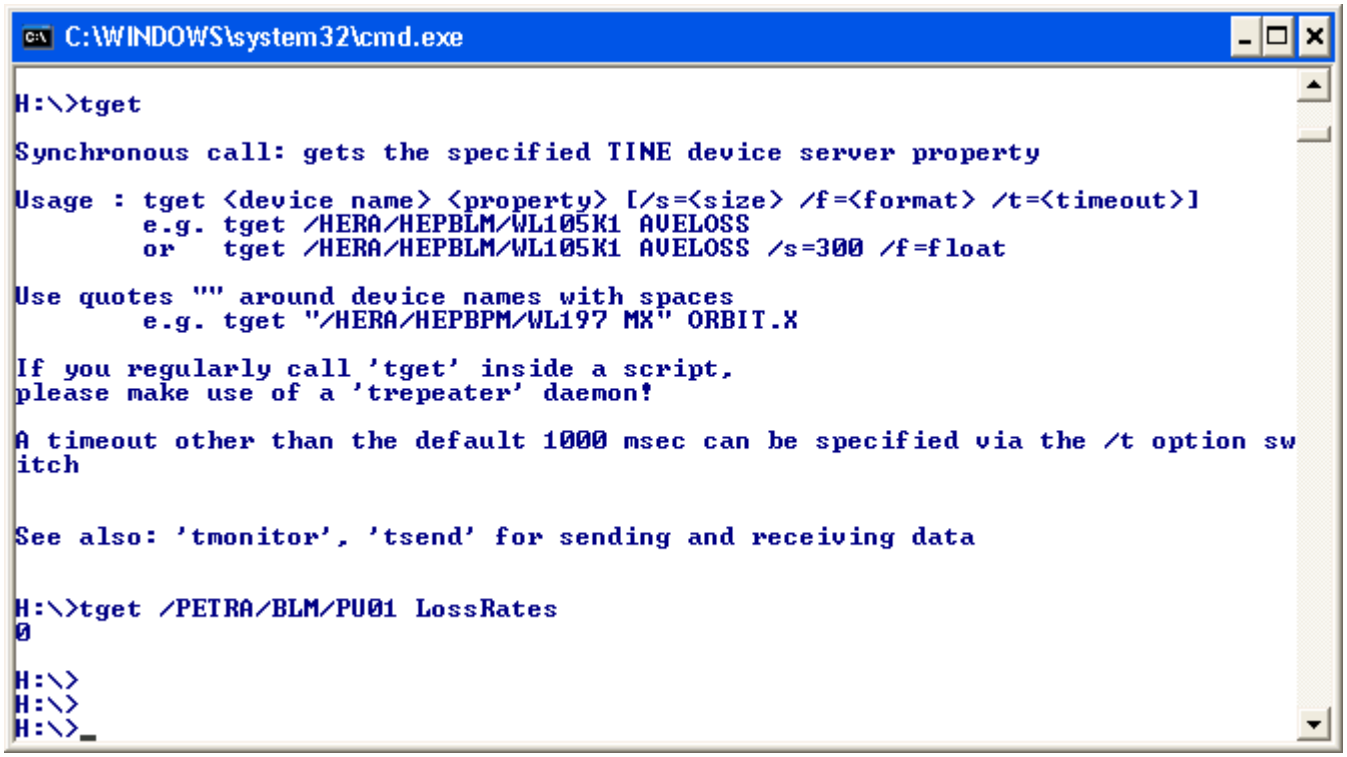


Command Line Utilities

- Data Acquisition:
  - tget
    - synchronous read-only call to server
  - tmonitor
    - asynchronous read-only poll to server
  - tsend
    - synchronous write call to server
  - tputget
    - synchronous atomic write/read for attibute style properties
  - tsendrecv
    - synchronous atomic write/read for call style properties)
  - twait4target
    - monitor readback from a server until a requested target is met

Command Line Utilities

e.g.

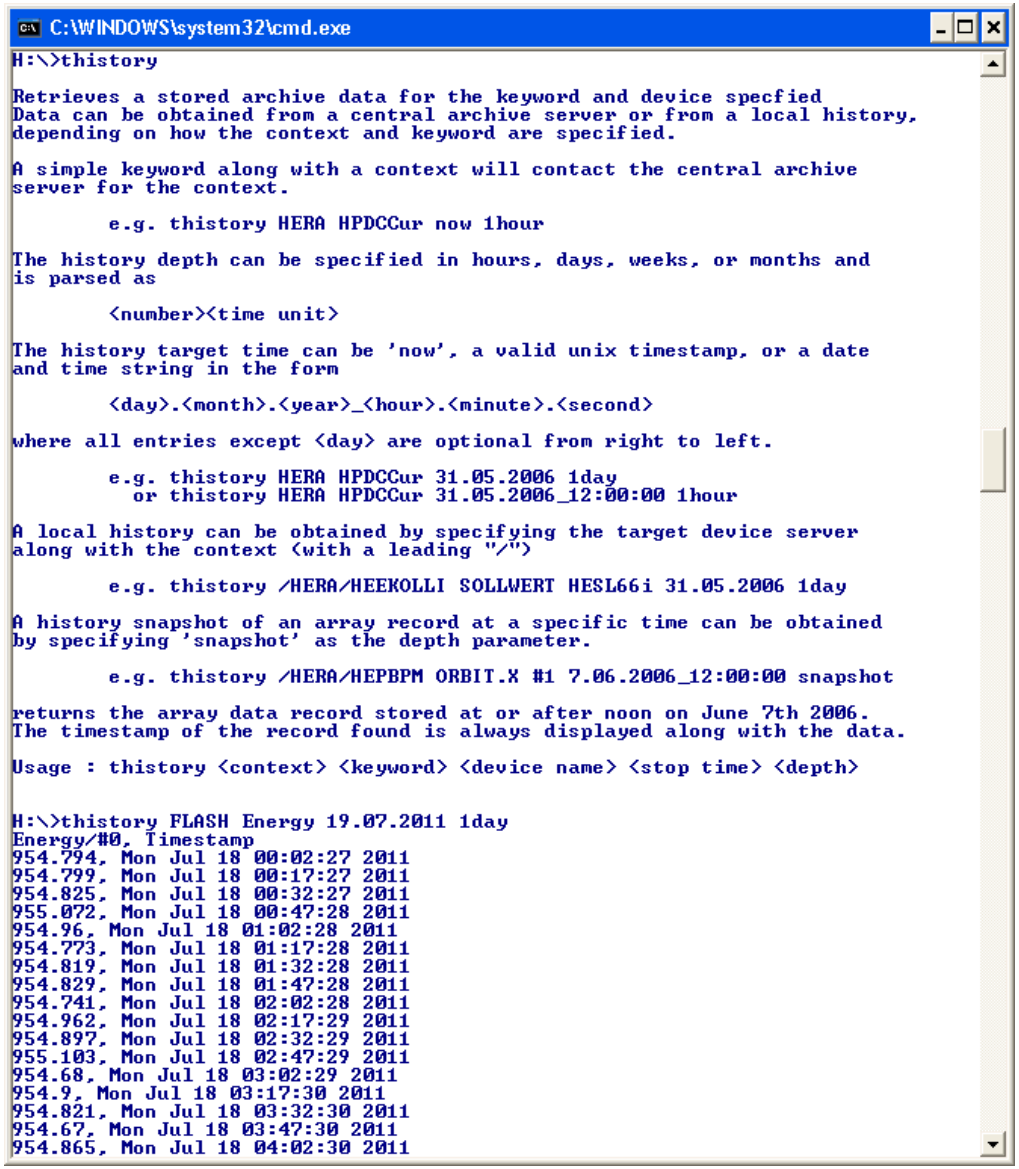


Command Line Utilities

- Services:
  - thistory
    - queries the archive server for archive data
  - talarms
    - queries a specific server for its current alarm list
  - tglobals
    - monitors and displays the current globals for the context given.
  - teventdata
    - queries the event archive server for event data

Command Line Utilities

e.g.

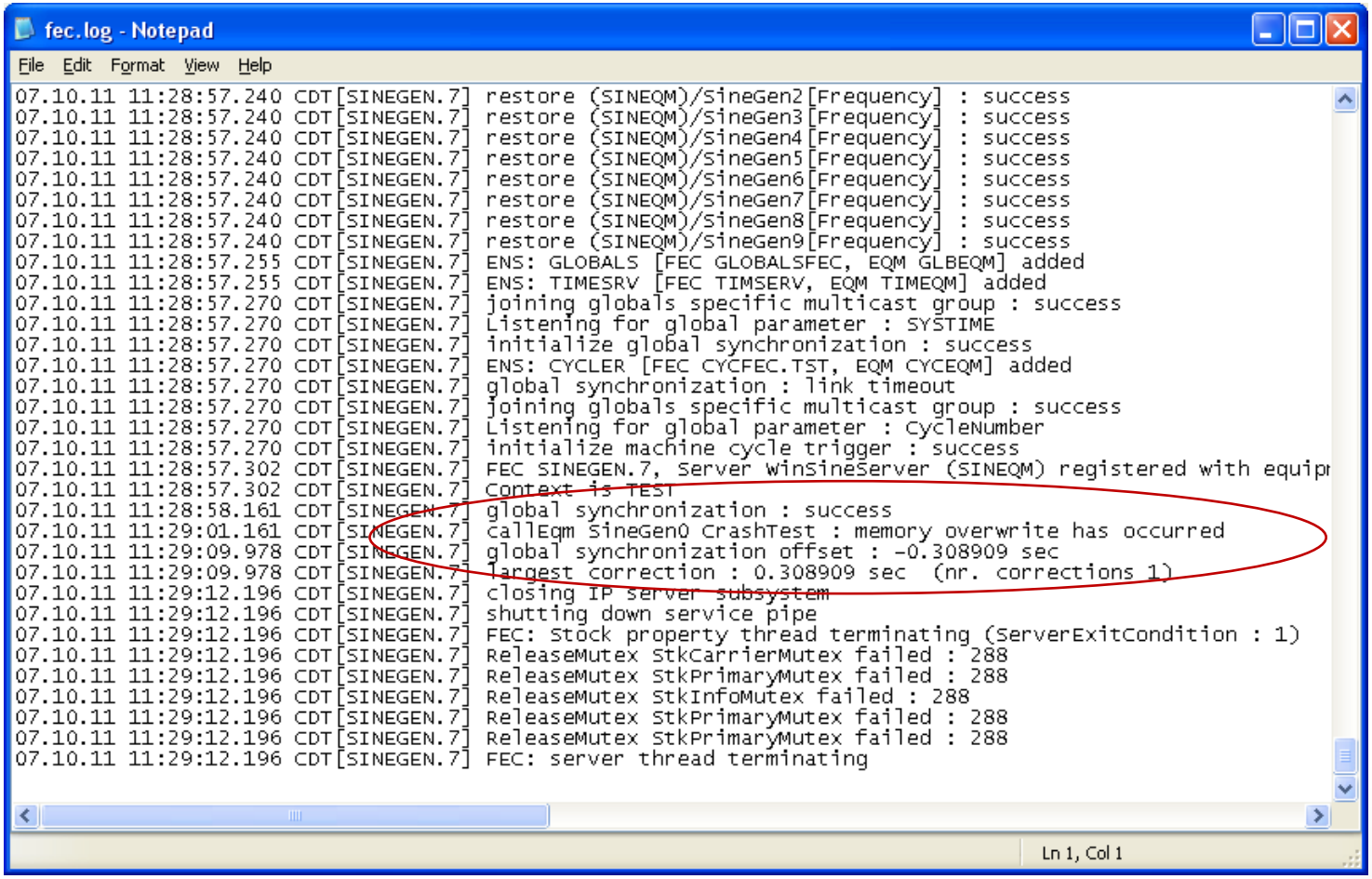


Debugging Utilities

Debugging Utilities

- See http://tine.desy.de -> “Remote Debugging Tools”
- Server Problems
  - *In the unlikely event of a* Crash
    - check core dumps, stack traces, etc.
    - report any TINE bugs to http://tinetracker.desy.de
    - note: segmentation faults or exceptions often occur in user code within dispatch routines !
    - check the last few entries in fec.log
      - detected memory overwrites in dispatch routines are logged (win32).
      - TINE configuration problems are logged.

Debugging Utilities

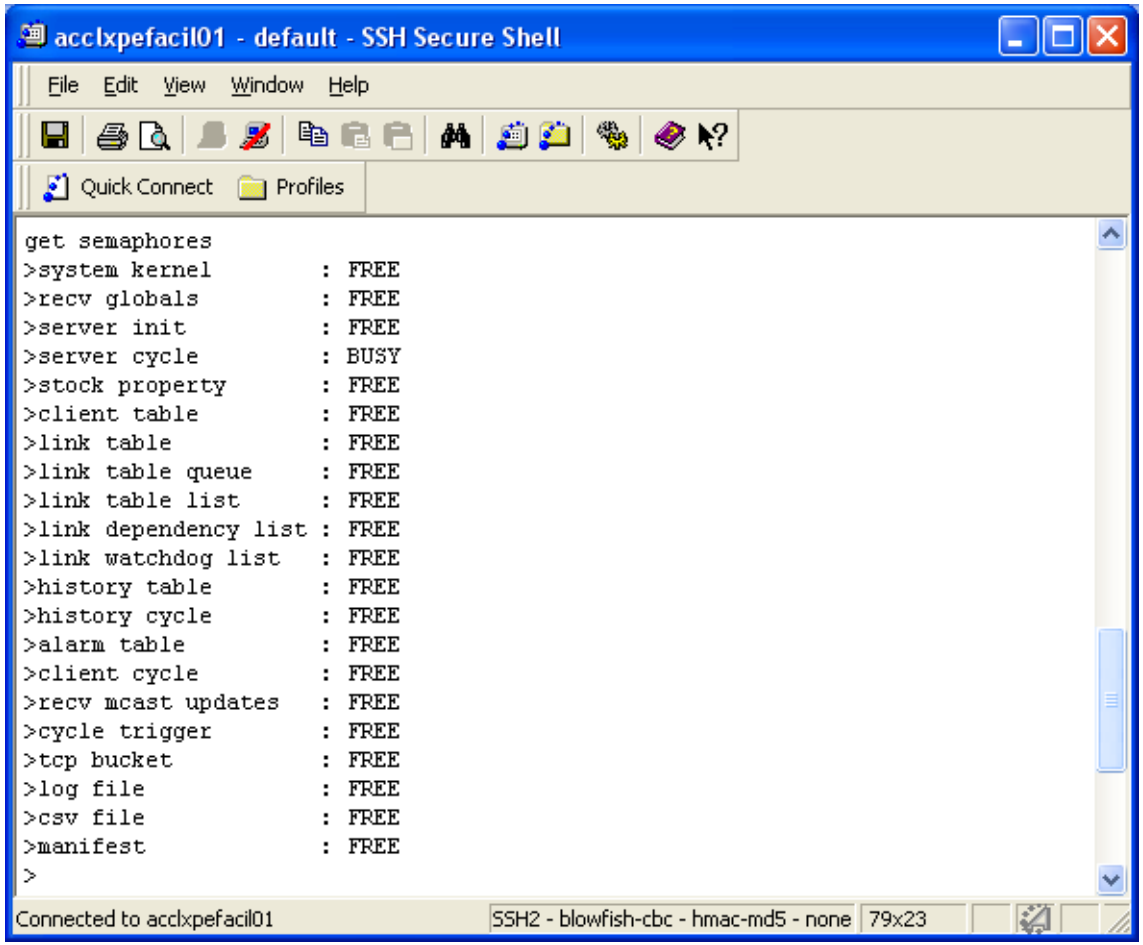


Debugging Utilities

- Server Problems
  - *In the unlikely event of a* Hang
    - check CPU load for the process
      - near 100% => infinite loop problem
      - near 0% => thread deadlock
    - use *attachfec* to check the activity
      - no response => debug thread is blocked too!
      - ‘get semaphores’ to check for deadlocks
      - ‘set debug=1’ to check for possible infinite loops, etc.

Debugging Utilities

- Normal state of semaphores and mutexes:



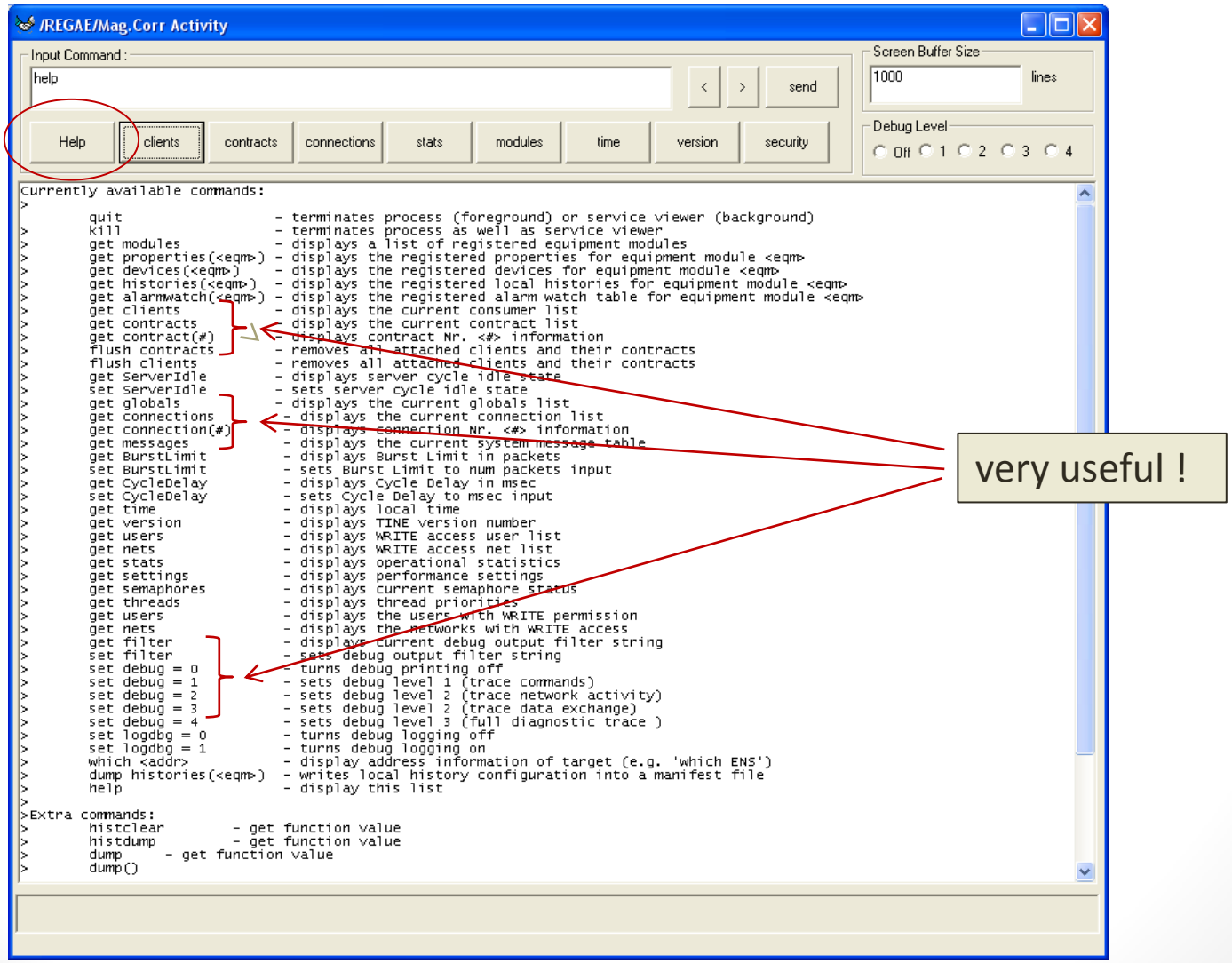
Debugging Utilities

- Server Problems
  - Make use of *attachfec* as a general rule.
    - *Windows*: GUI
    - *Unix*: command line tool

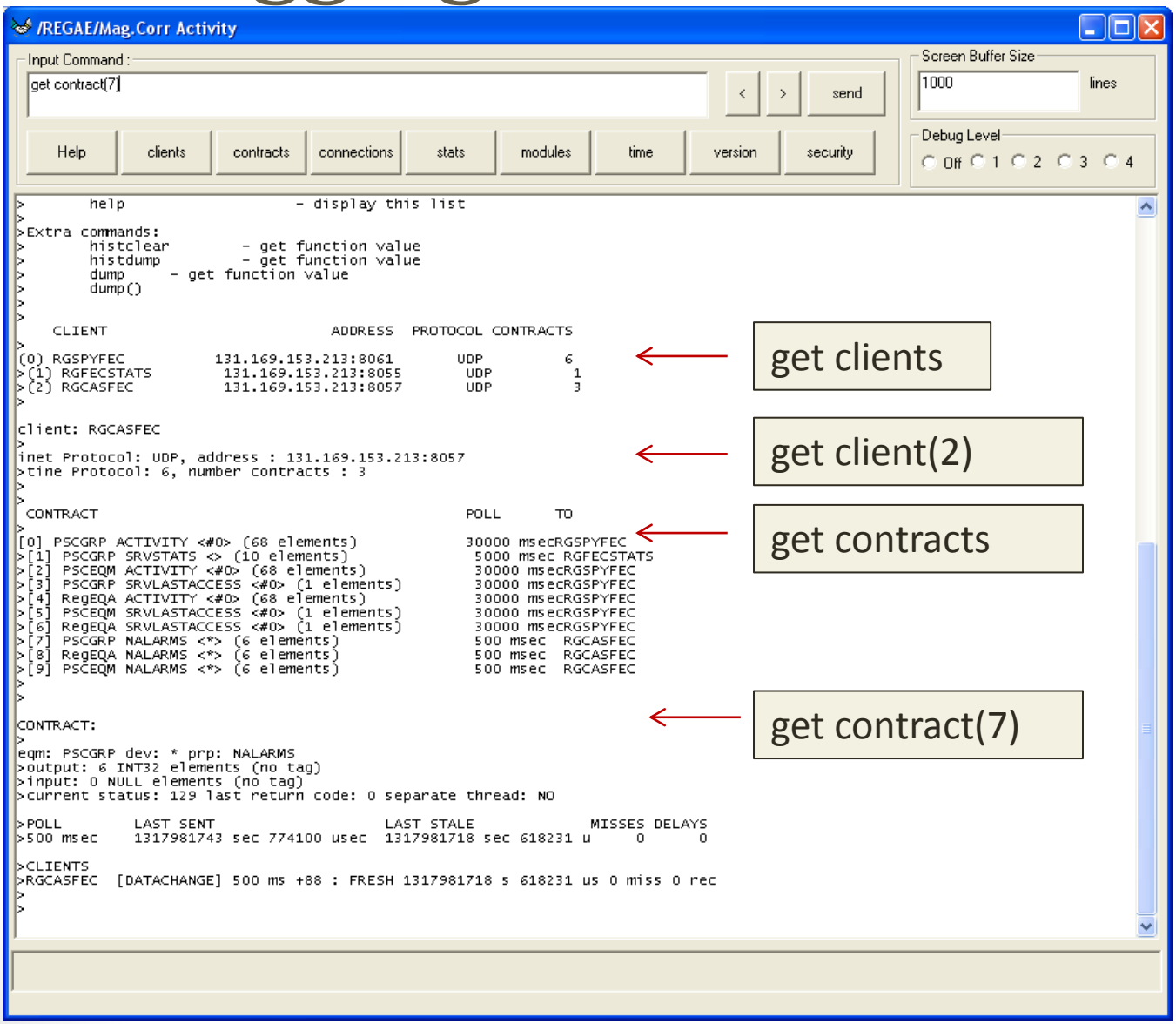
  - '*local pipe*'
    - does not involve the network
    - *attachfec* must run on same host as the FEC.
  - '*remote stream*' connects via a dedicated TCP socket.

Debugging Utilities

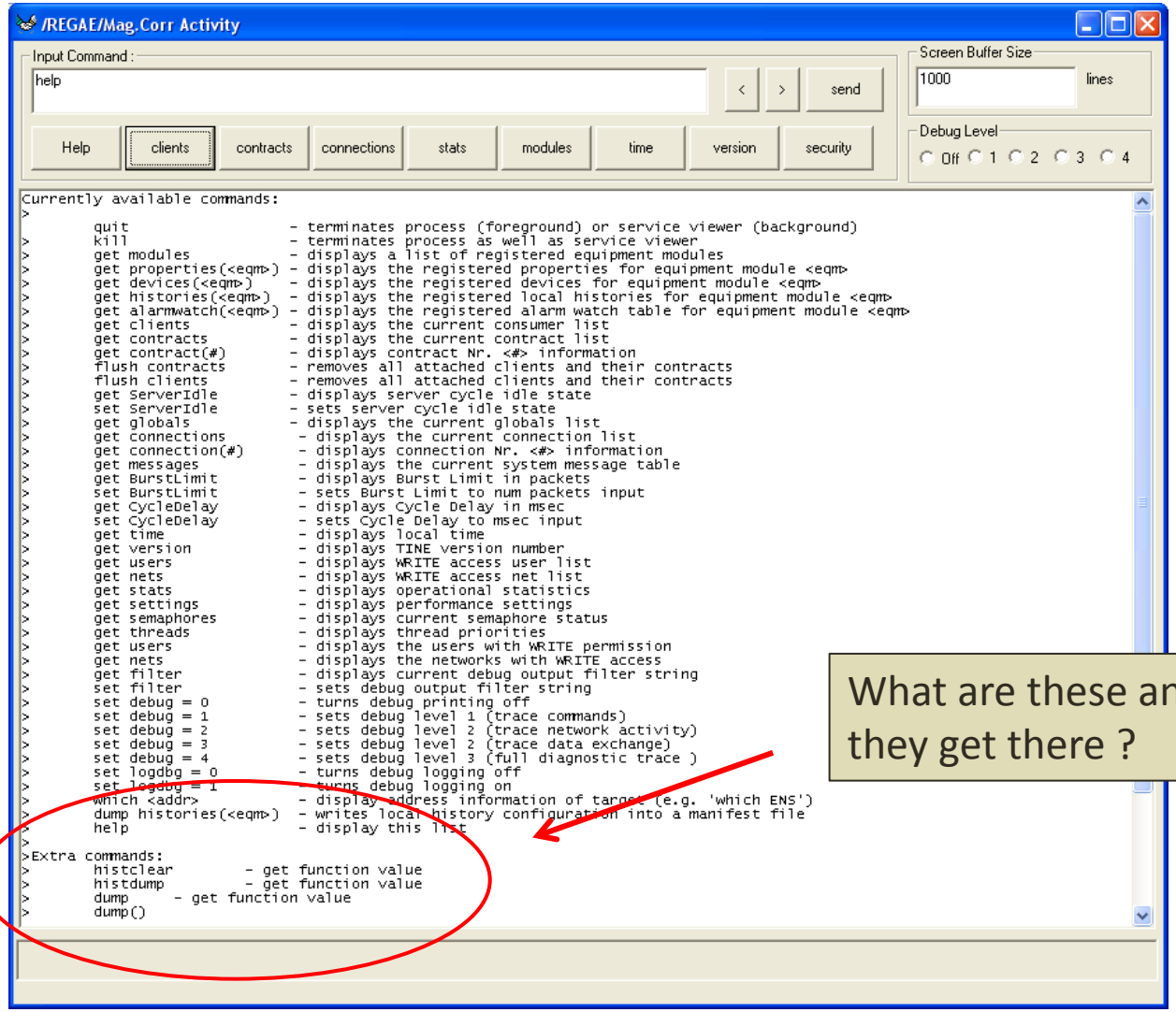
- e.g. attachfec /REGAE/Mag.Corr



Debugging Utilities



Debugging Utilities



What are these and how did they get there ?

Debugging Utilities

get/set style commands (legacy routine)

Function Documentation

```
int RegisterUserCommand ( char *                  cmd,
                          int(*)(int, int, int, int) fcn,
                          int *                   iparam,
                          float *                 fparam,
                          int                     access
                        )
```

Registers a user-defined command or variable which can be accessed via the TINE command line interpreter.

As a console application, a running tine server (or client) offers a variety of services at the command line (just type 'help'). For instance, you can turn debugging on or off, get the current server statistics, examine the connection tables, etc. This interface is avaible via the 'remote' accessor even if the server is running in the background. It it often frequently desireable to examine or alter server-specific variables or call a server specific routine. This can be partially achieved by making use of this registration function.

Parameters:

- **cmd** is the command string to be parsed by the command interpreter.
- **fcn** is an optional user defined function taking four integer arguments.
- **iparam** is a pointer to an optional global integer variable.
- **fparam** is a pointer to an optional global float variable.
- **access** is a TINE access code (either CA_READ or CA_READ|CA_WRITE). To allow 'set' commands at the command line, you should include the CA_WRITE flag.

Note:

Only one of fcn, iparam, or fparm, should be non-NULL when making this call. If the users types "get <cmd>" without arguments at the command line, then either iparam or fparam is displayed on the console, depending on which parameter was non-NULL at registration time. Likewise "set <cmd> = <val>" can set this variable to a value entered at the console. If a function is registered, then "get <cmd>(arg1,arg2,arg3,arg4)" will call the function registered with the arguments supplied. "set <cmd>(arg1,arg2,arg3,arg4) = <val>" will set all arguments equal to <val> and then call the function. If fewer arguments than 4 are supplied then the function will be called with '0' for the missing arguments.
A more reasonable command line function registration call is underway. However the routine so presented is generally sufficient for checking hardware readouts etc. and checking and setting the contents of global variables.

Returns:

0 if successful, otherwise a TINE completion code

Debugging Utilities

'Call' style (modern)

int RegisterUserFunction (char * name,
int(*)(int argc, char *argv[]) fcn
)

Registers a user-defined function which can be accessed via the TINE command line interpreter.

As a console application, a running tine server (or client) offers a variety of services at the command line (just type 'help'). For instance, you can turn debugging on or off, get the current server statistics, examine the connection tables, etc. This interface is avaible via the 'remote' accessor even if the server is running in the background. It it often frequently desireable to examine or alter server-specific variables or call a server specific routine. This can be partially achieved by making use of this registration function.

Parameters:

name is a string identifying the function to be parsed by the command interpreter.

fcn is the user defined function taking the argc and argv.

Returns:

0 if successful, otherwise a TINE completion code

e.g.

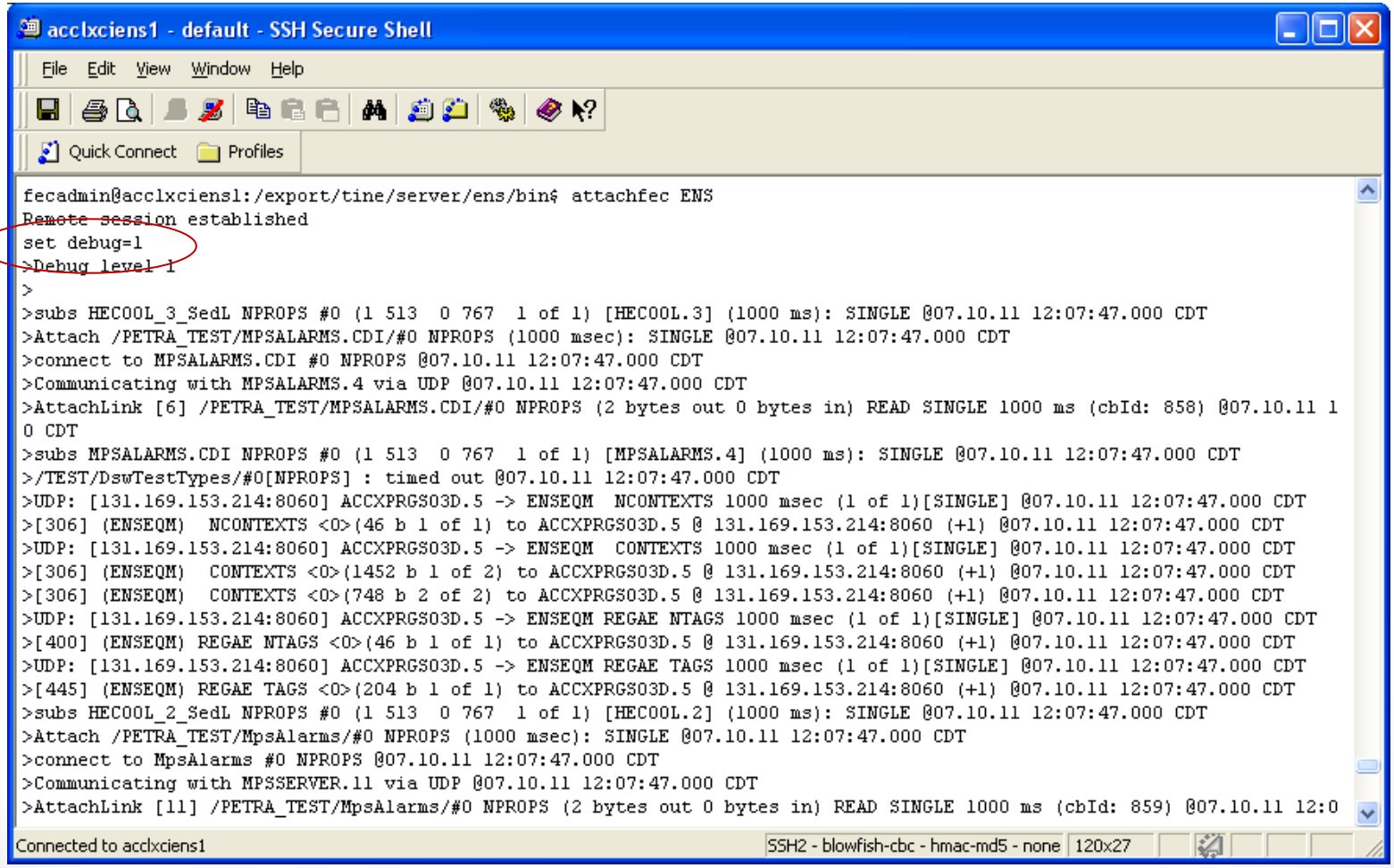
```
// other initialization omited ...

RegisterUserFunction("reload",daqReload);

// ...
```

```
int daqReload(int argc,char *argv[])
{
  argc = argc; argv = argv;
  feclog("reload DAQ database");
  /* flushStructCache(); */
  freeDaqDb();
  /* we should be back to square one now */
  daqInit();
  return 0;
}
```

Then: type 'reload' at the command prompt to call the designated function.

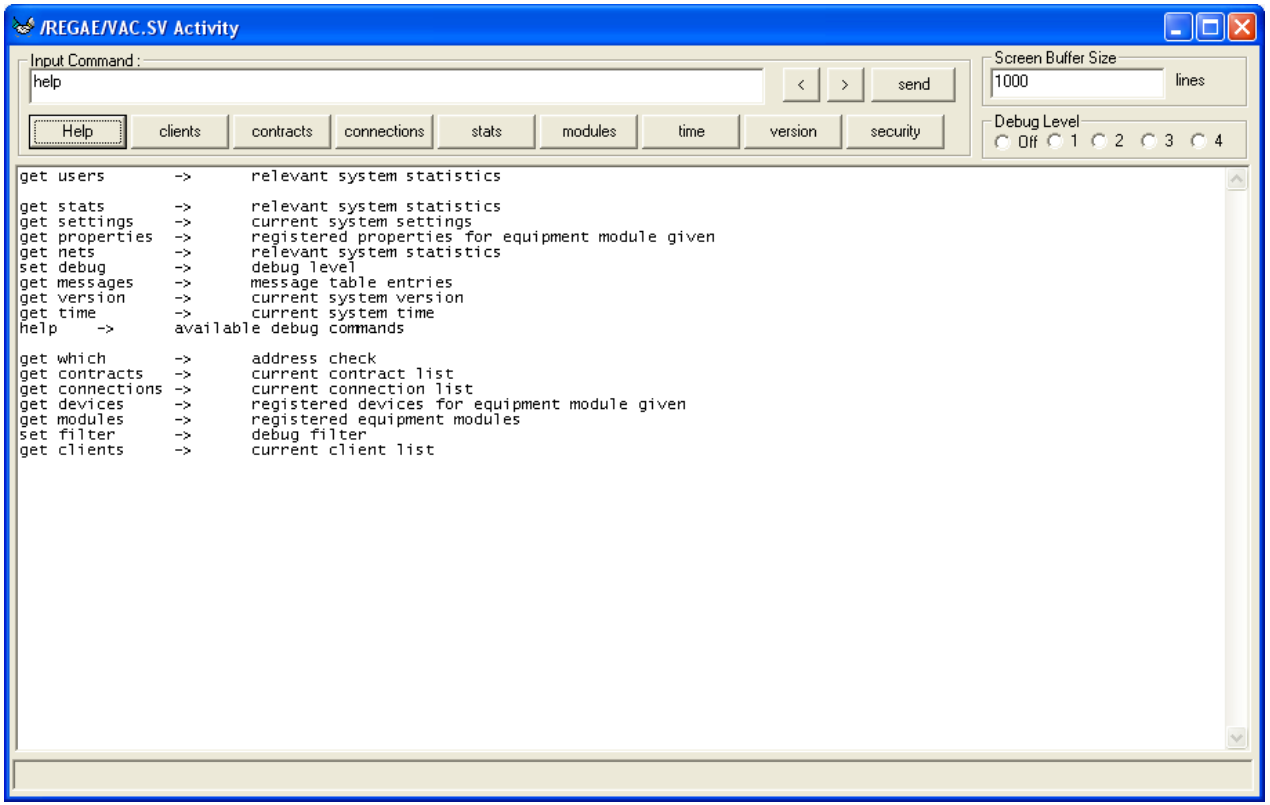
Debugging Utilities

- attachfec ENS (someone who is usually busy!)



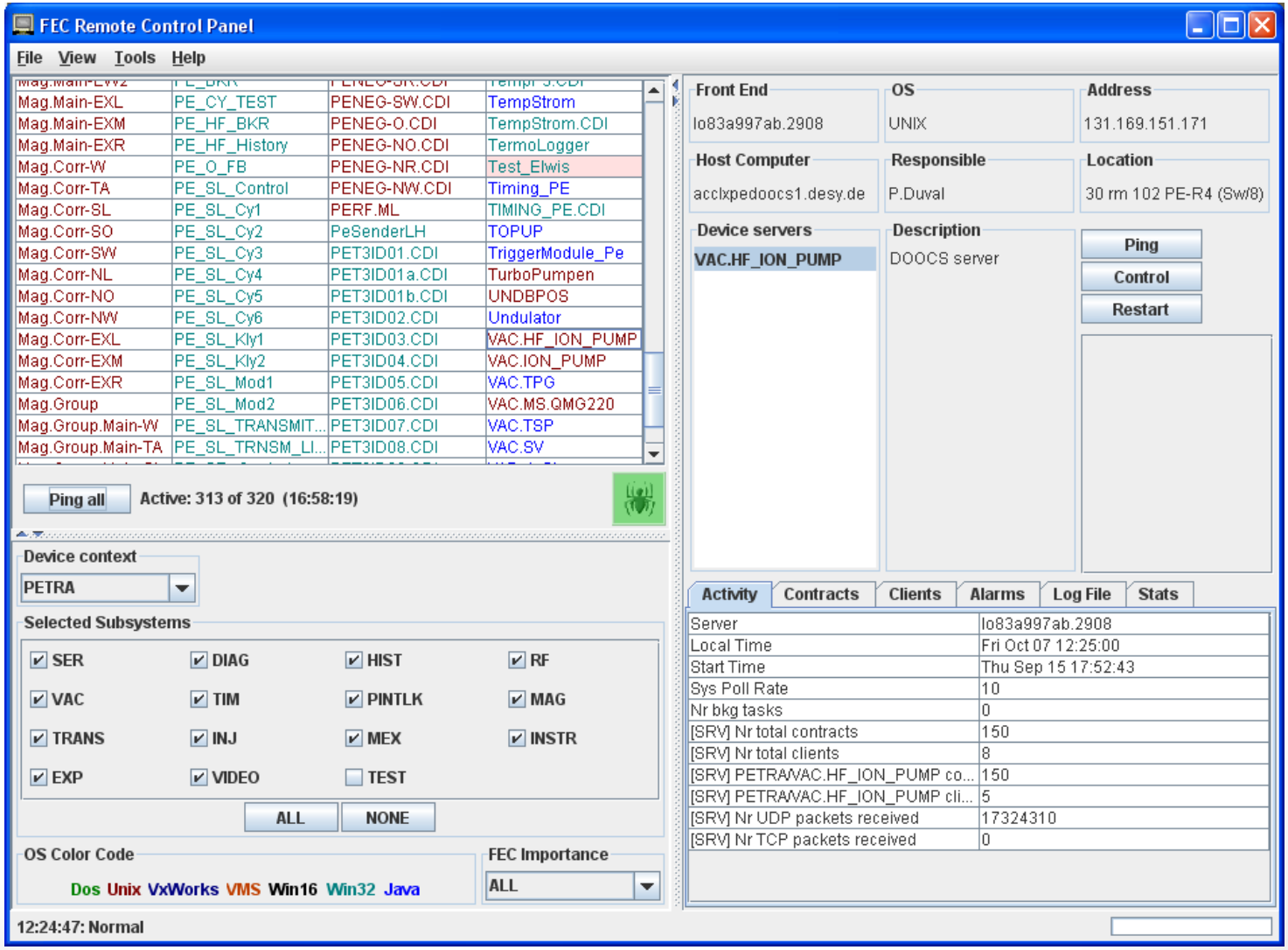
Debugging Utilities

- NOTE: java servers
  - have a reduced set of debug commands
  - have slightly different output with debug > 0



Debugging Utilities

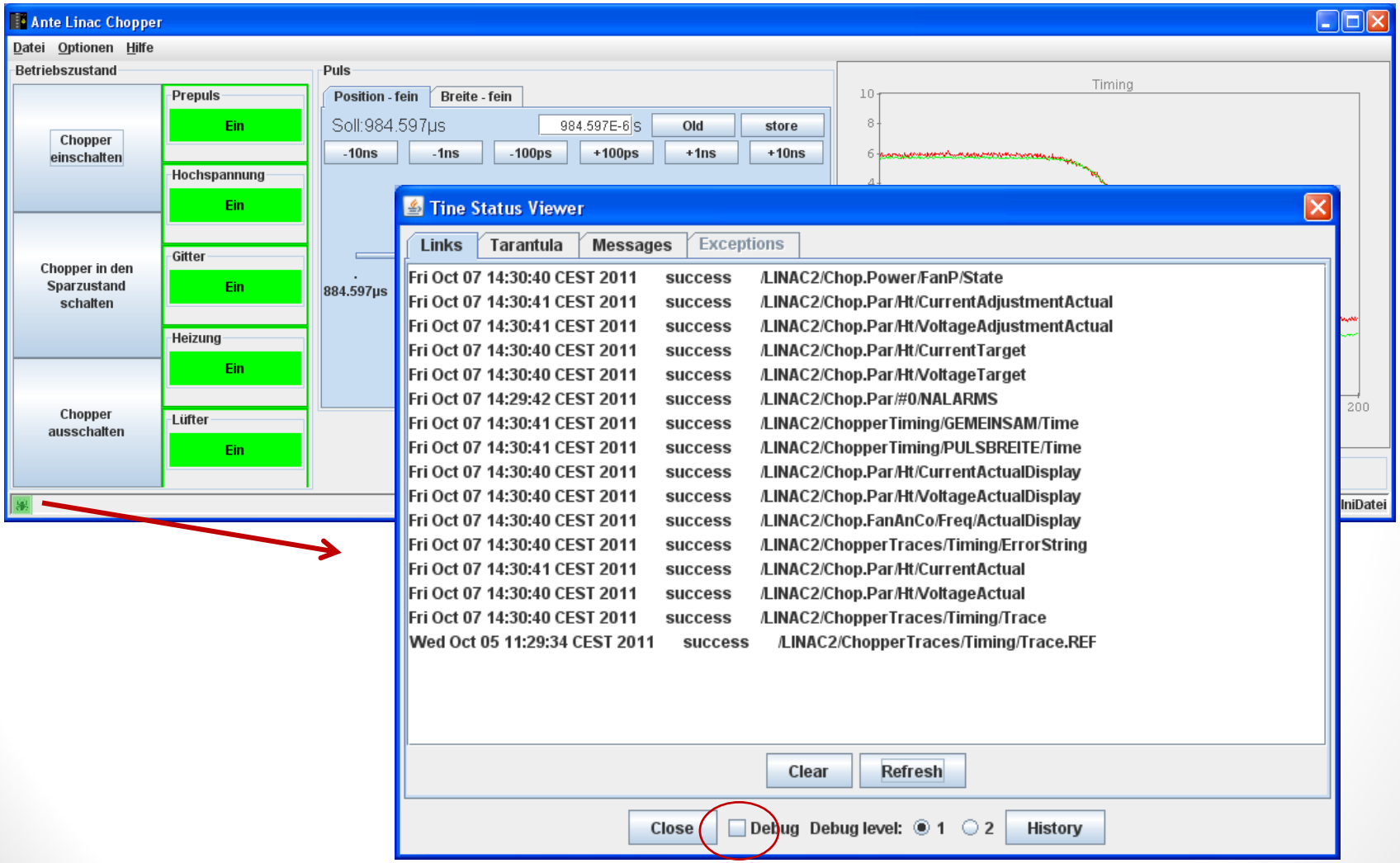
- General overview of all servers :



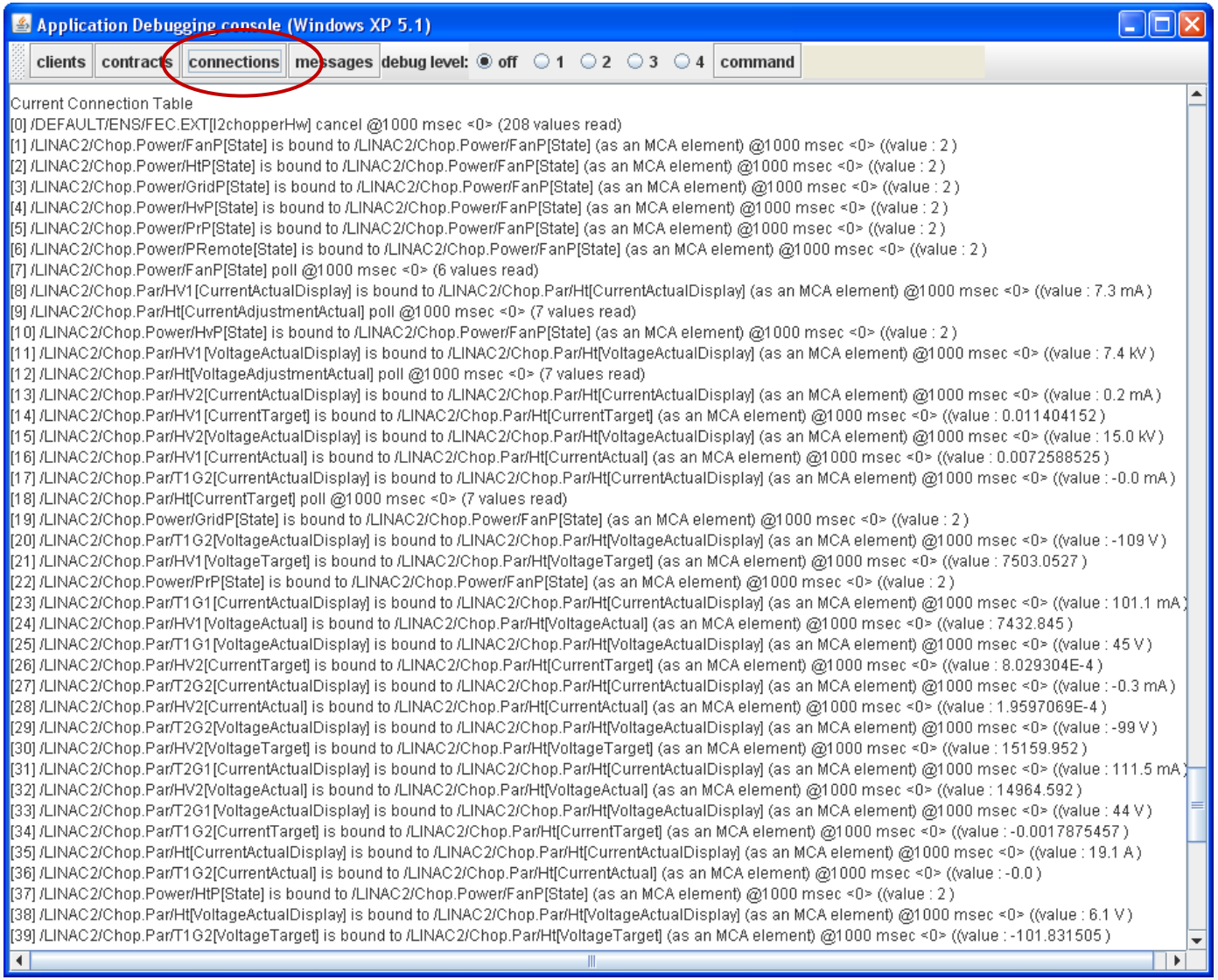
Debugging Utilities

- Client problems (where the client is not a FEC)
  - just some process (not bound to a host, port, pid, …)
    - no de-facto log file.
  - GUI applications with the ACOP spider can launch a console-like debugging session. (java and activeX)
    - ACOP Tarantula can provide a complete hierarchical link status tree (java only)
  - Can also use ‘attachfec’ locally (even though it isn’t a FEC) if the partners can agree on a pipe name!
    - e.g. the pid

Debugging Utilities

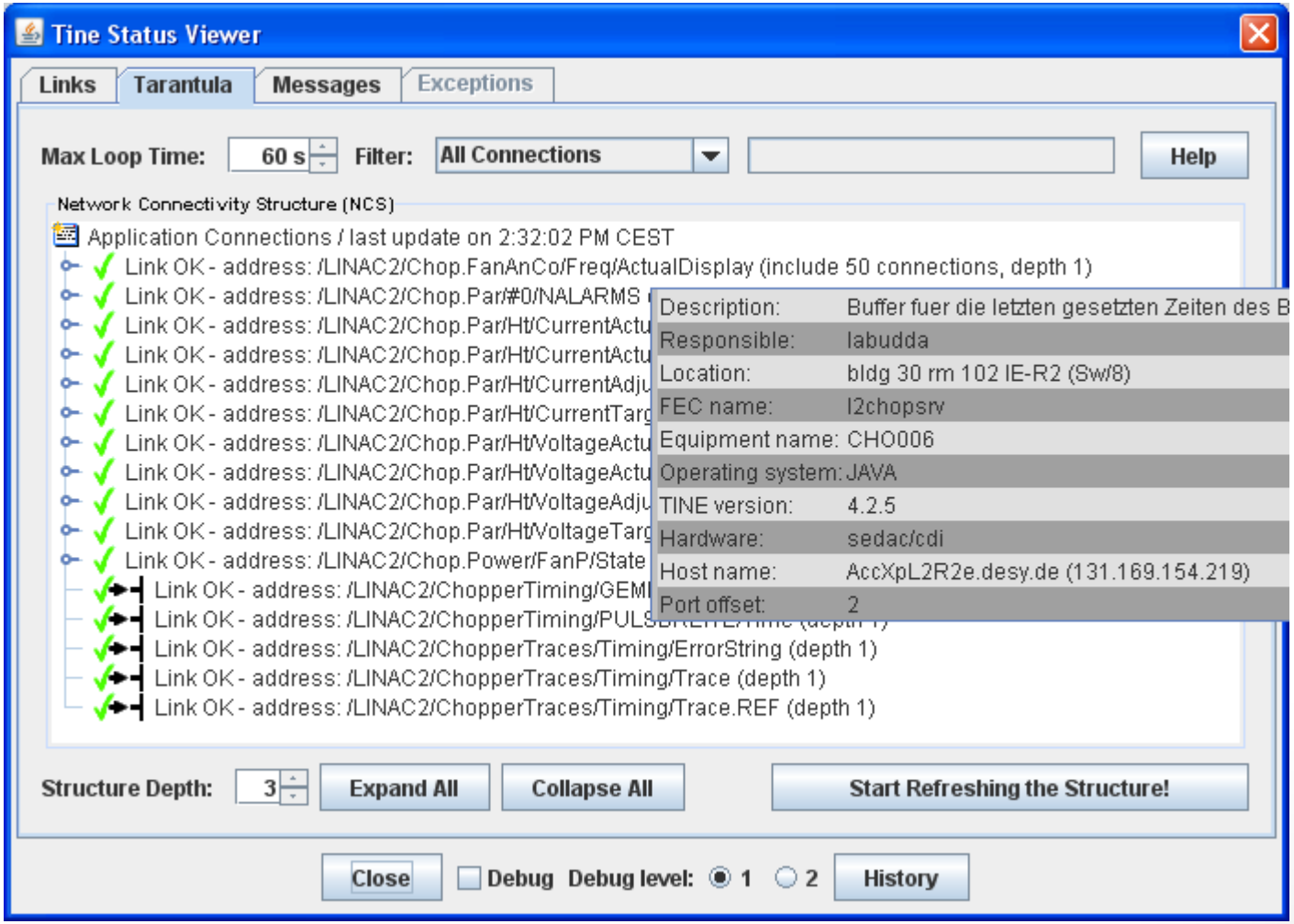


Debugging Utilities



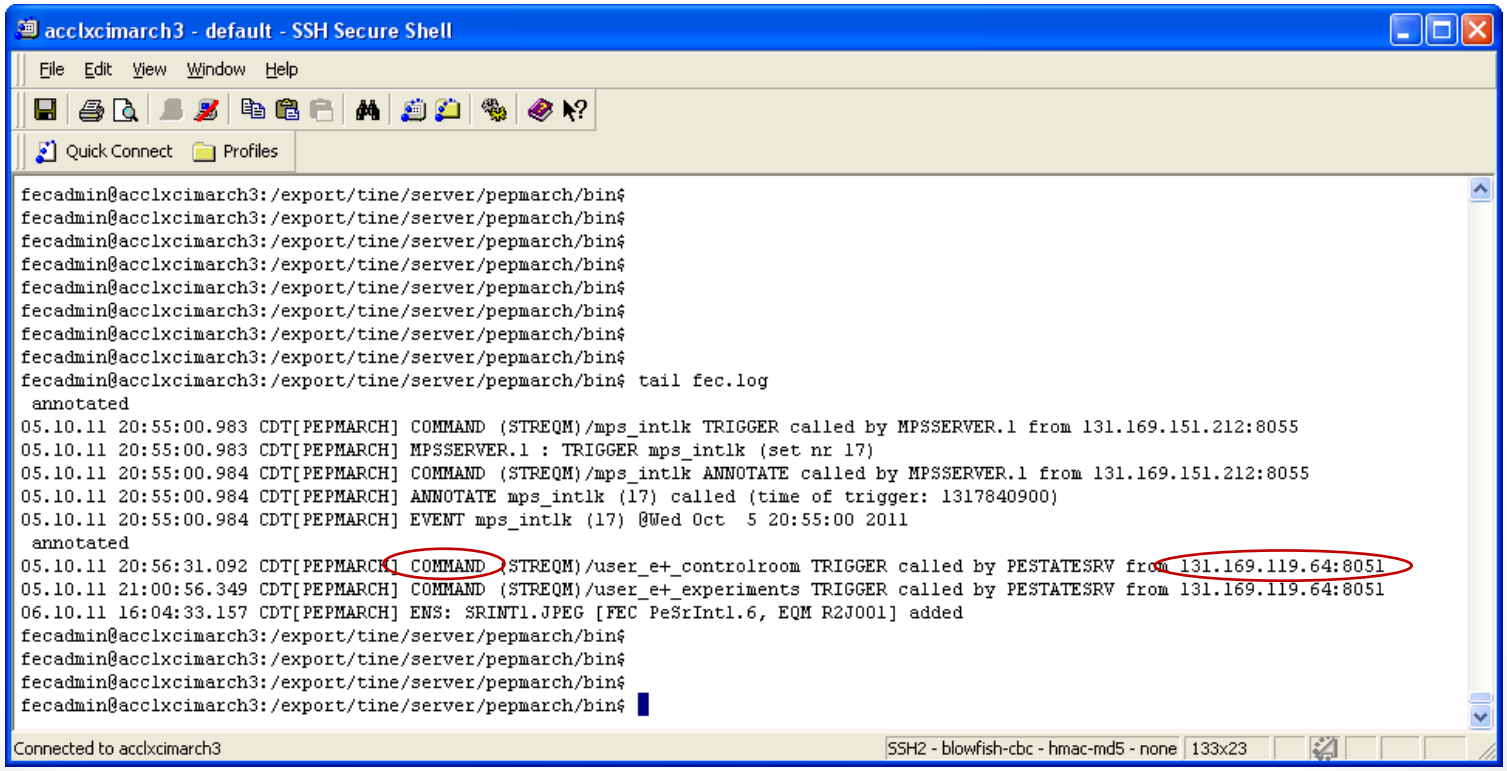
Debugging Utilities

Debugging Utilities



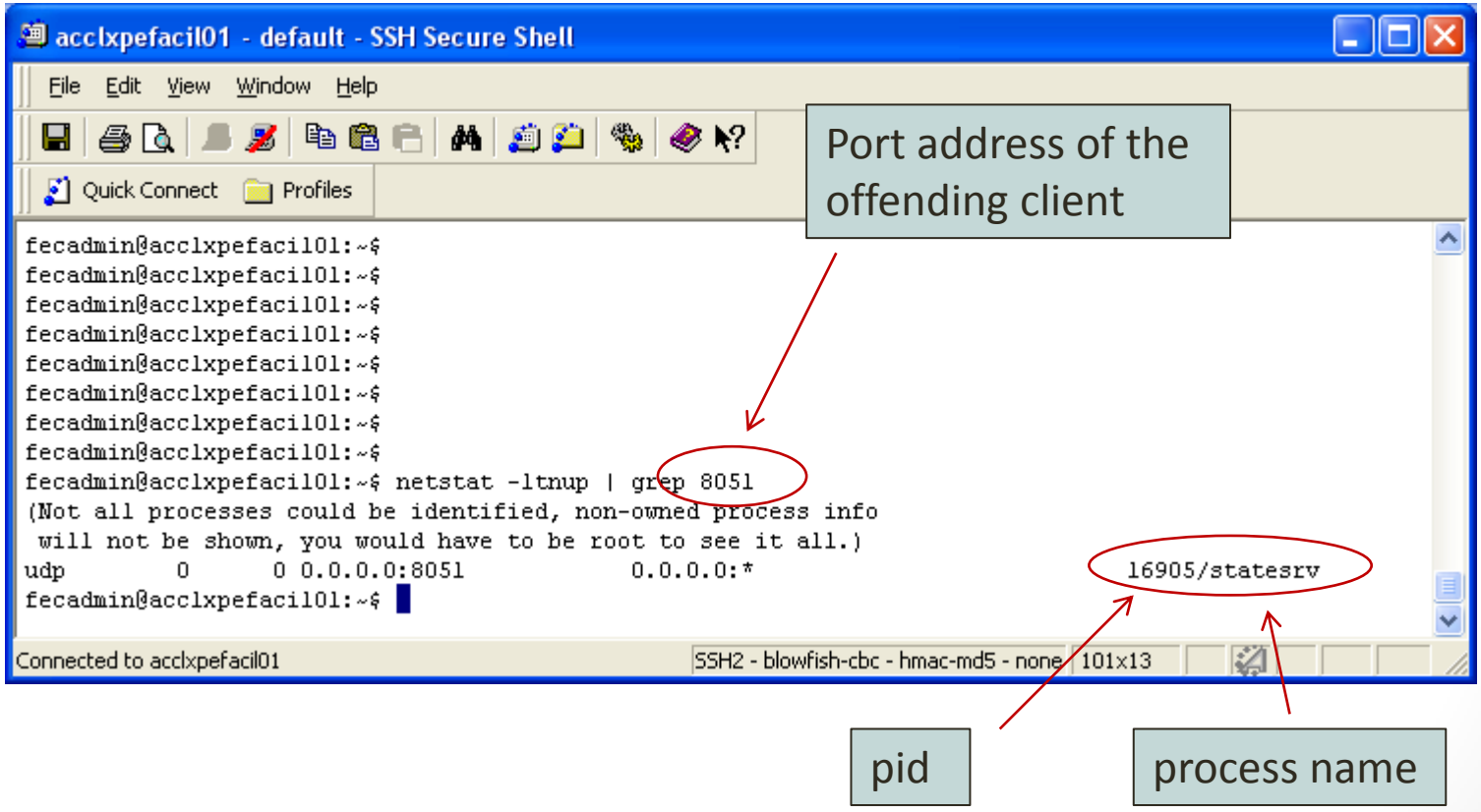
Debugging Utilities

- Finding the '*bad*' client …
  - A server is '*under attack*' from some client
  - The clients network address (including port) is displayed in the server's fec.log as well as on the debug console.



Debugging Utilities

- login to 131.169.119.64 and use e.g. ‘netstat’ (for linux)



Debugging

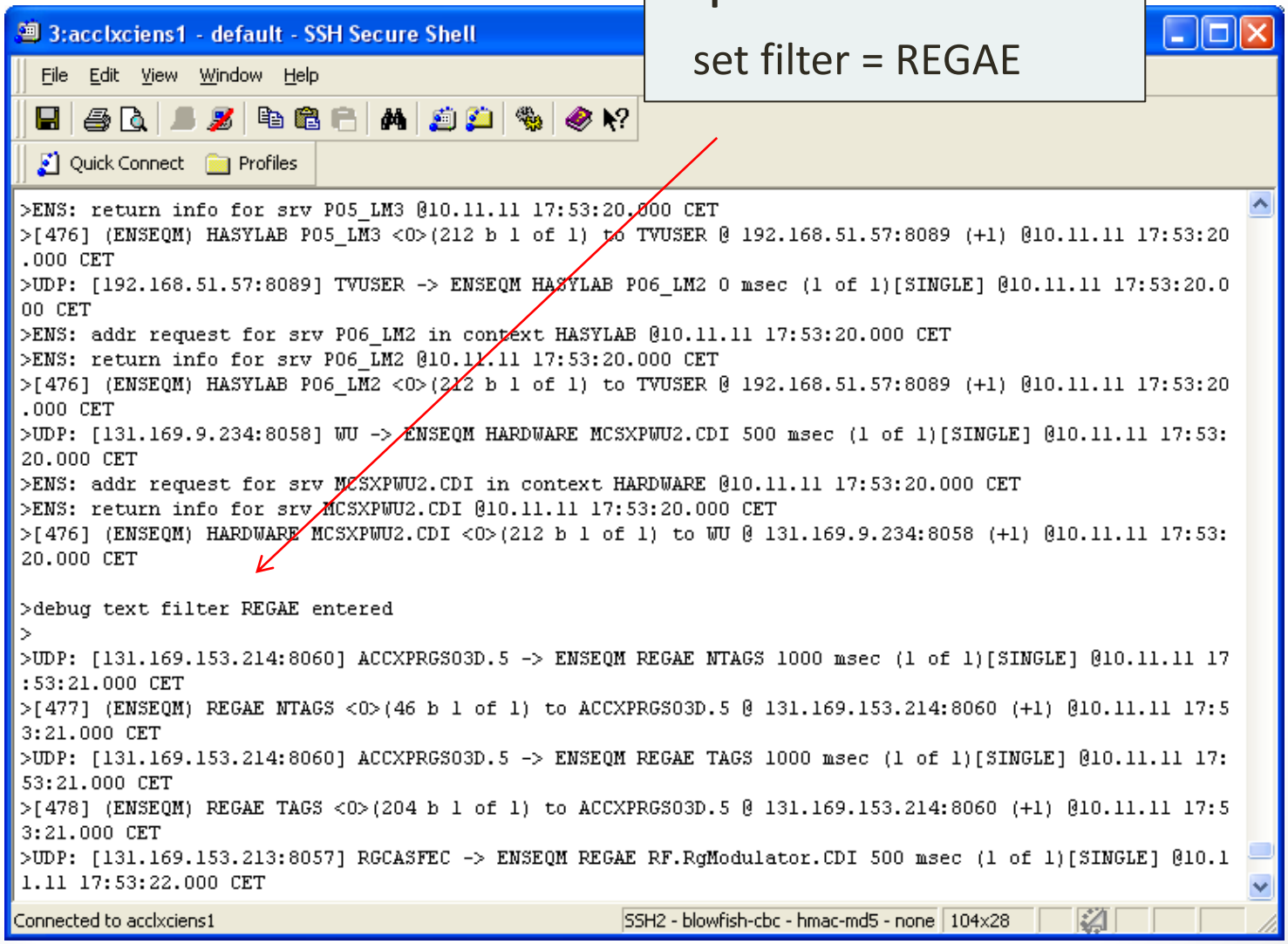
- Things that sometimes happen ….
  - *Rapid polling from a client*
    - e.g. a script running through all 1000 devices one-by-one at 1 Hz.
    - will make the server (and probably the network) very busy
    - a script doing this will also make the ENS busy !

- Use '*filters*' to help find needles in haystacks …
  - positive filter
  - negative filter

Debugging

a positive filter :

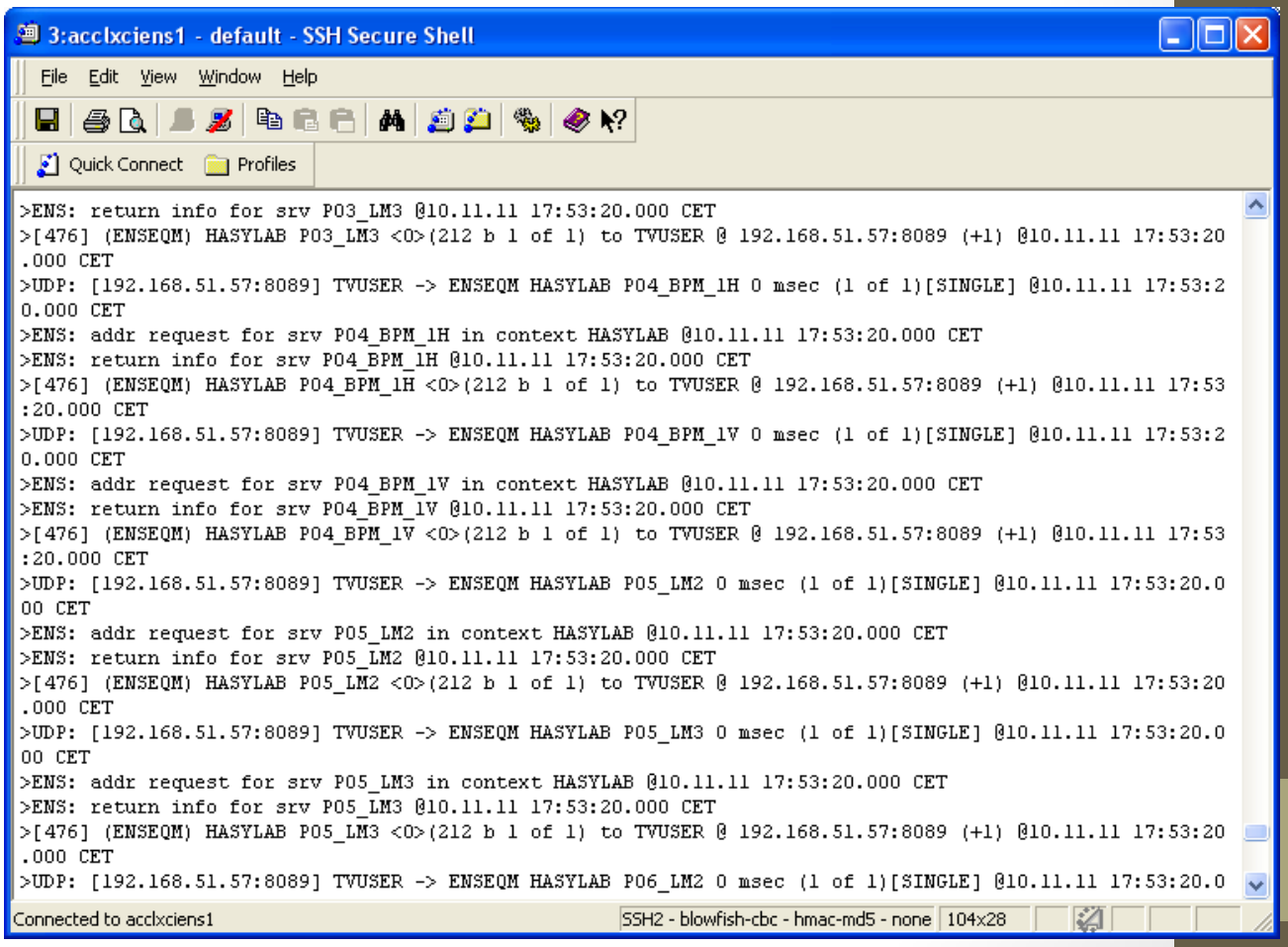
set filter = REGAE



Debugging

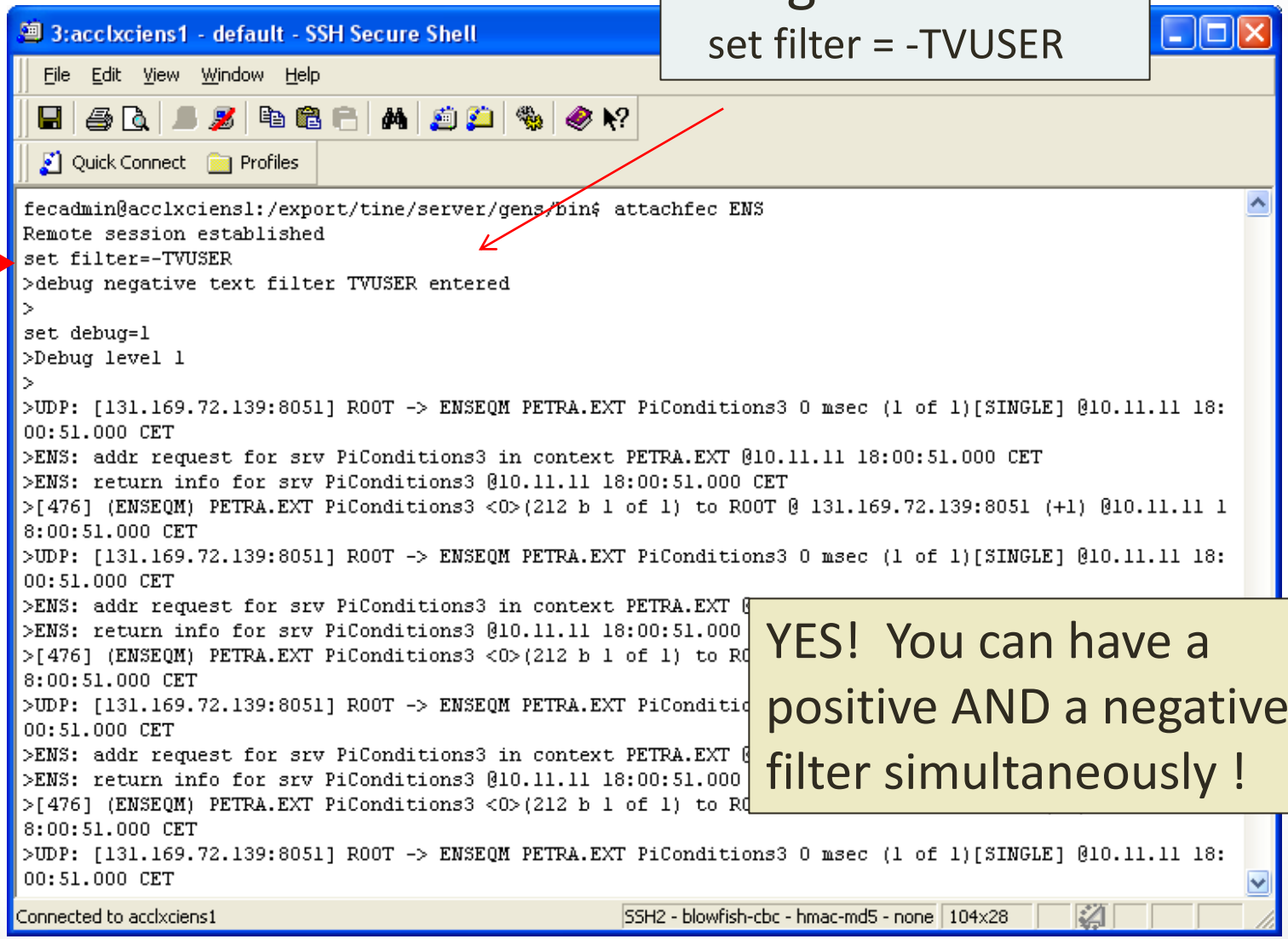
But sometimes you want to filter ‘out’ and not filter ‘in’ :

e.g. get rid of “TVUSER” from the output !



Debugging

a negative filter :
set filter = -TVUSER



YES! You can have a positive AND a negative filter simultaneously !

Part III: The Instant Client

It doesn't do everything, but it tries …

The Instant Client

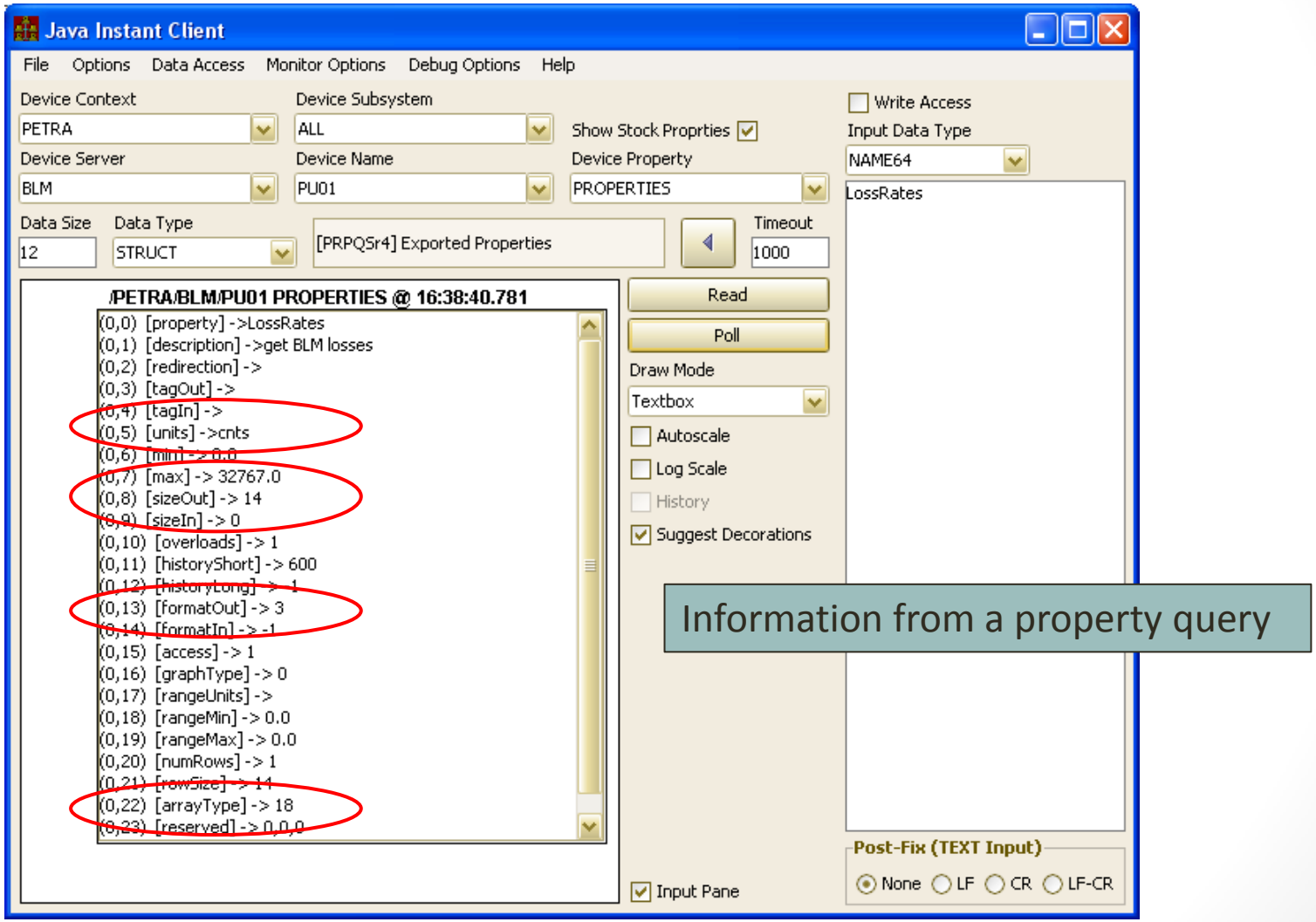
- a control system *browser*
- queries ENS for *contexts*, *servers*, and *subsystems*
  - these entries are fixed in the *java* combos!
  - and should be fixed in the *windows* version as well!
- queries selected server for *devices* and *properties*
  - the populated combo boxes here are NOT fixed
    - BUT any random, entered property and device names are unlikely to succeed!
  - determines one of :
    - *flat hierarchy*
    - *device-query precedence* (device server model)
    - *property-query precedence* (property server model)

The Instant Client

- queries selected property for all relevant information
  - data size and format
  - data array type
  - units and settings
  - property description
  - number of overloads (usually = 1)
  - history depths
- information used to fill in default selections
  - e.g. Draw Mode = “poly line” when array type = TRACE
  - Draw Mode = “histogram” when array type = CHANNEL
  - Draw Mode = “text” when array type = SCALAR or UNKNOWN

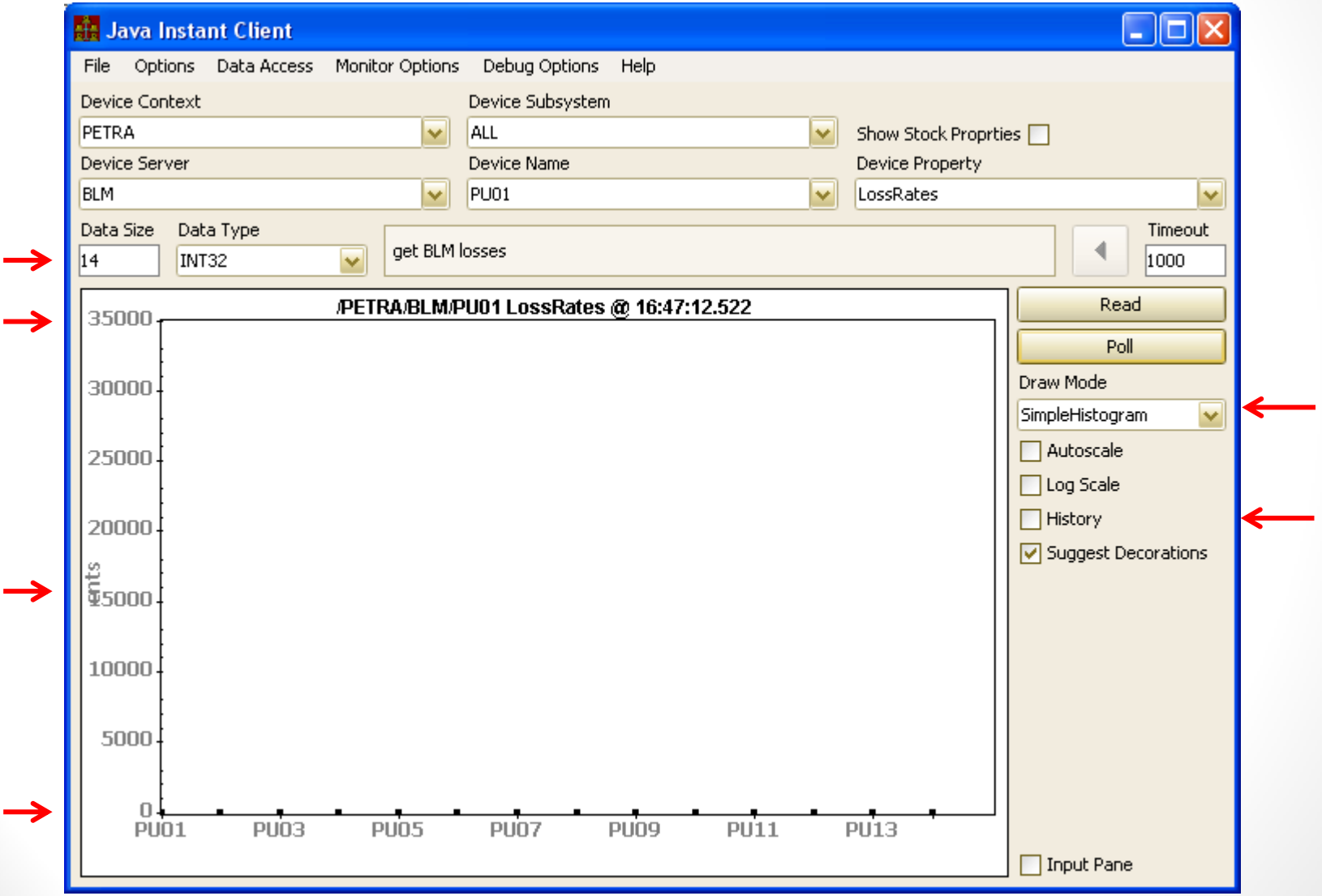
Note: this particular adjustment is sometime annoying
=> need a checkbox to turn this off (coming soon!)

The Instant Client

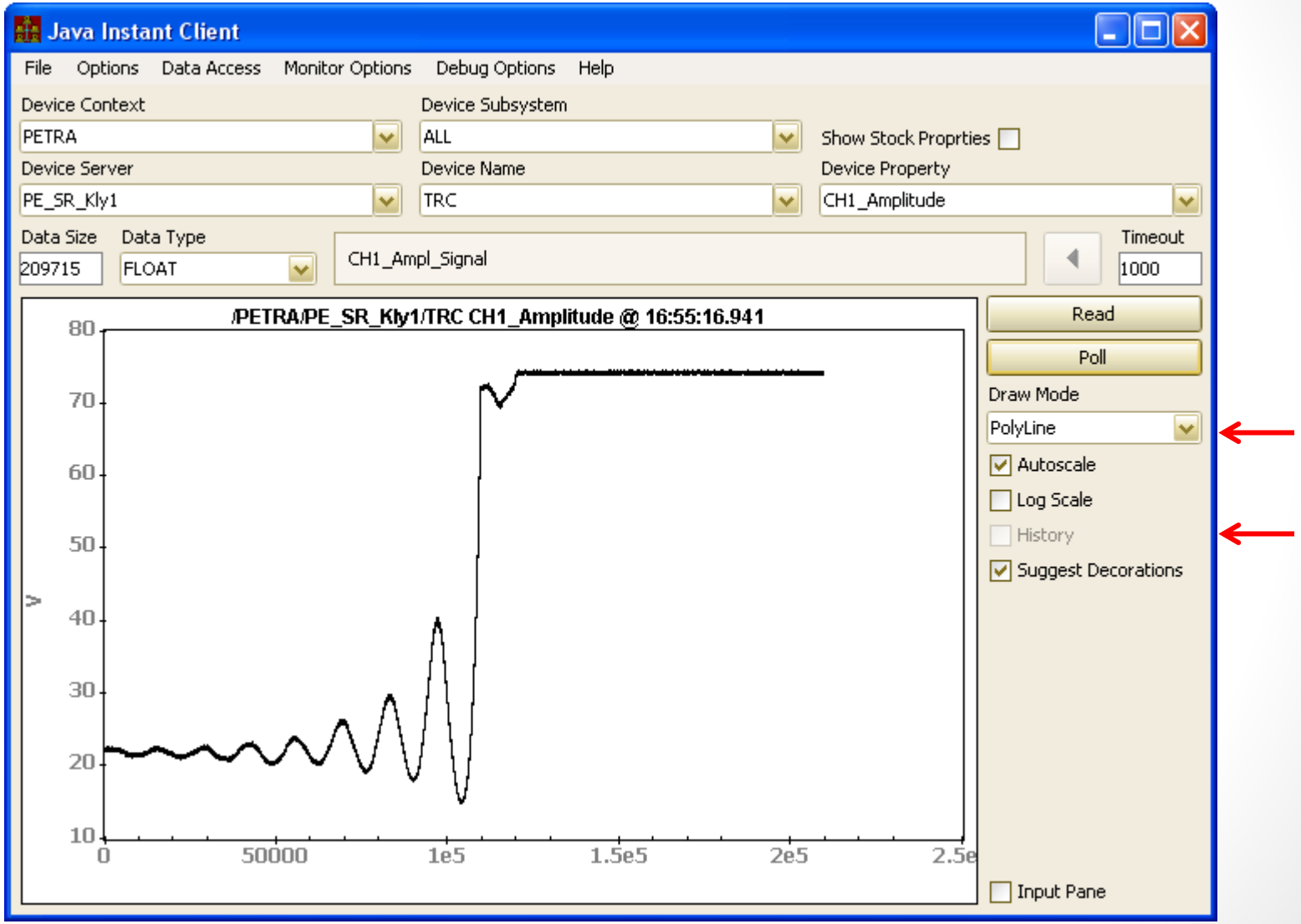


Information from a property query

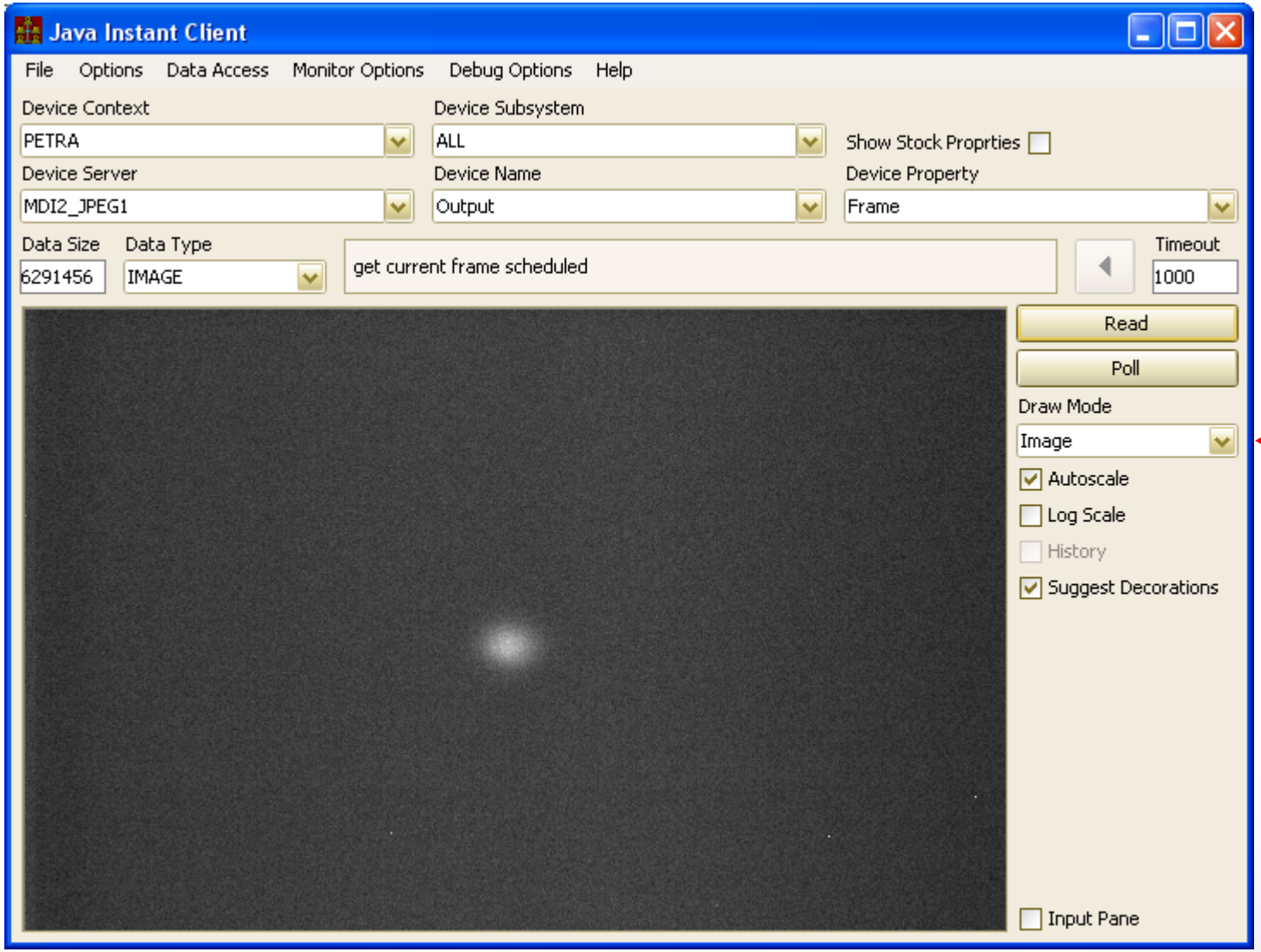
The Instant Client



The Instant Client

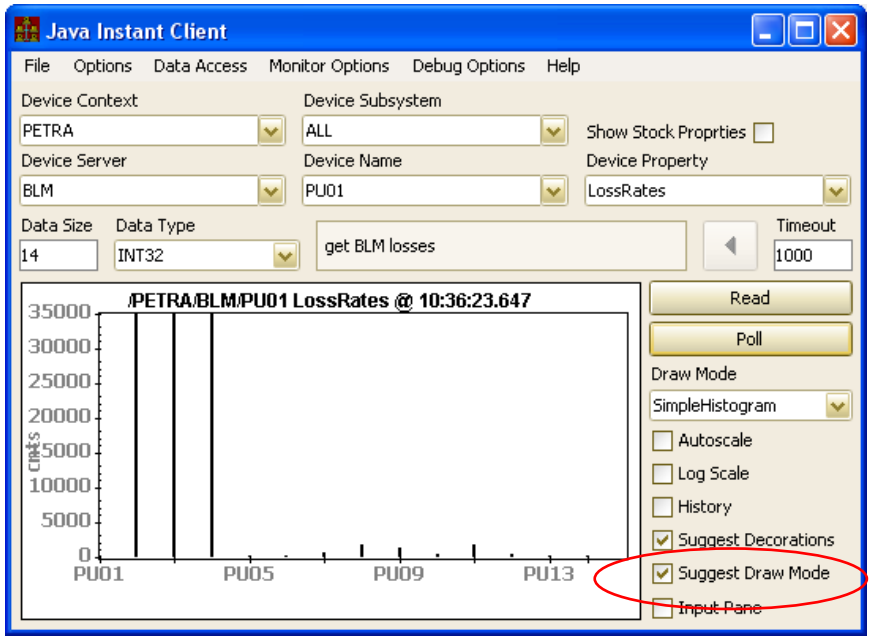


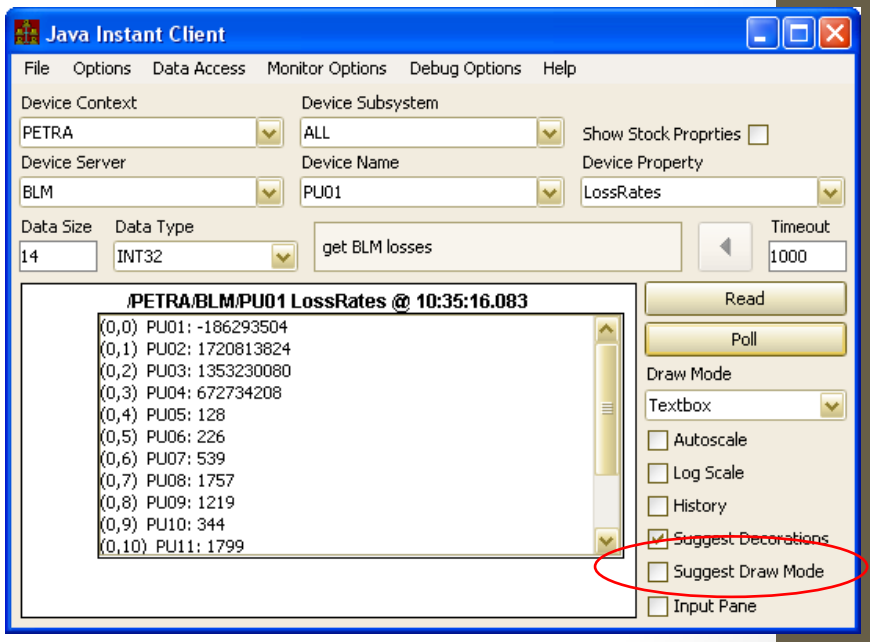
The Instant Client



The Instant Client

- Hot off the presses:



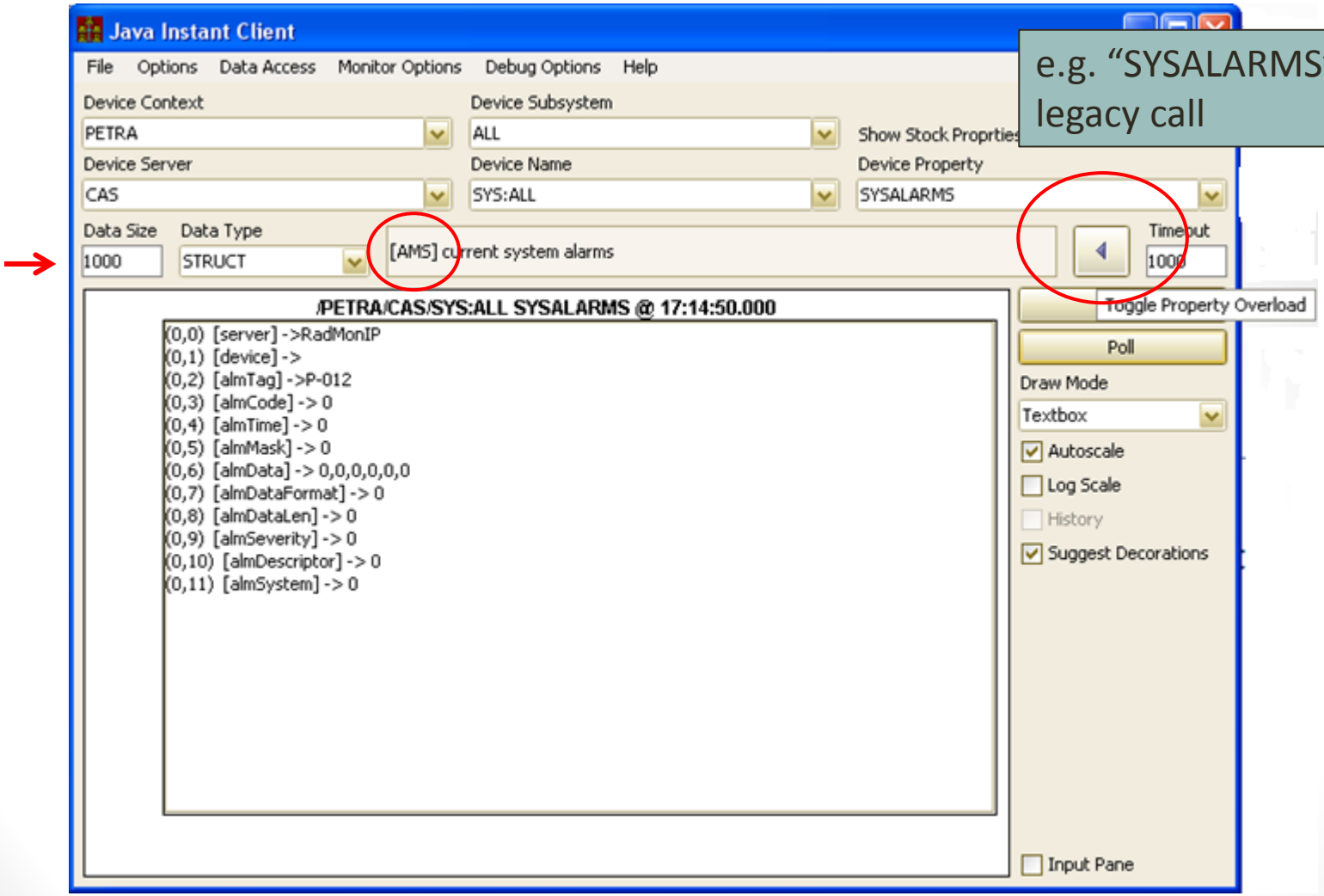


The Instant Client

- *overloaded* properties
  - a server can register a property *more than once* (with different data input/output criteria) !
    - e.g. output format = INT32 -> raw hardware readback;

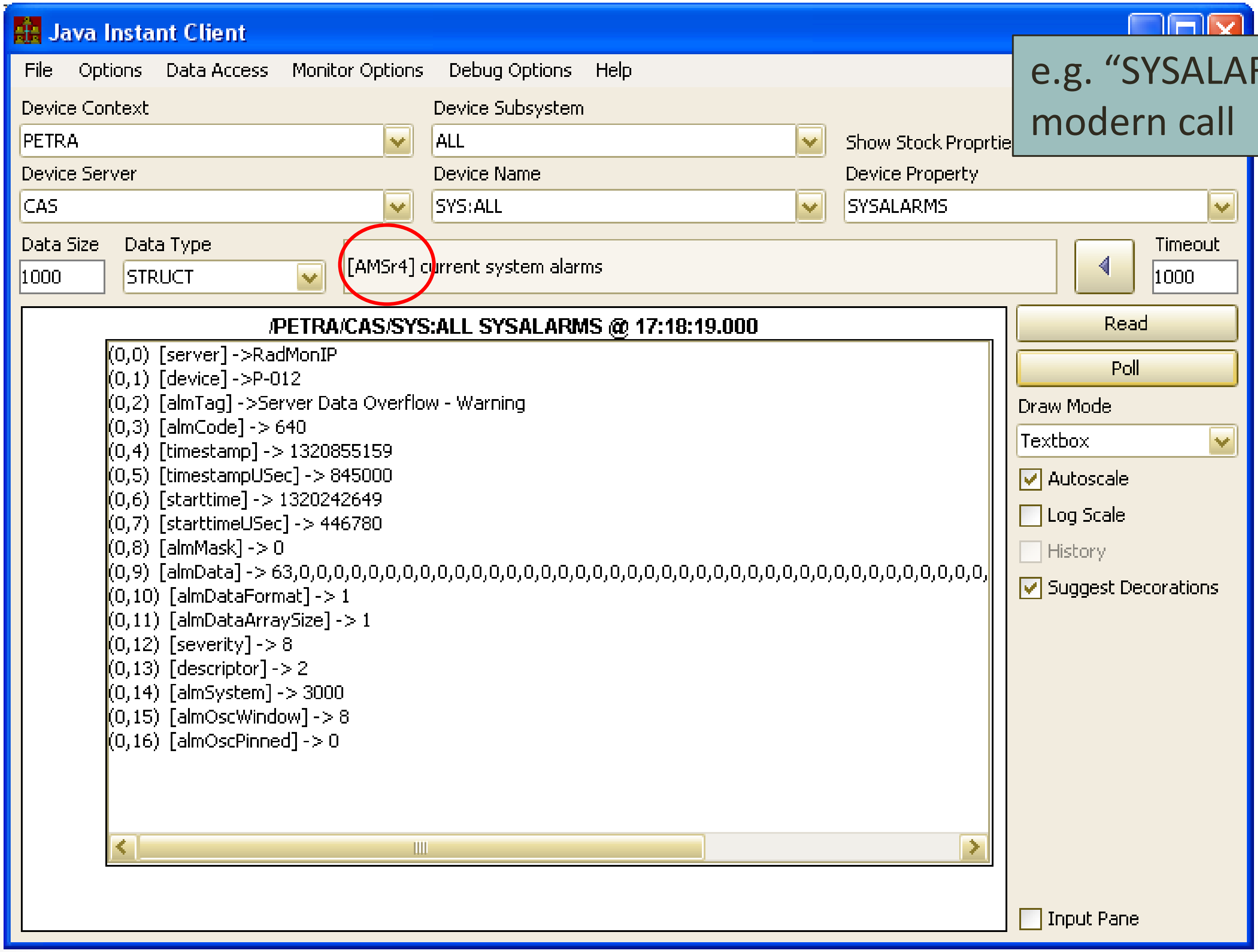
      output format = FLOAT -> calibrated readback
    - e.g. structured format supporting ‘legacy’ as well as ‘modern’ structures.

The Instant Client



e.g. “SYSALARMS”: legacy call

The Instant Client

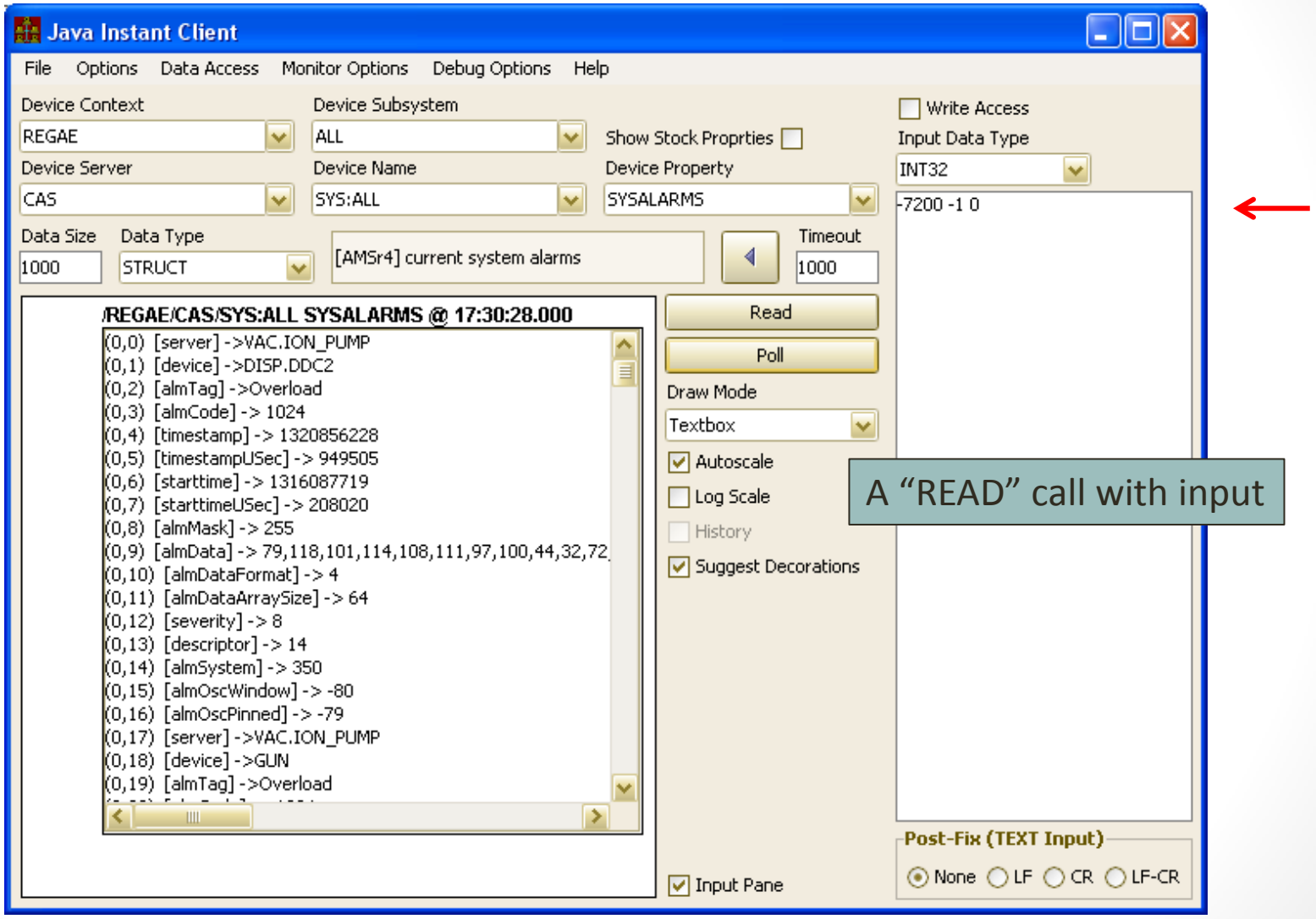


e.g. “SYSALARMS”: modern call

The Instant Client

- *Input panel*
  - accepts all input data types (*except* STRUCT)
    - (parsing structure input might be a good cosylab assignment)
  - array input:
    - delimiter = *comma* or *white space*
    - strings with *blanks* enclose in “”
  - *special case* : type TEXT
    - Can add a CR, LF, or CR-LF as a post-fix
      - e.g. RS232 input requires a termination character
  - *important Reminder*:
    - Data Input does **NOT** imply WRITE ACCESS !

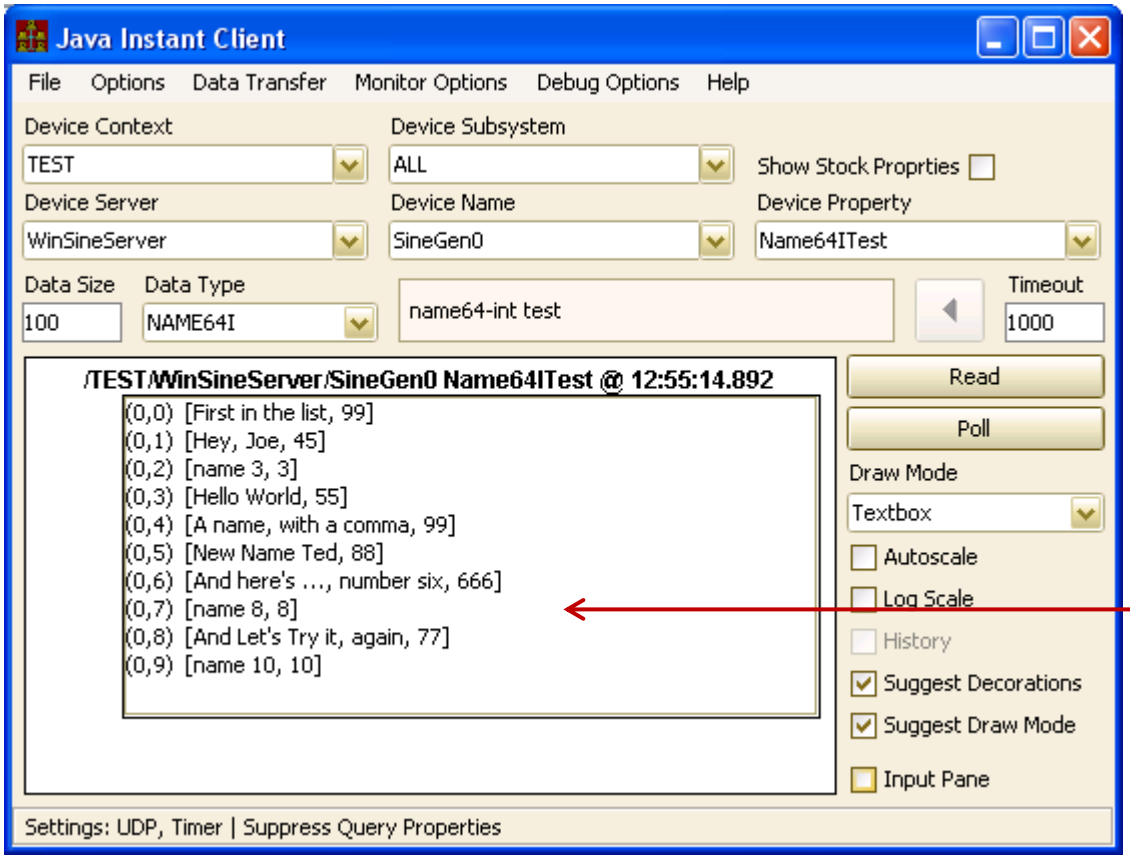
The Instant Client



A “READ” call with input

The Instant Client

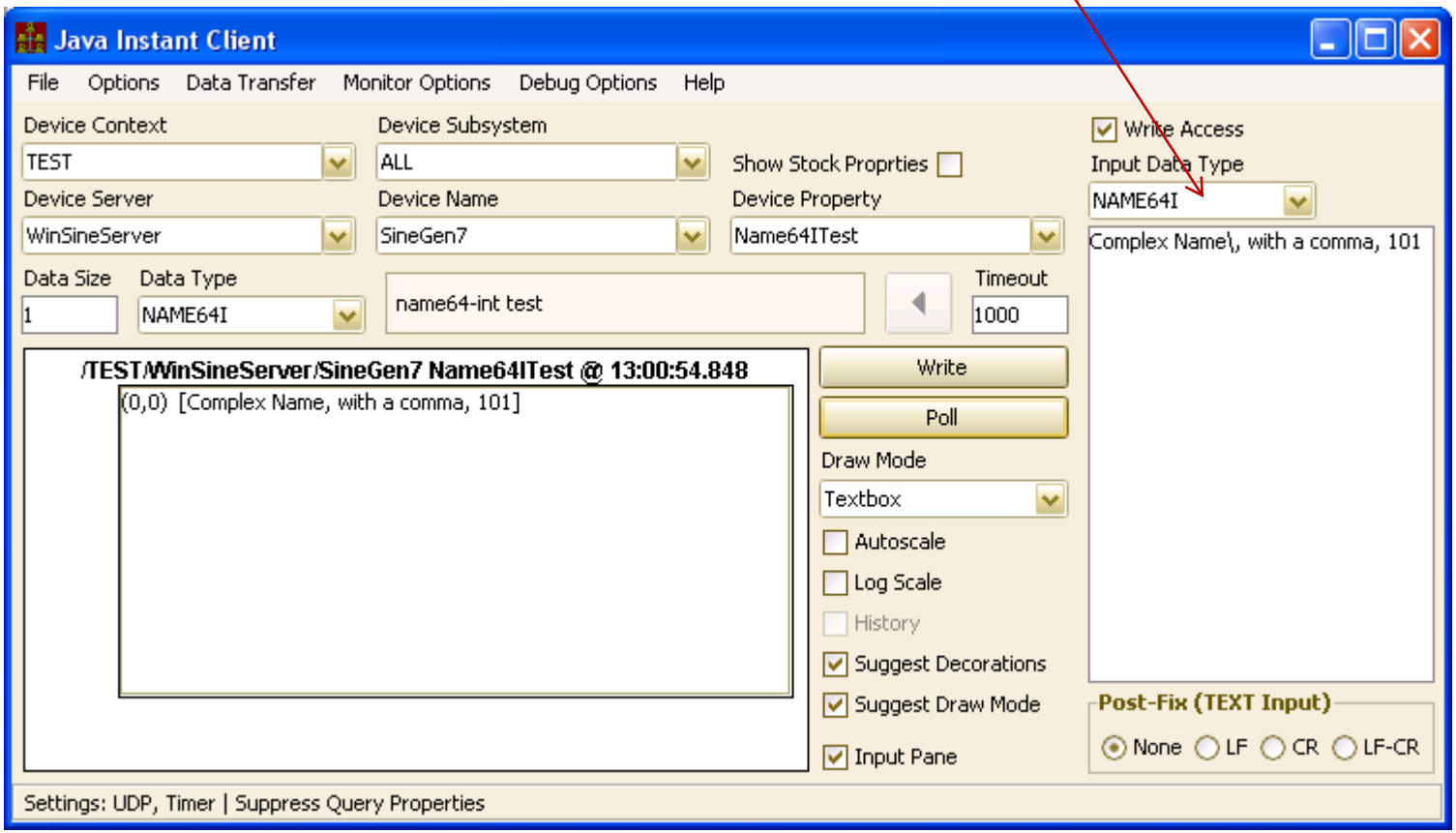
Complex data types: A parsing nightmare …



Let's change this !

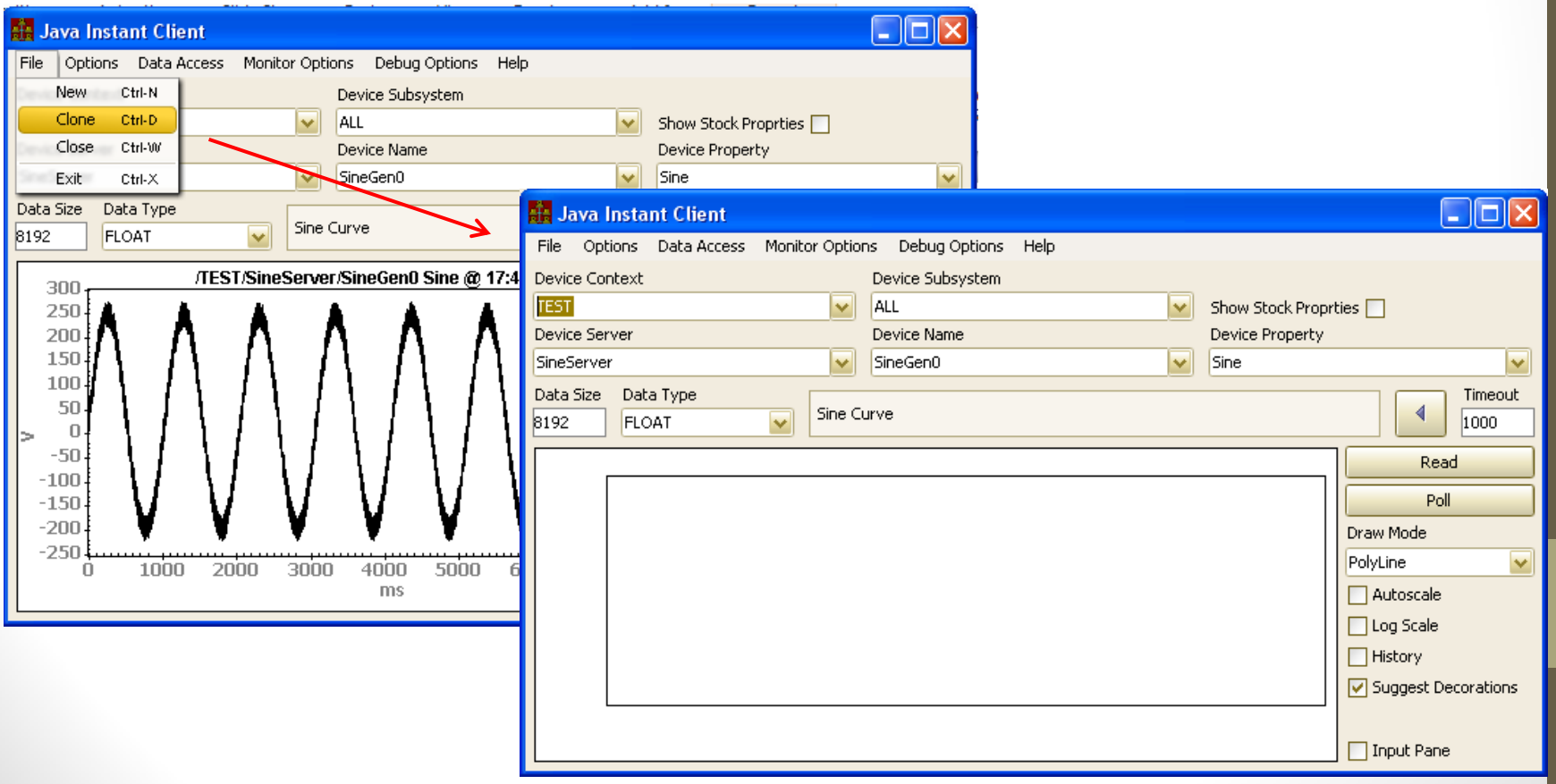
The Instant Client

‘escape’ the characters that belong to the string !



The Instant Client

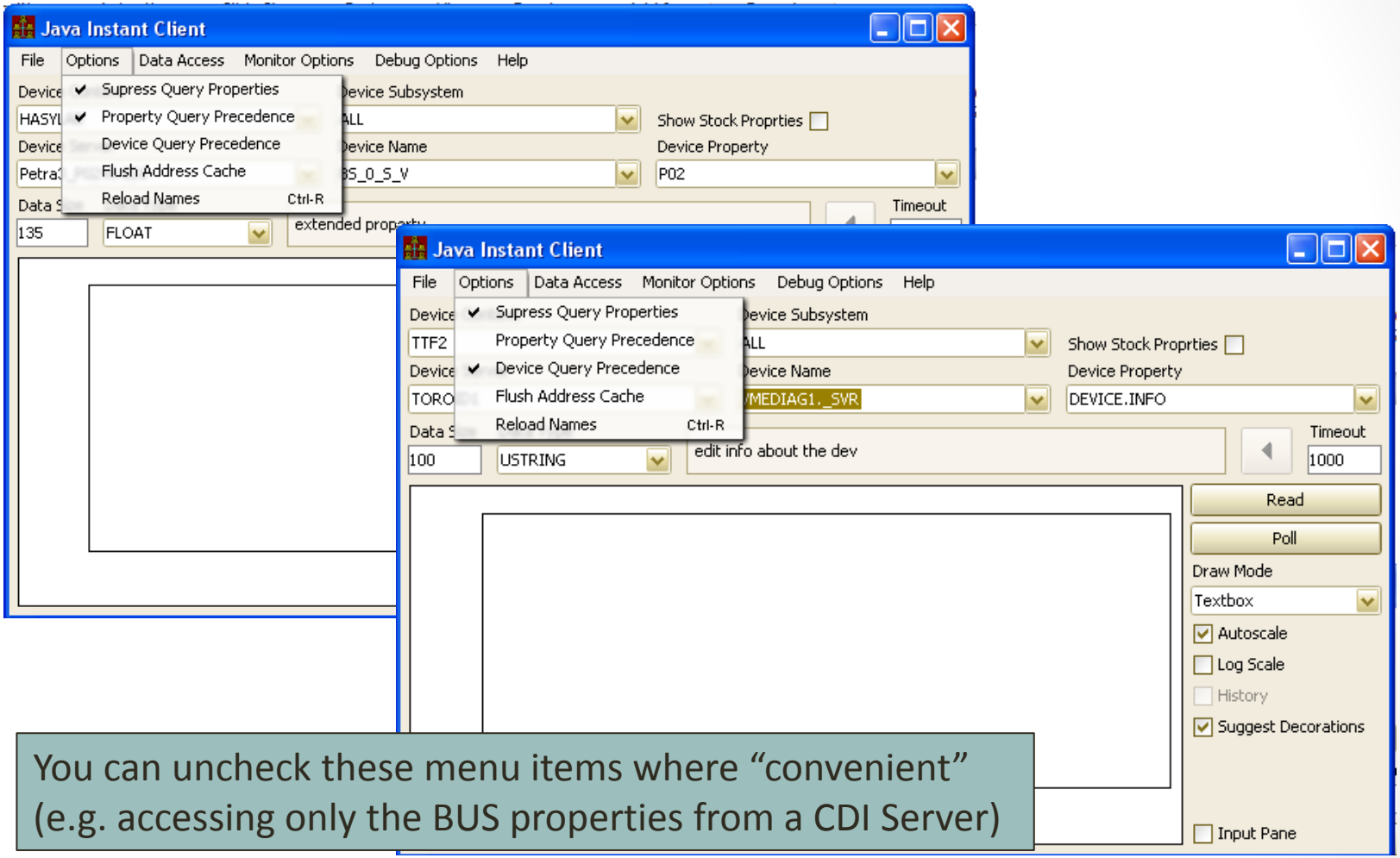
- Menu Items
  - File:
    - ‘New’ launches new Instant Client
    - ‘Clone’ launches new Instant Client and preserves current settings



The Instant Client

- Menu Items
  - Options:
    - Suppress Query Properties -> hide ALL *over-ridden* meta properties
    - Property Query Precedence -> re-acquire *device list* following change of property
    - Device Query Precedence -> re-acquire *property list* following change of device
    - Flush Address Cache -> force new address acquisition from ENS
    - Reload Names -> re-acquire contexts and servers list from ENS
      - Note: re-selecting a context will automatically re-acquire servers for the given context !

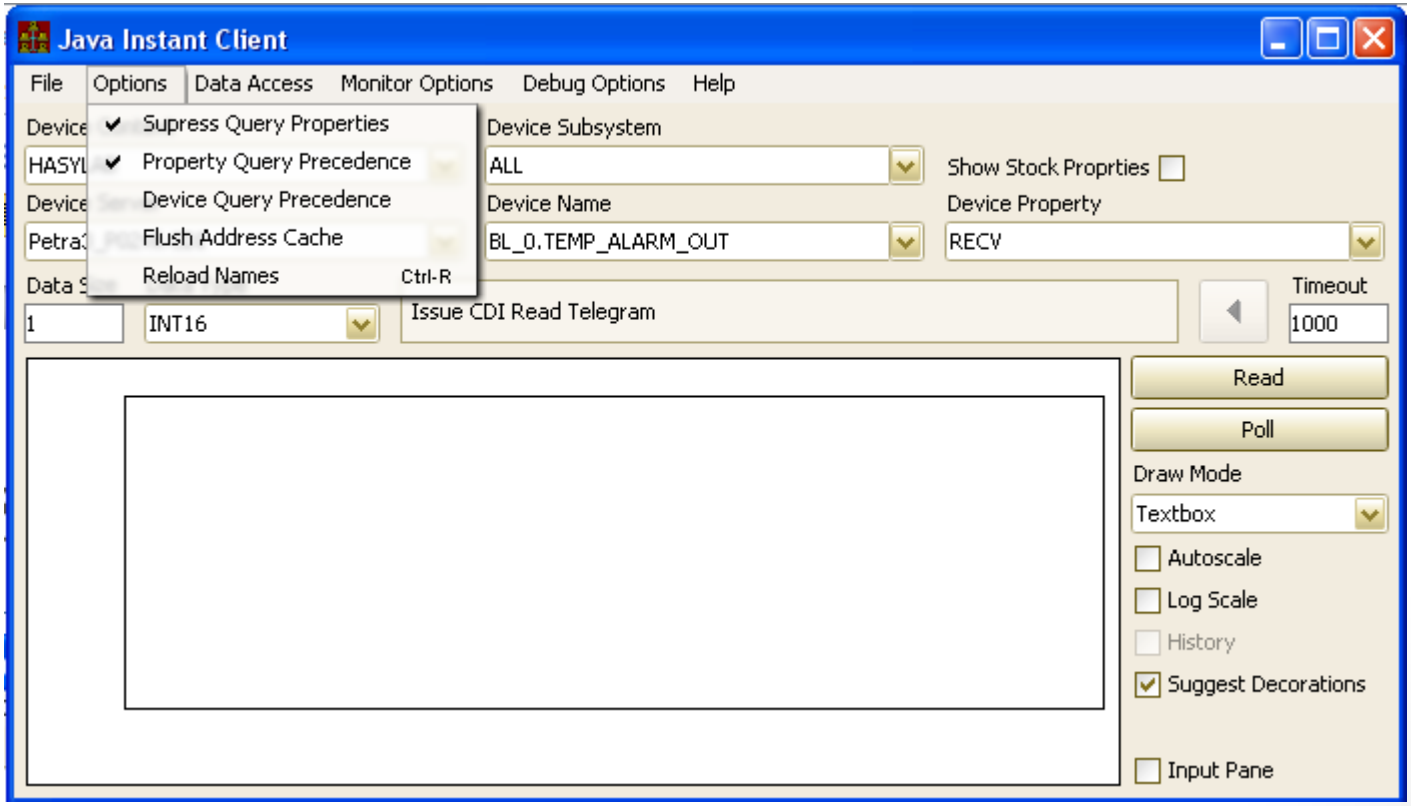
The Instant Client



You can uncheck these menu items where “convenient” (e.g. accessing only the BUS properties from a CDI Server)

The Instant Client

- Switching from property “RECV” to “RECV.CLBR”, “SEND”, “SEND.RECV.ATOM” etc. will automatically re-query devices unless you uncheck “Property Query Precedence” !



The Instant Client

- Menu Items
  - Data Access
    - Use UDP (default) -> normal peer-to-peer communication via UDP
    - Use TCP -> normal peer-to-peer communication via TCP
      - respects given timeout parameters
    - Use STREAM -> peer-to-peer communication via TCP Stream
      - timeout only on connection establishment
      - i/o error only on TCP stack detected error
      - does not otherwise timeout !
      - Note: only multithreaded servers support TINE STREAM transport
    - Use NETWORK flag (multicast)
      - requests transfer per multicast
      - consistent only with UDP and asynchronous data acquisition.

The Instant Client

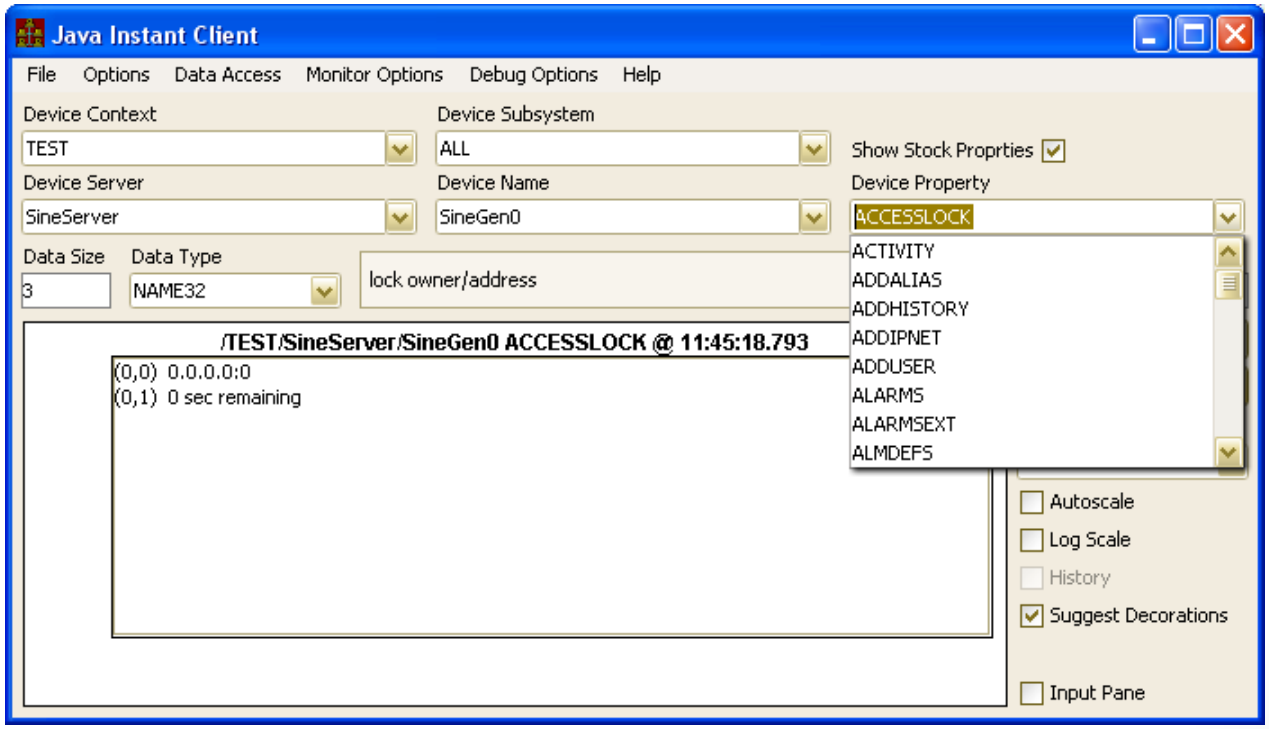
- Menu Items
  - Monitor Options
    - TIMER
      - normal asynchronous (server-side) polling monitor.
      - data sent to caller at the designated timer interval
    - DATACHANGE
      - normal asynchronous (server-side) polling monitor.
      - data examined at the server for changes.
      - data sent to caller if a change has been detected
        - zero-tolerance at the server !
        - caller can supply a notification tolerance (*but NOT the instant client*)
    - EVENT
      - normal asynchronous (server-side) polling monitor.
        - But the polling interval is irrelevant
      - data sent to caller only if it has been scheduled at the server
        - heartbeat updates are suppressed

The Instant Client

- Menu Items
  - Debug Options:
    - Show Fec Information
      - display relevant FEC information of the FEC pertaining to the calling parameters showing.
    - Debug Off
      - Turns local debugging OFF. Closes the TConsole Panel.
    - Debug Level 1 (, 2, 3, 4)
      - Turns local debugging ON at debug level 1 (,2, 3, 4). Opens the TConsole Panel (see debugging).

The Instant Client

- Stock Properties
  - “Show Stock Properties” check box: hidden by default
  - used systematically
    - => you need to know how they work in order to use them yourself!



The Instant Client

- Stock Properties
  - "ACCESSLOCK"
    - Not an attribute ! Separate READ and WRITE behavior !
    - i.e. the property is *overloaded*.
    - Use the API calls instead !
      - SetAccessLock(), GetAccessLockInformation(), GetAccessLockStatus(), FreeAccessLock()
  - "ACTIVITY"
    - READ only
    - Used by FEC Remote Panel
    - Use the overload with the "AQS" structure tag in the Instant Client
      - -> quasi meaningful information for the casual user
  - "ADDALIAS"
    - WRITE only

Scope: Server

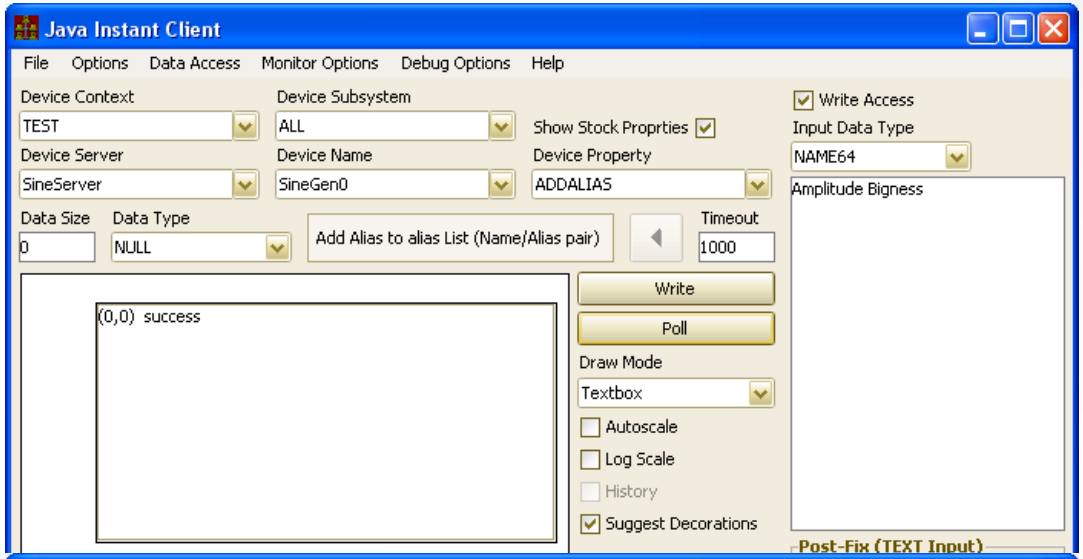
Scope: FEC

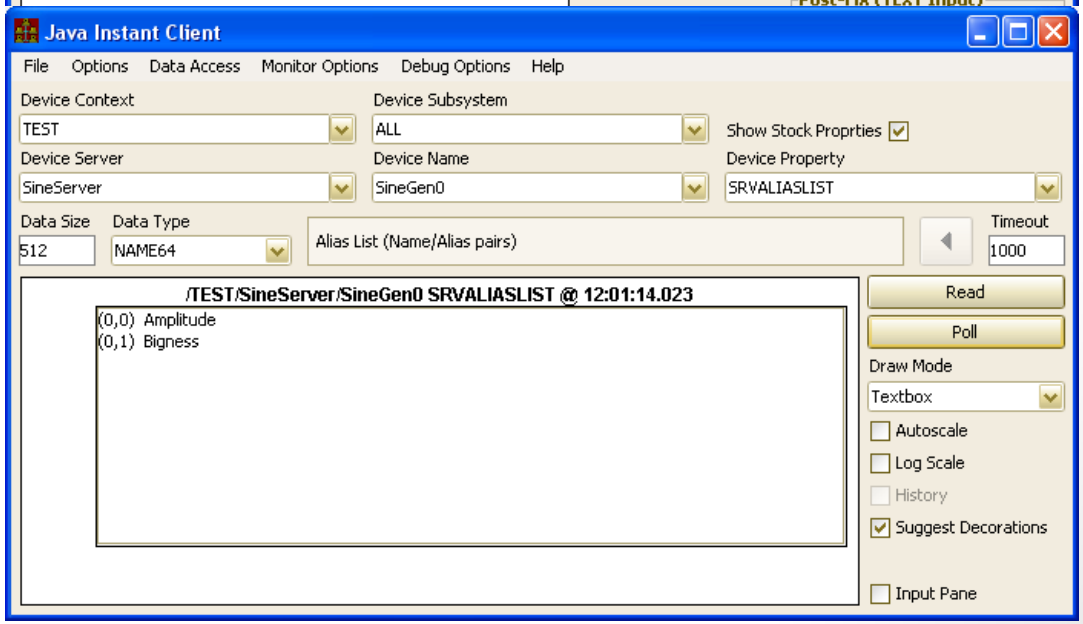
Scope: FEC

The Instant Client

- Stock Properties
  - "ADDALIAS"
    - WRITE only
  - "SRVALIASLIST"
    - READ only

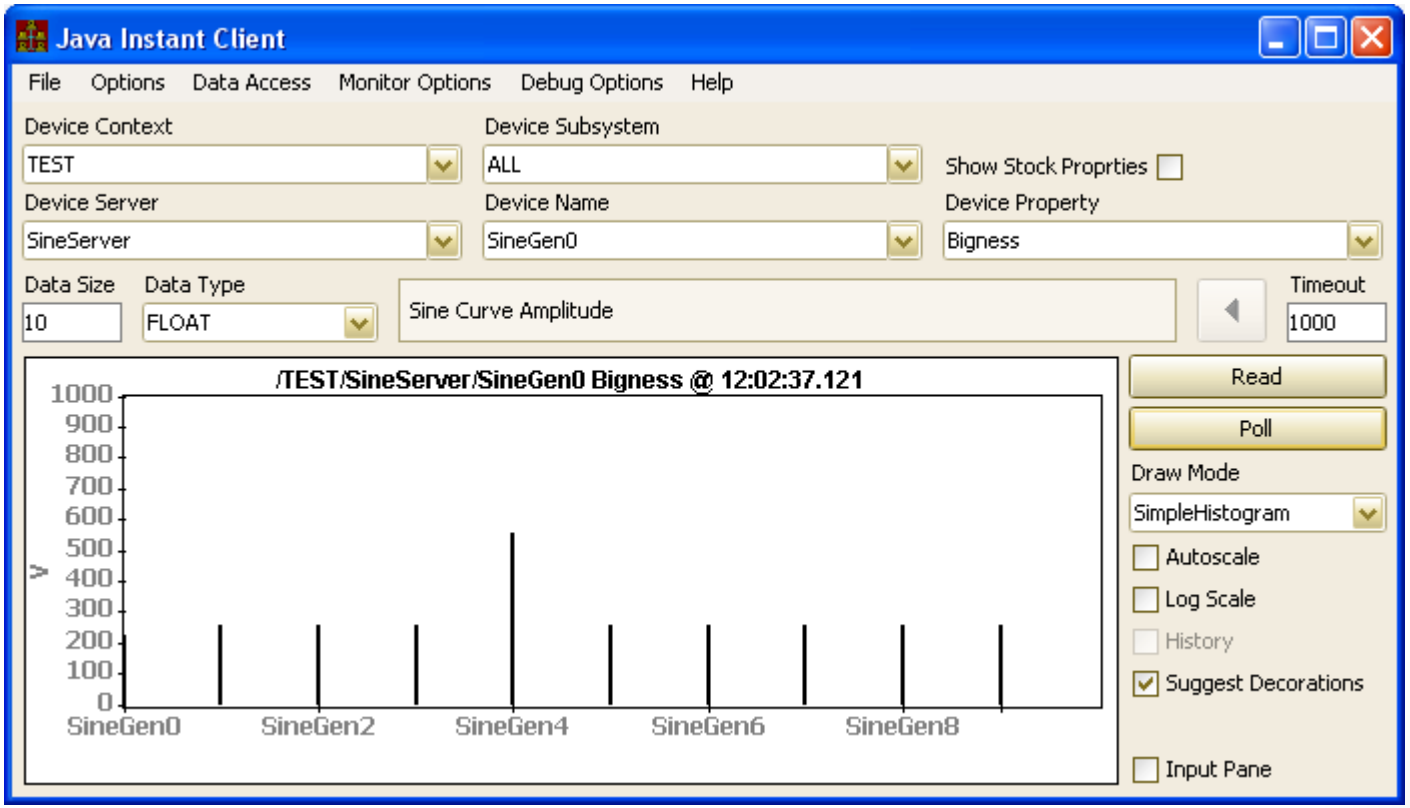
Scope: FEC





The Instant Client

- “Bigness” -> “Amplitude”



The Instant Client

- Stock Properties
  - "ADDHISTORY"

    Scope: Server
    - WRITE only
    - Used in the "Add Local History" panel in the Archive Viewer
    - Don't even think of trying to use this from the Instant Client!
  - "ADDUSER", "ADDIPNET"

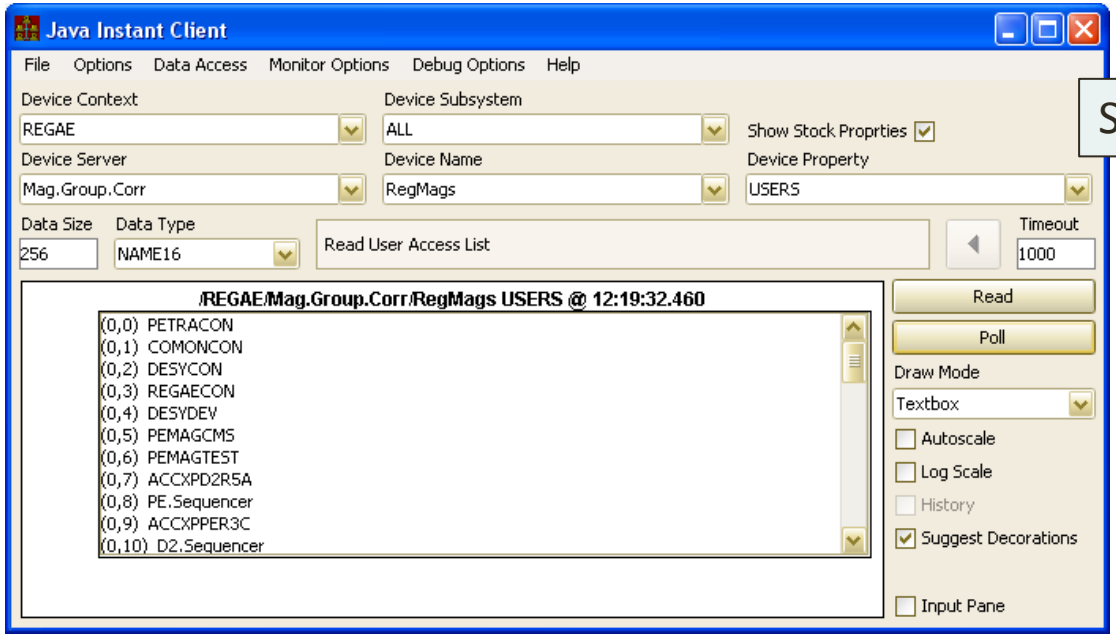
    Scope: Server
    - WRITE only
    - List of users, IP nets (can be CIDR qualified), to add to the ACL lists of the equipment module
    - Updates local database information (non-volatile changes !)
    - No input => forces a re-scan of local database information !
  - "DELUSER", "DELIPNET"

    Scope: Server
    - WRITE only
    - List of users, IP nets to be removed from the ACL lists of the equipment module
    - Updates local database information (non-volatile changes !)

The Instant Client

- Stock Properties
  - "USERS", "IPNETS"
    - READ only
    - no input => returns list of 'allowed' users or IP Nets for the equipment module
    - input = registered Property or Device => returns list of 'allowed' users or IP Nets for the give property or device

Note: "DENIEDUSERS" gives list of users on the denied access list ("these guys can't even READ!)

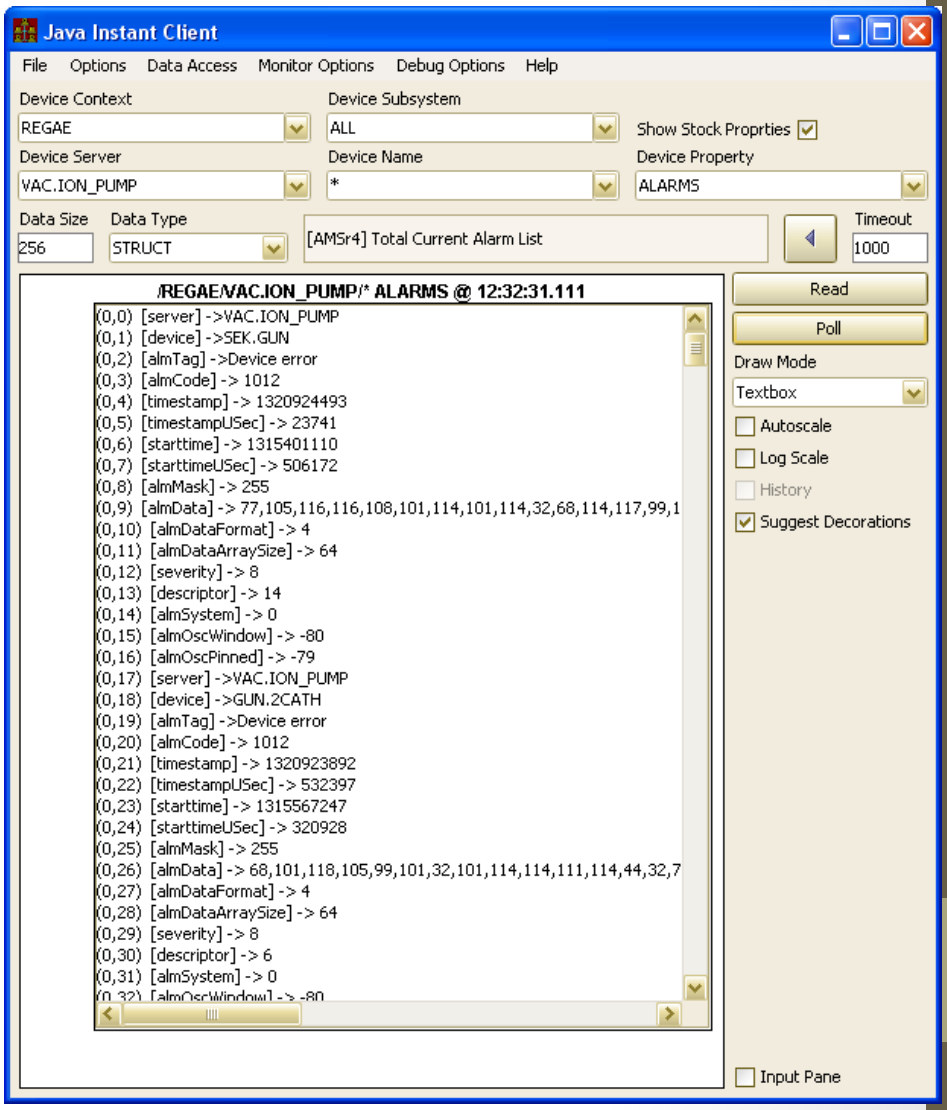


Scope: Server

The Instant Client

- Stock Properties
  - “ALARMS”
    - READ only
    - Used by CAS
    - input
      - none => all alarm times
      - 2 INT32 values (start and stop as UTC timestamps)
    - device = ‘*’ => all alarms
    - overload with structure “AMSr4” of most interest
  - “ALARMSEXT”
    - completely equivalent to “ALARMS”
    - *deprecated*

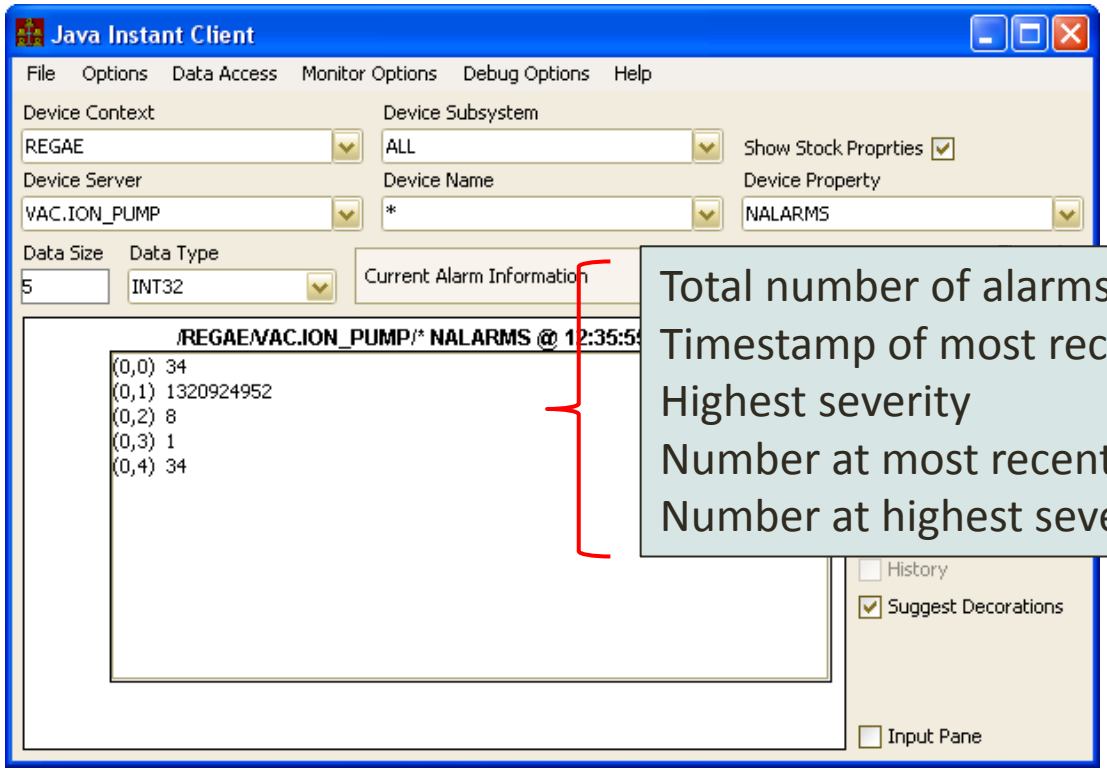
Scope: Server



The Instant Client

- Stock Properties
  - "NALARMS"
    - READ only
    - Array of (up to) 5 INT32 values (see Alarms)

Scope: Server

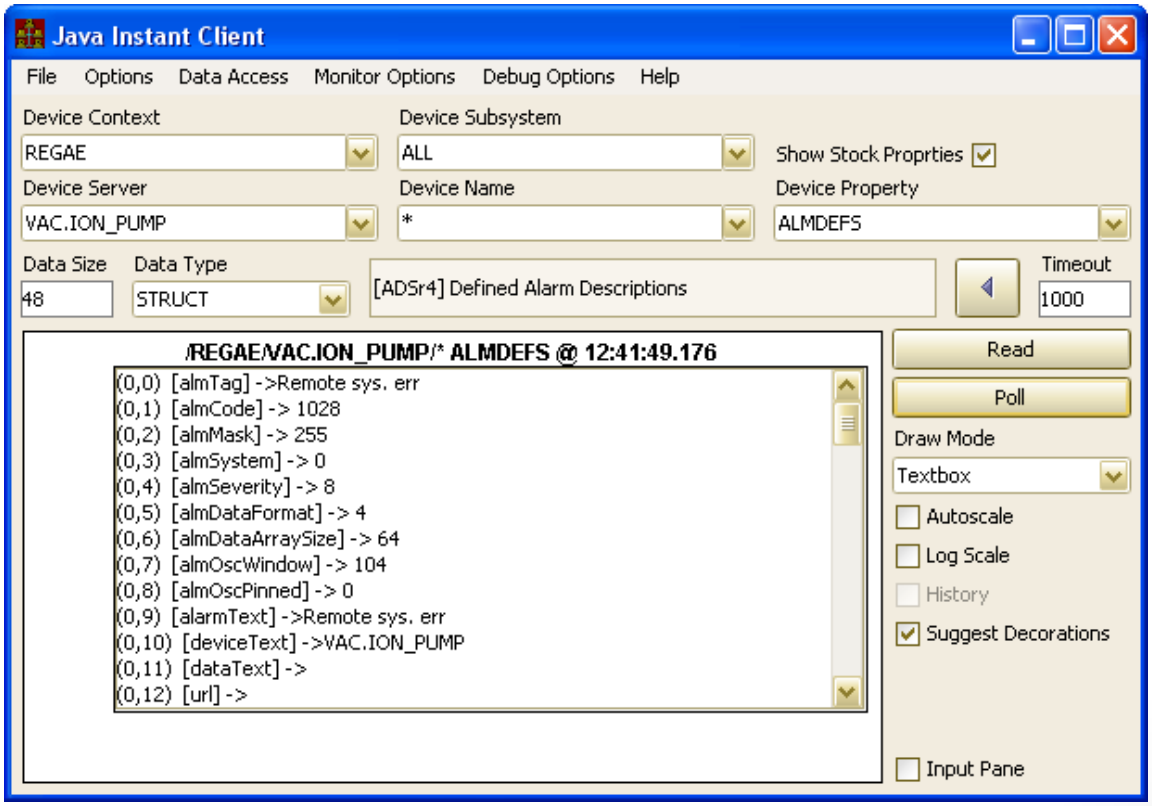


Total number of alarms
Timestamp of most recent
Highest severity
Number at most recent timestamp
Number at highest severity

The Instant Client

- Stock Properties
  - "ALMDEFS"
    - READ only
    - array of static alarm information for defined alarms

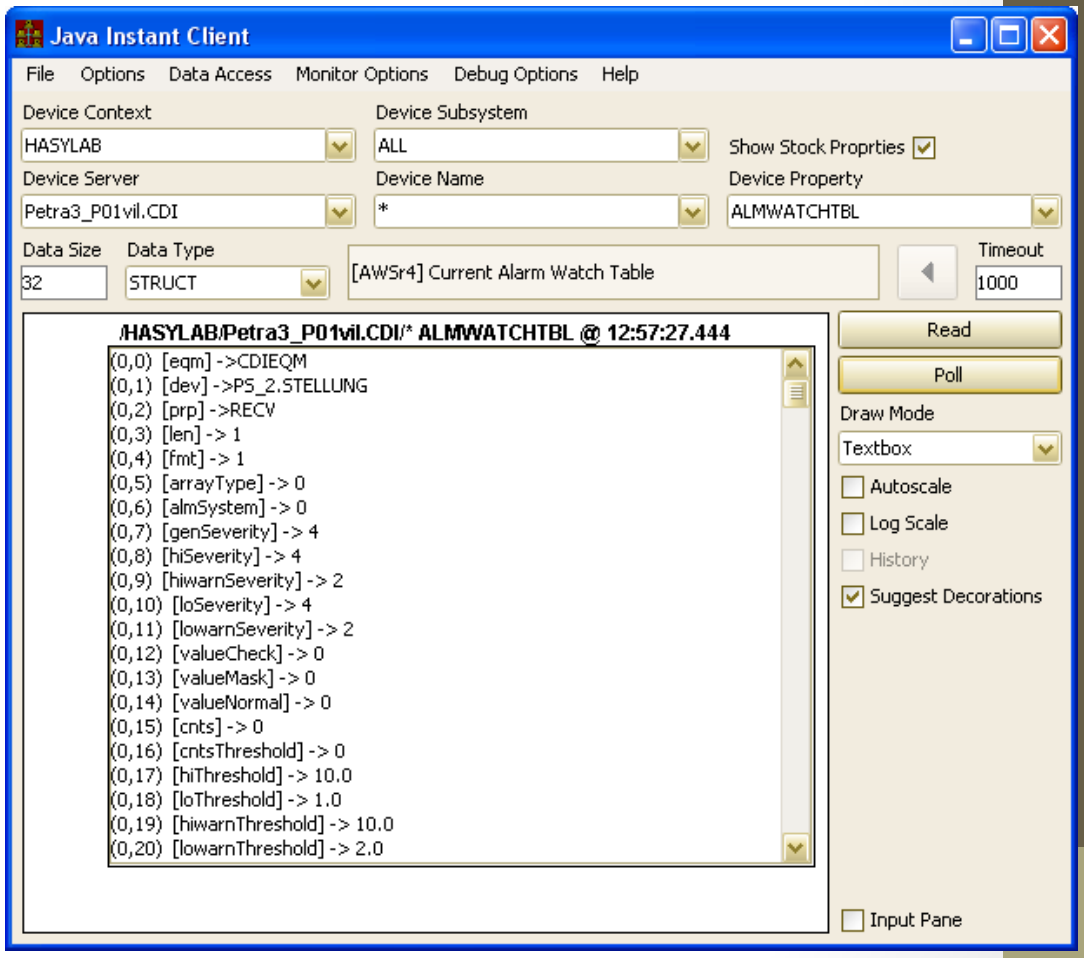
Scope: Server



The Instant Client

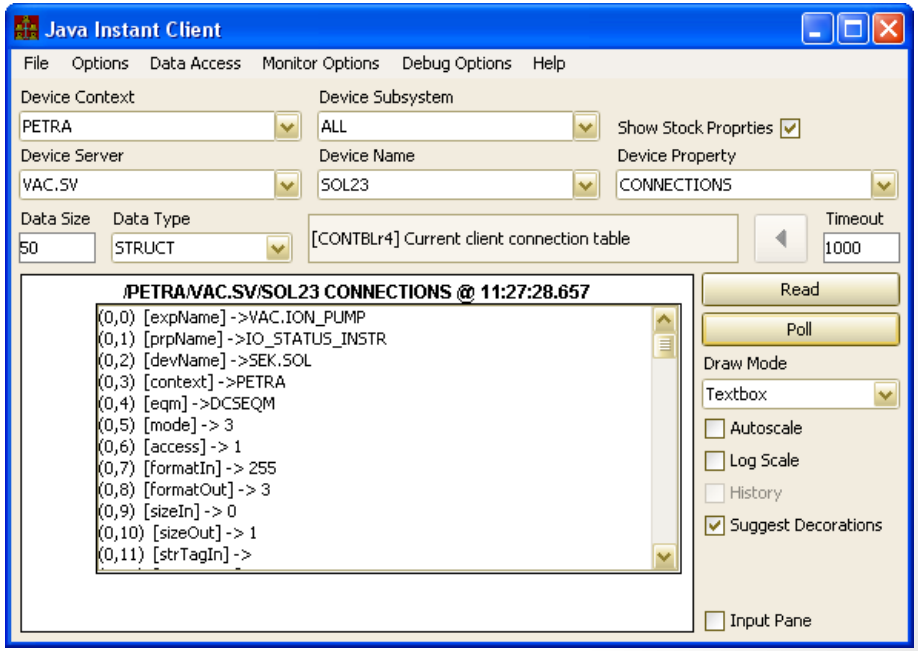
- "ALMWATCHTBL"
  - READ/WRITE
    - But don't try to WRITE with the instant client !
    - WRITE used by Alarm Viewer to add/edit watch table entries
    - READ returns current list of alarm watch table elements

Scope: FEC



The Instant Client

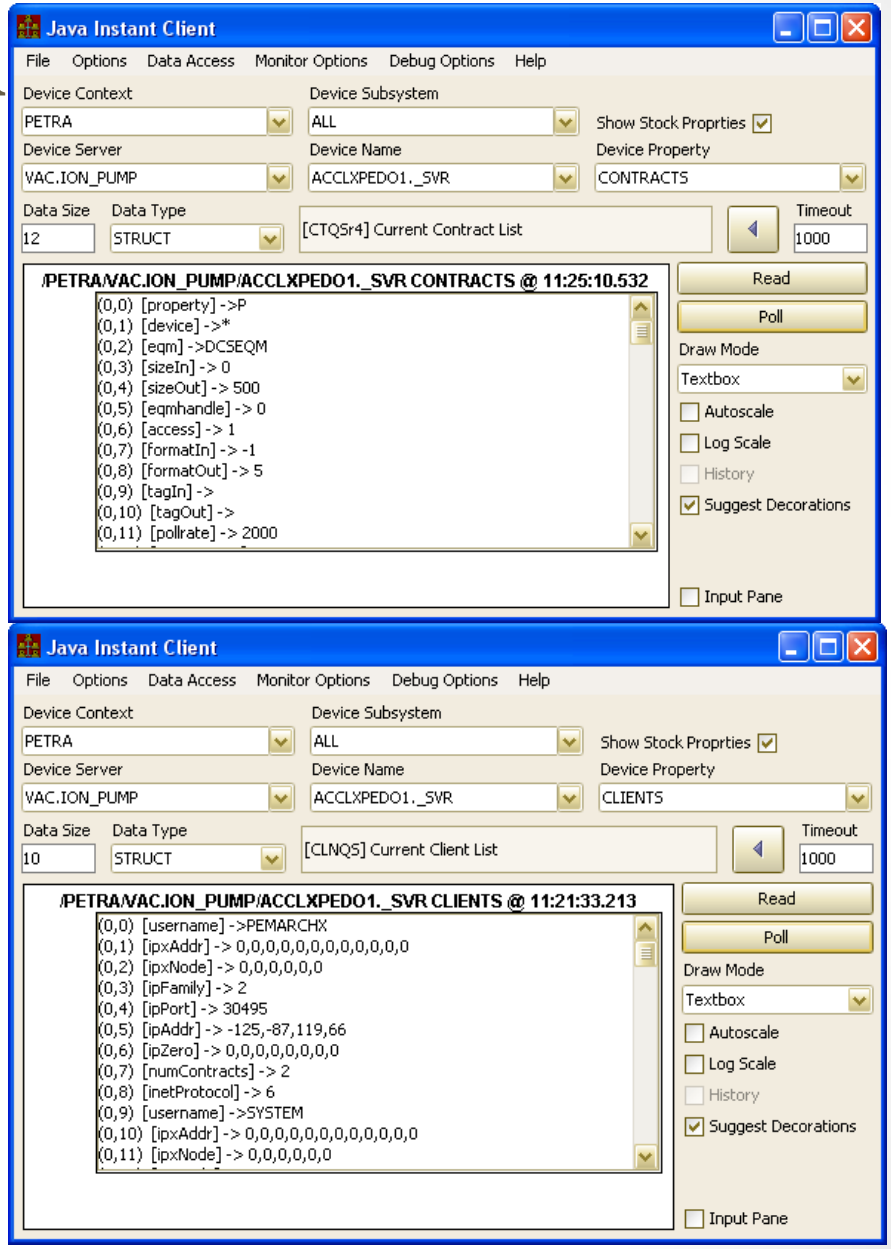
- Stock Properties
  - “NALMDEFS”, “NALMWATCH”, “NDEVICES”, “NPROPS”, “NPROPERTIES”, “NHISTORIES”, “NUSERS”, “NIPNETS”, “NSTOCKPROPS”
    - READ only
    - Return a single INT32 giving the “number of” …
  - “CONNECTIONS”
    - READ only
    - A middle-layer server’s connection table
    - e.g. used by the Tarantula



The Instant Cli

- Stock Properties
  - "CONTRACTS"
    - READ only
    - overloaded => use the "CTQSr4" structure
    - e.g. used by the FEC Remote panel, SPY server
  - "CLIENTS"
    - READ only
    - overloaded => use the "CLNQS" structure
    - e.g. used by the FEC Remote panel, SPY server

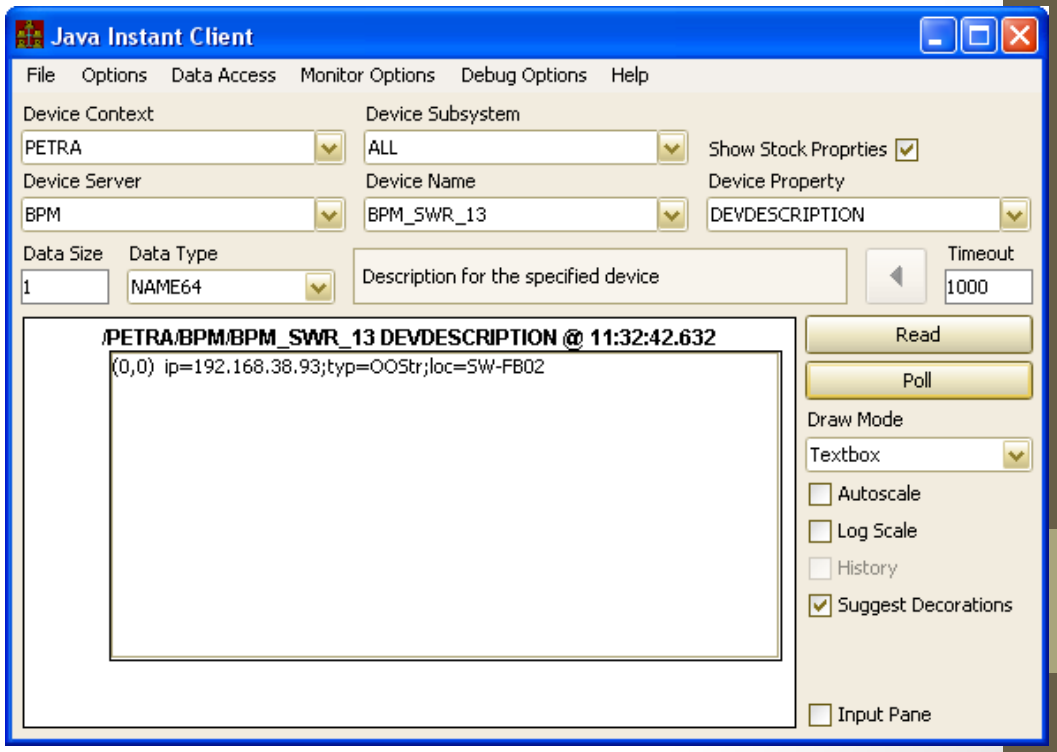
Scope: FEC



The Instant Client

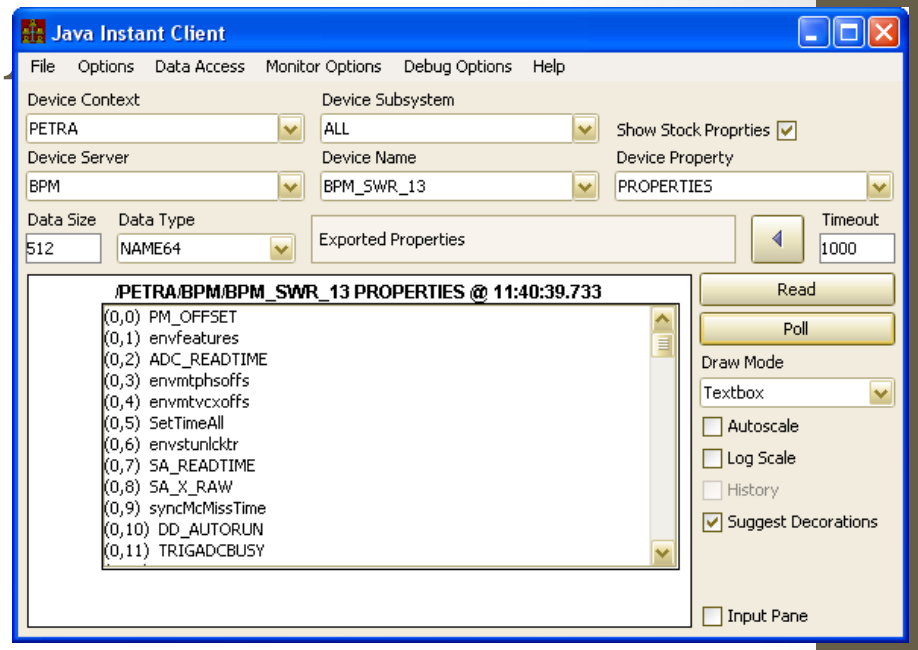
- Stock Properties
  - "DEVICES"
    - READ only
    - server's *registered* device list
  - "DEVDESCRIPTION"
    - READ only
    - A registered device's description
    - e.g. BPM Server:

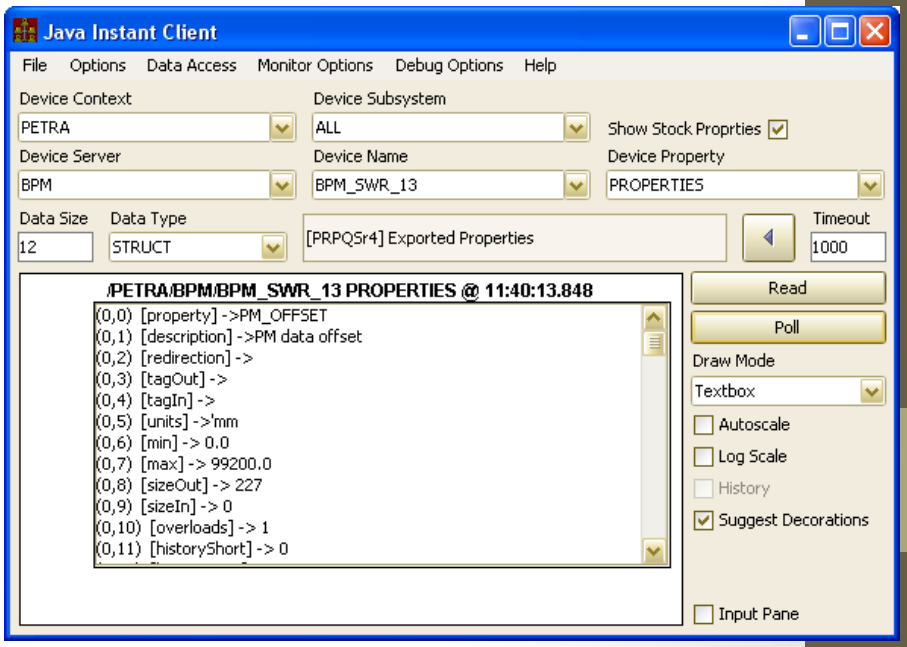
Scope: Server



The Instant Client

- Stock Properties
  - "PROPERTIES" — Scope: Server
    - READ only
    - "PROPS" is an alias
    - multiply overloaded
      - use NAME64 for a list
      - use "PRPQSr4" structure for full info
  - "STOCKPROPS" — Scope: FEC
    - READ only
    - analogous to "PROPERTIES"

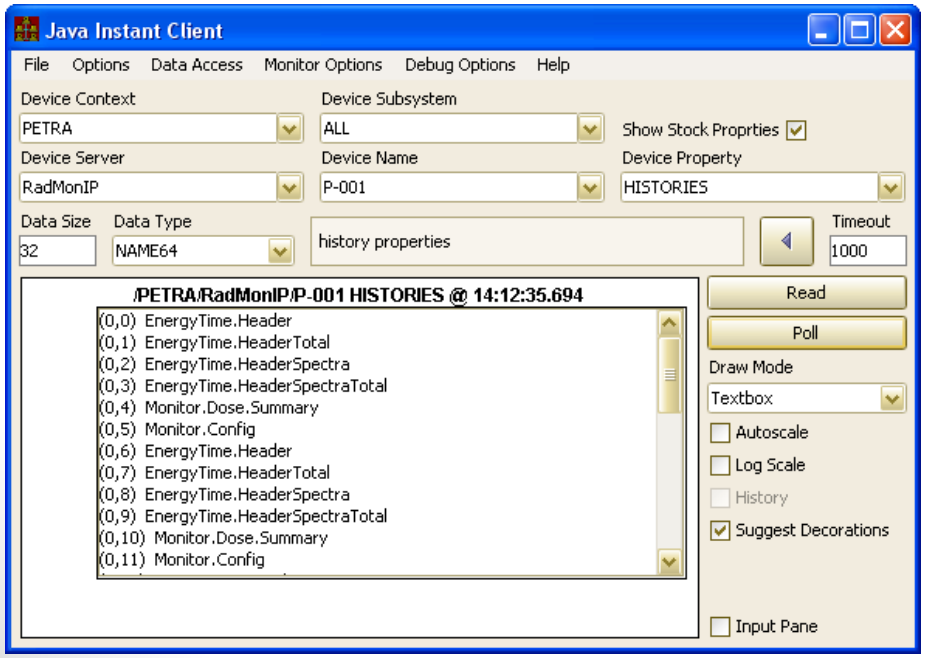


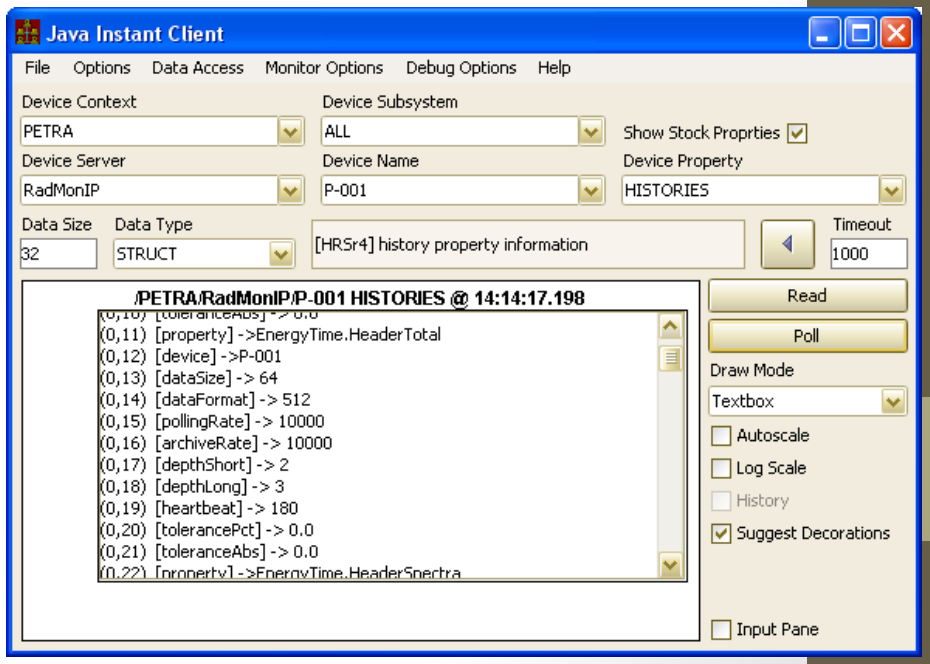


The Instant Client

- Stock Properties
  - "HISTORIES"
    - READ only
    - overloaded
      - use NAME64 for a list of properties with local history
      - Use "HRSr4" structure for detailed information

Scope: Server



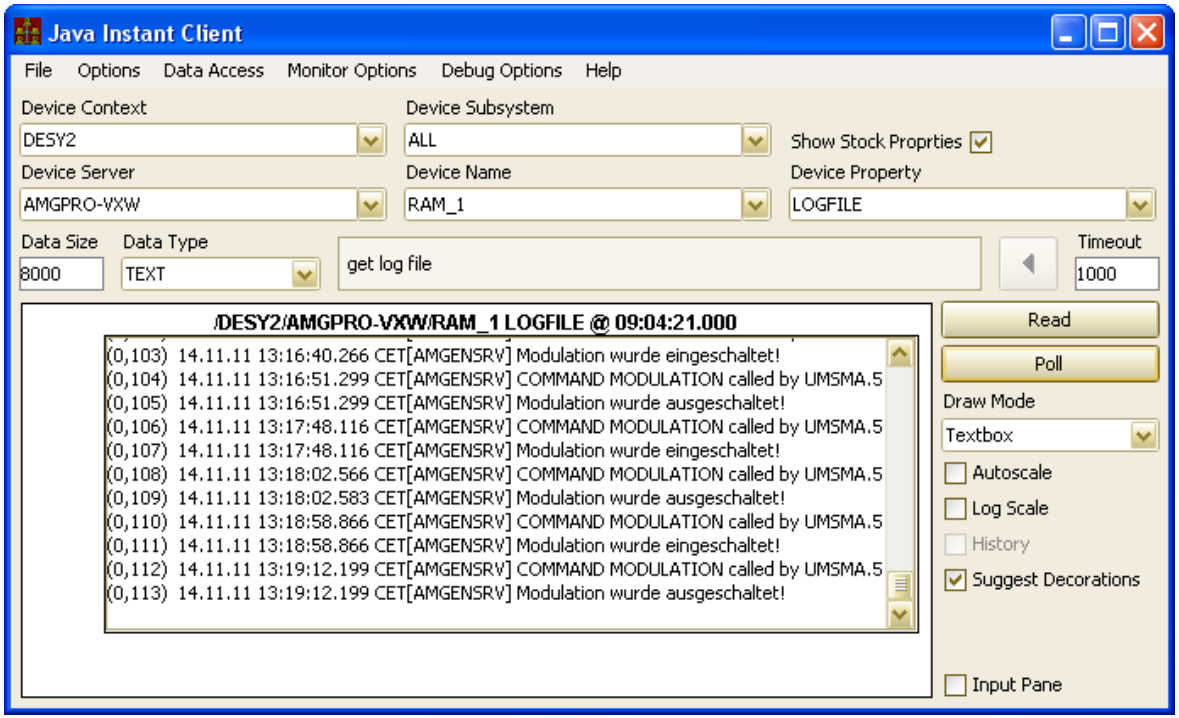


The Instant Client

- Stock Properties
  - "LOGFILE"

Scope: FEC

    - READ only
    - Returns the most recent entries of the current FEC log file
      - number of entries determined by READ buffer size
      - either virtual file (e.g. VxWorks) or disk file (including file rotations)



The Instant Client

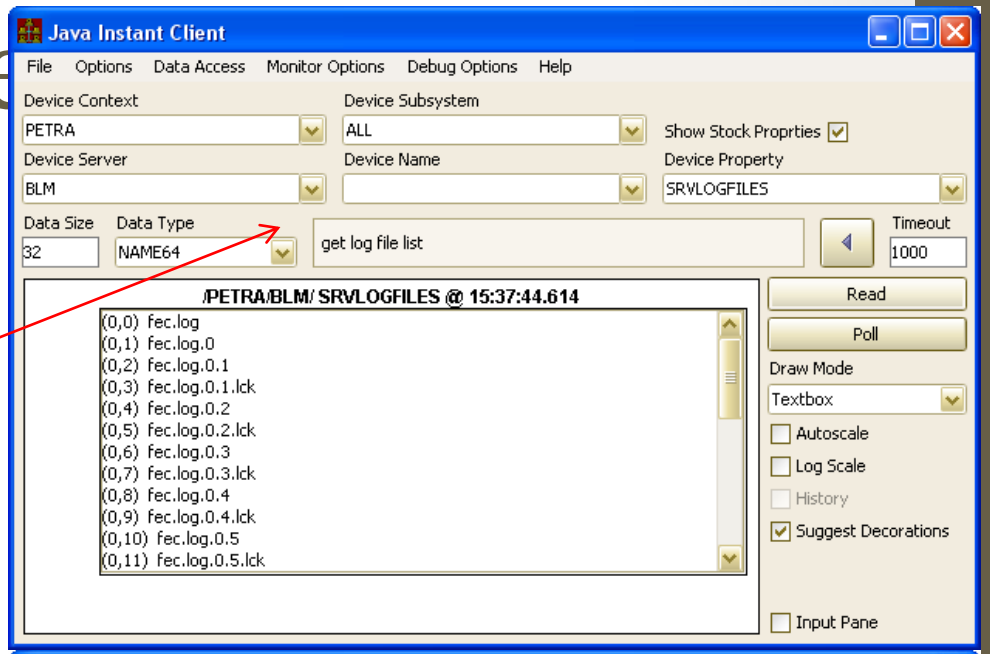
- Stock Properties
  - "LOGCOMMANDS"
    - READ/WRITE
    - en/dis-able COMMAND logging inside 'fec.log'
  - "LOGDEPTH"
    - READ/WRITE
    - specify depth of fec.log file (before rotation)
  - "SRVLOGFILES"
    - READ only
    - List of 'text'-based log files
      - e.g. *.log, *.csv, *.txt
    - input:
      - via input data or 'device name'
      - none => use the FEC_HOME location
      - otherwise => find files in the path given.

Scope: FEC

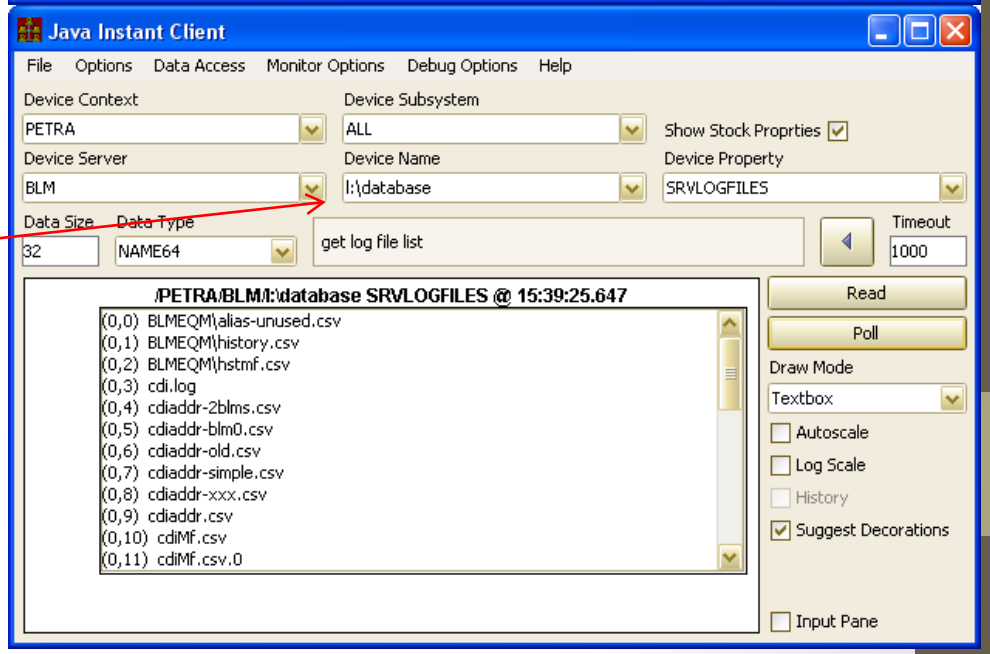
The Instant Clie

- "SRVLOGFILES"
  - READ only

No path = FEC_HOME path



Path = I:\database
(=> I know this is a windows system)



The Instant Client

- Stock Properties
  - "SRVLOGFILE"

    Scope: FEC
    - READ only
    - input:
      - via input data or 'device name'
      - none => fec.log from the FEC_HOME location
      - otherwise => the file given
    - return:
      - the (text) file input (most recent N bytes; N = requested data size)
  - "SRVBINFILE"
    - READ only
    - input:
      - via input data or 'device name'
      - none => fec.log from the FEC_HOME location
      - otherwise => the file given
    - return:
      - the (binary) file input (most recent N bytes; N = requested data size)

The Instant Client

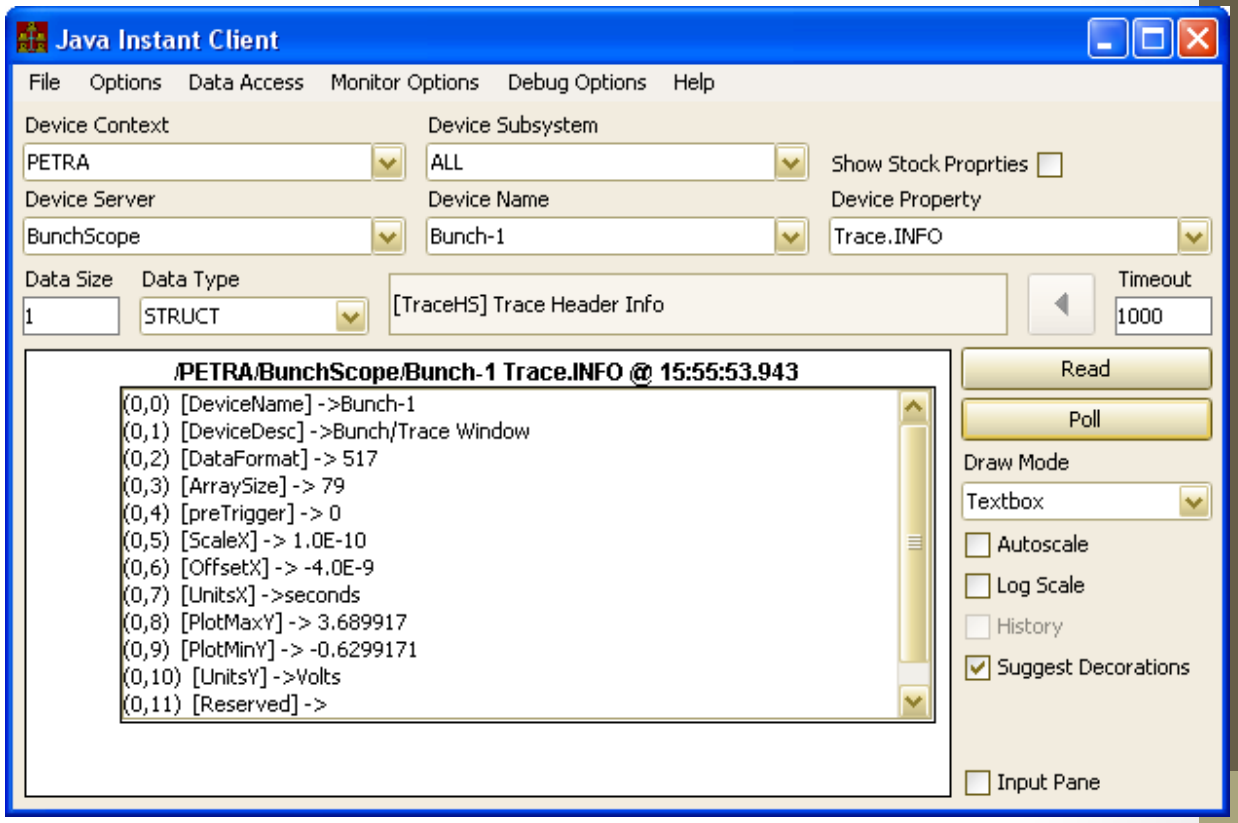
Scope: FEC

- Stock Properties
  - “STRUCTFORMAT”
    - READ only
    - input = ‘’tag” of interest
    - structure information
      - used in data type ‘discovery’
  - “BITFIELDFORMAT”
    - READ only
    - input = ‘’tag” of interest
    - bitfield information
      - used in bitfield ‘discovery’

The Instant Client

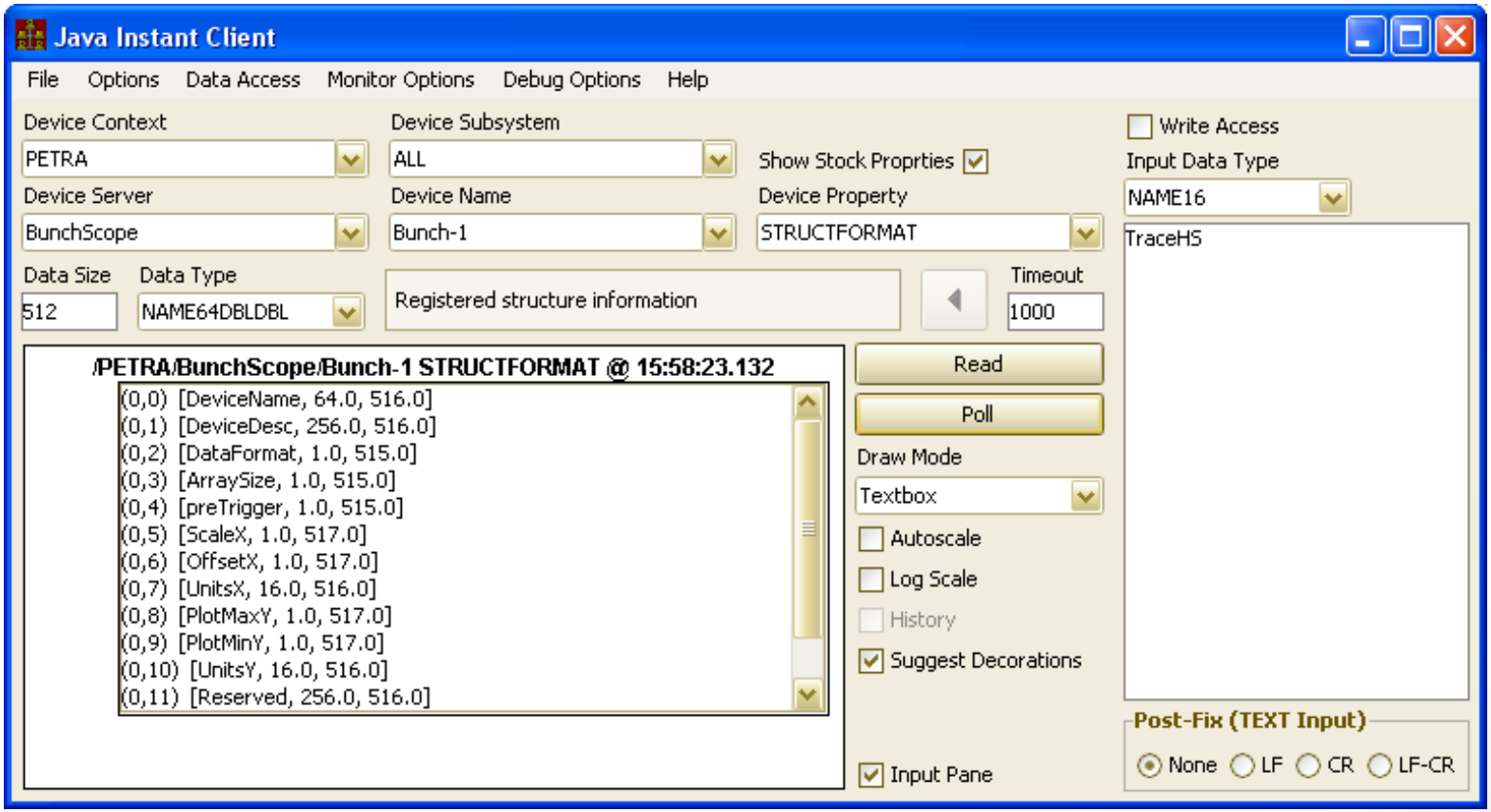
Tagged structure acquisition

- e.g. “How does this work ?



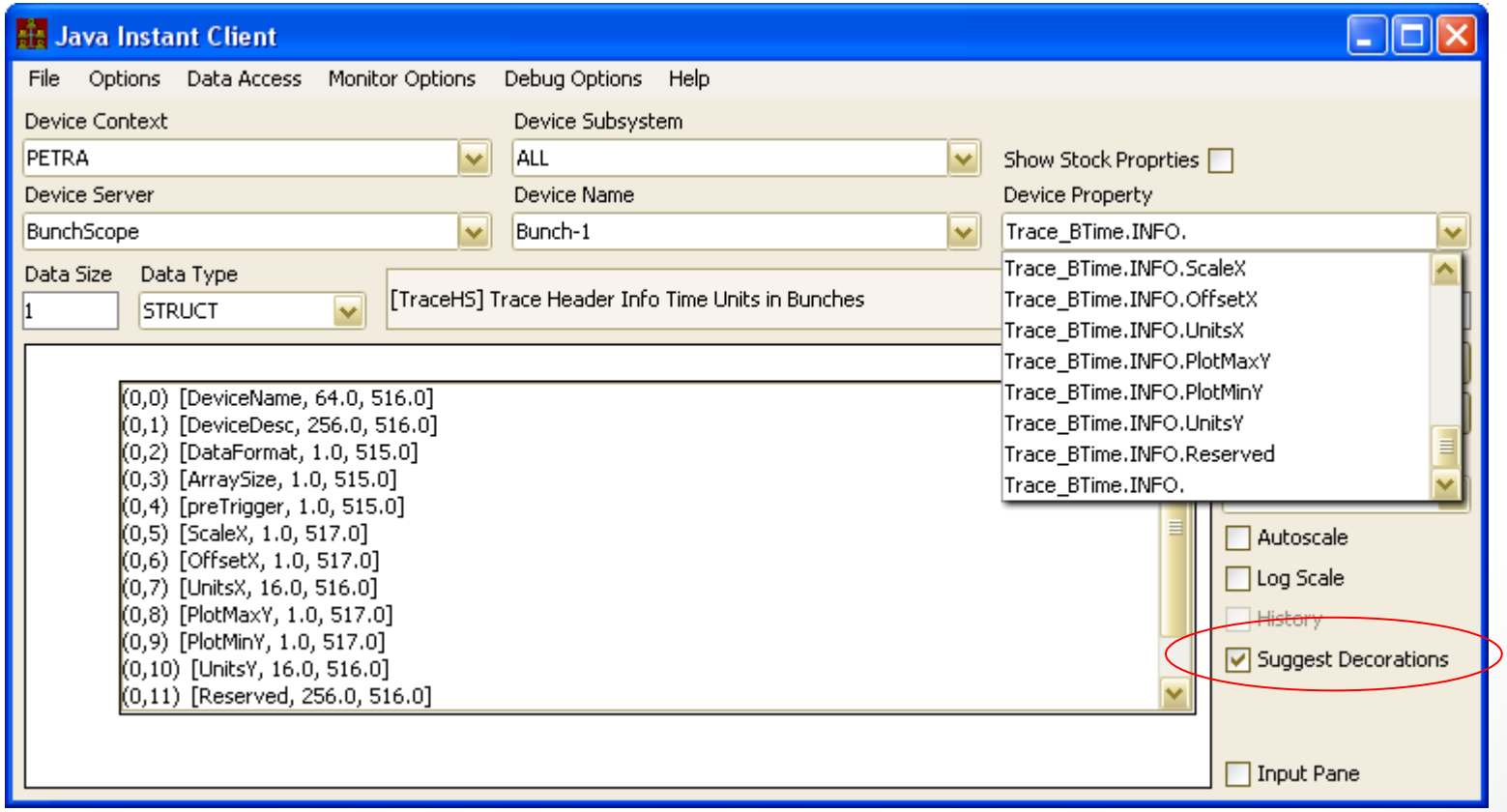
The Instant Client

'learn' the structure composition via "STRUCTFORMAT"



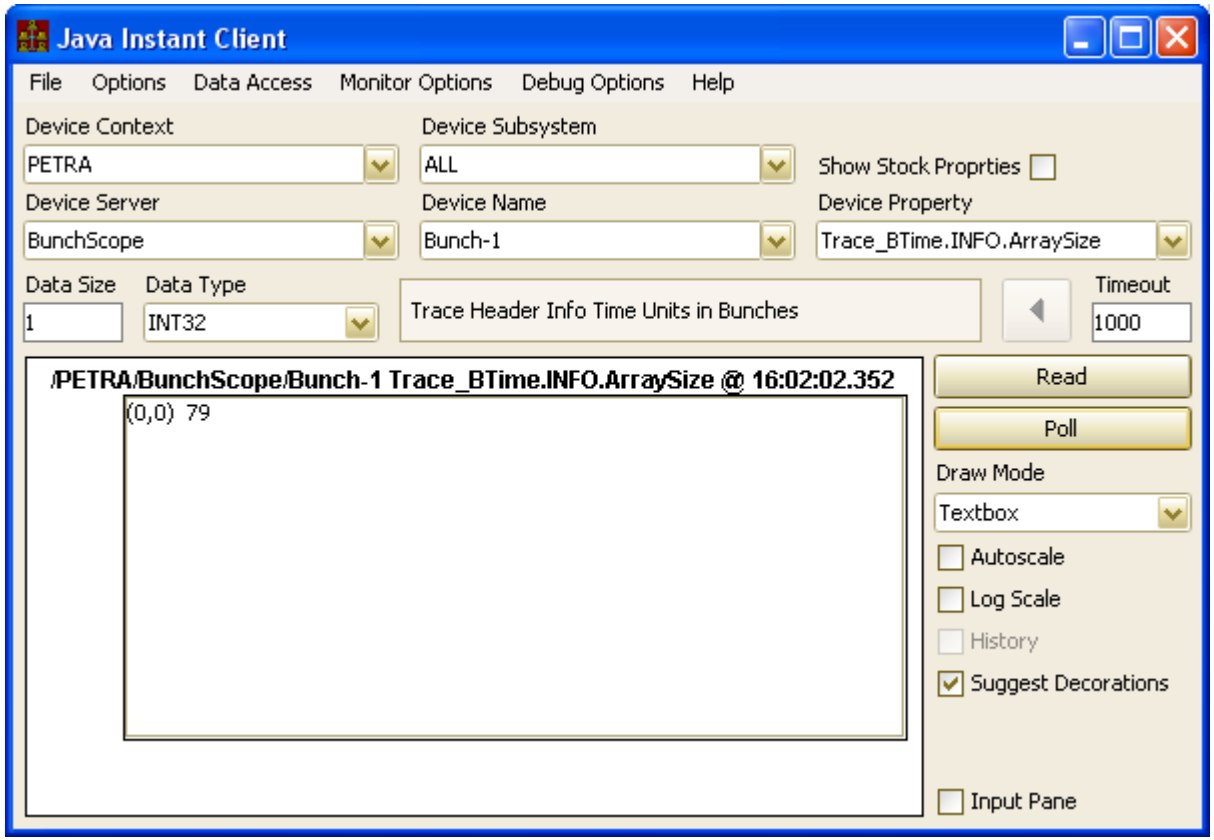
The Instant Client

"Intelligent" browsing :



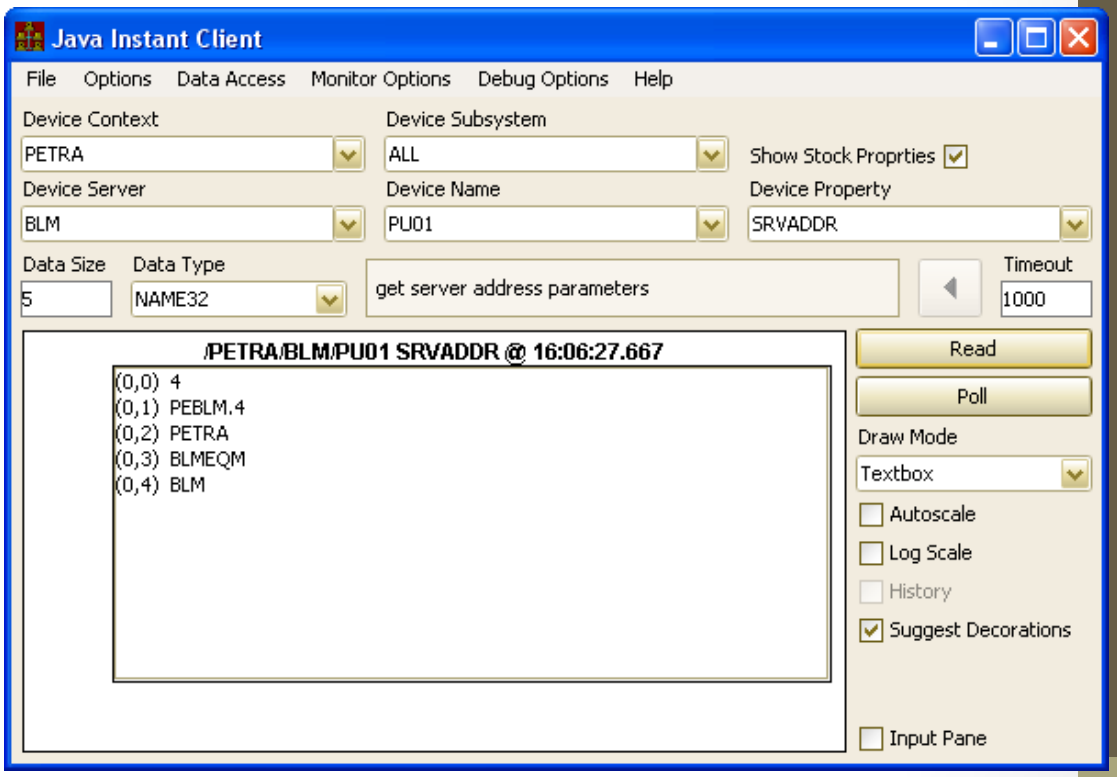
The Instant Client

Can obtain individual ‘fields’ :



The Instant Client

- Stock Properties
  - SRVADDR
    - READ only
    - Output:
      - Up to 5 NAME32 entries
        - port offset
        - FEC name
        - context
        - local name
        - server name



The Instant Client

- Stock Properties
  - SRVOS
    - READ only
    - return: server (i.e. FEC) OS
  - SRVLOCATION
    - READ only
    - return: server (i.e. FEC) location
  - SRVDESC
    - READ only
    - return: server description
  - SRVSUBSYSTEM
    - READ only
    - return: server subsystem

Scope: FEC

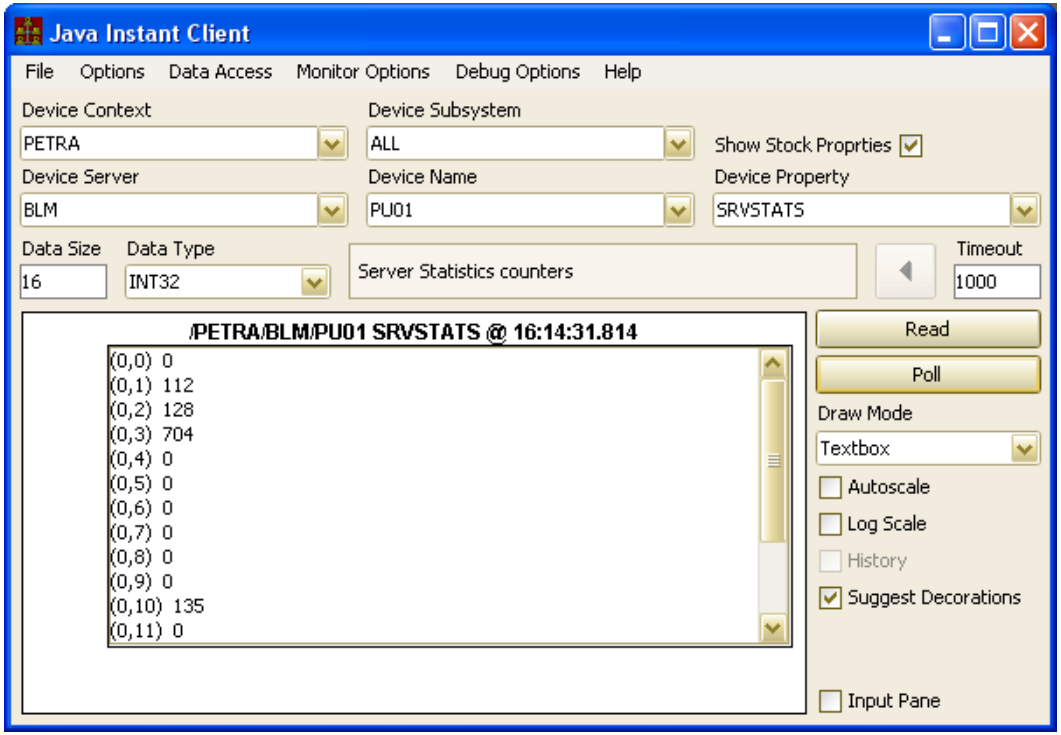
Scope: server

The Instant Client

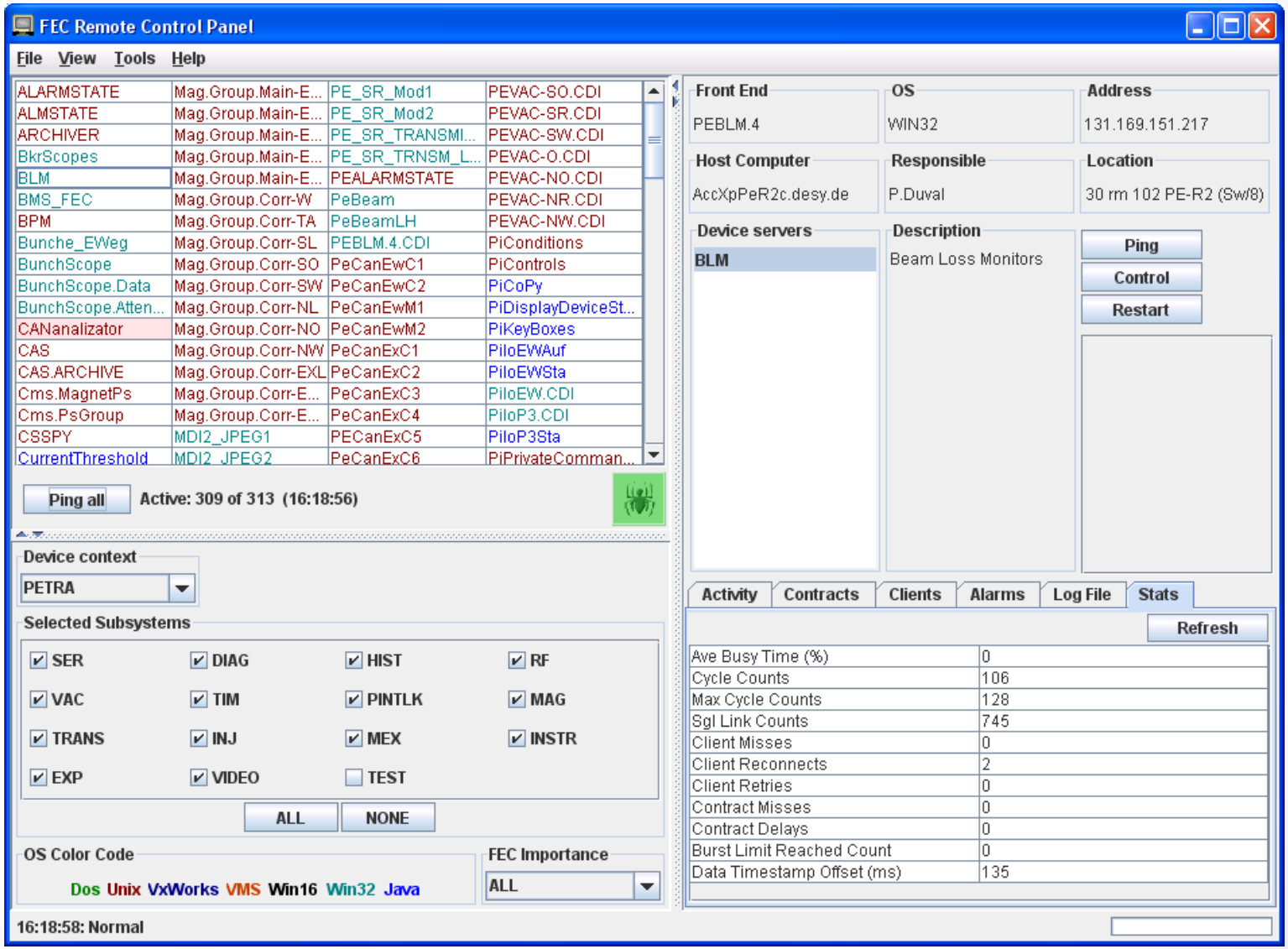
- Stock Properties
  - SRVSTATS

Scope: FEC

    - READ only
    - return: a ‘collection’ array of 16 INT32s
    - ‘you have to know what they are’ (the FEC panel does)



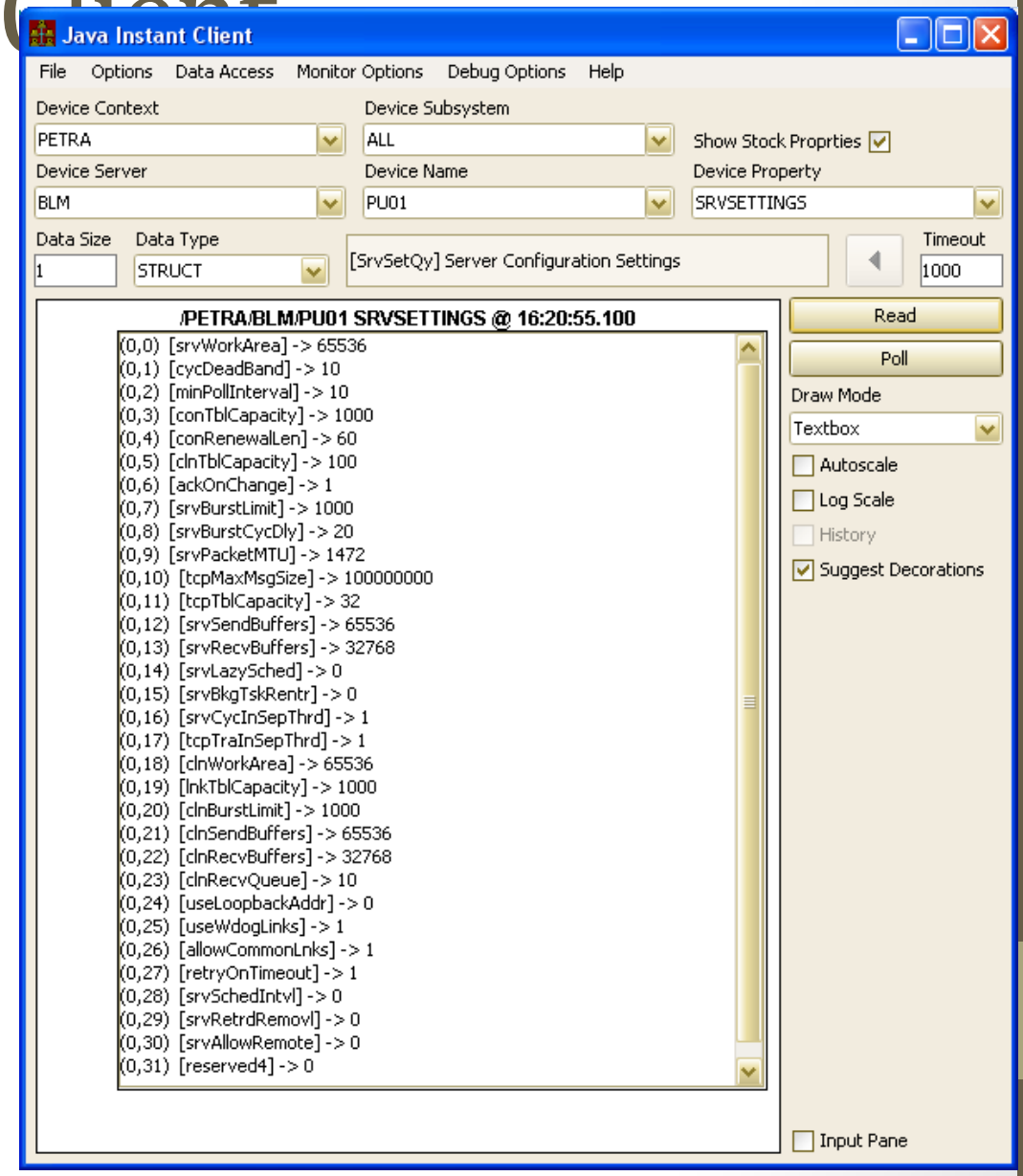
The Instant Client



The Instant Client

- Stock Properties
  - SRVSETTINGS
    - READ only
    - configuration settings.

Scope: FEC



The Instant Client

- Stock Properties

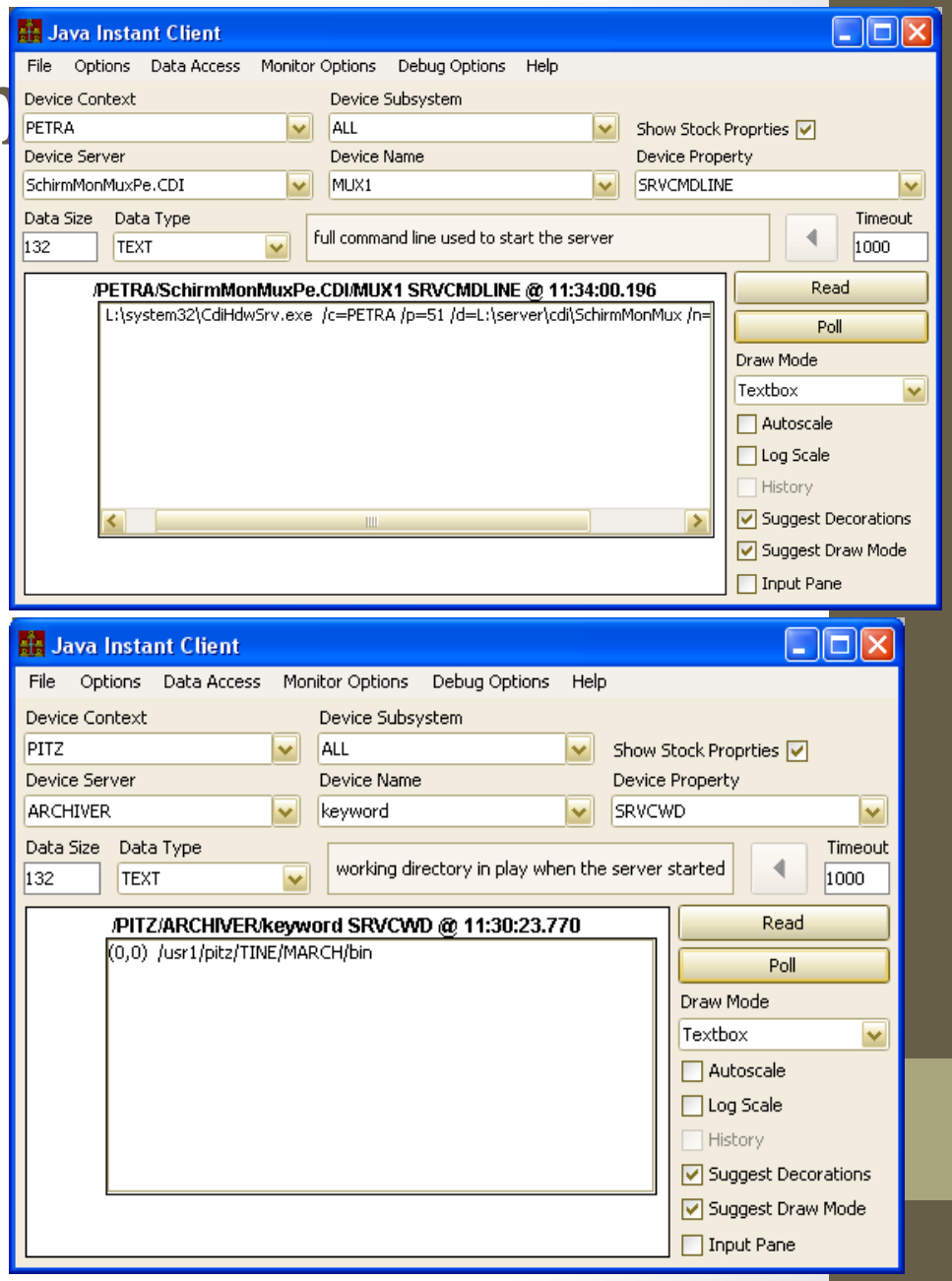
  Scope: FEC

  - SRVSTARTTIME
    - READ only
    - return: server startup time
  - SRVPID
    - READ only
    - return: server process id

The Instant Clien

- Stock Properties
  - SRVCMDLINE
    - READ only
    - return: server command line
  - SRVCWD
    - READ only
    - return: server current working directory

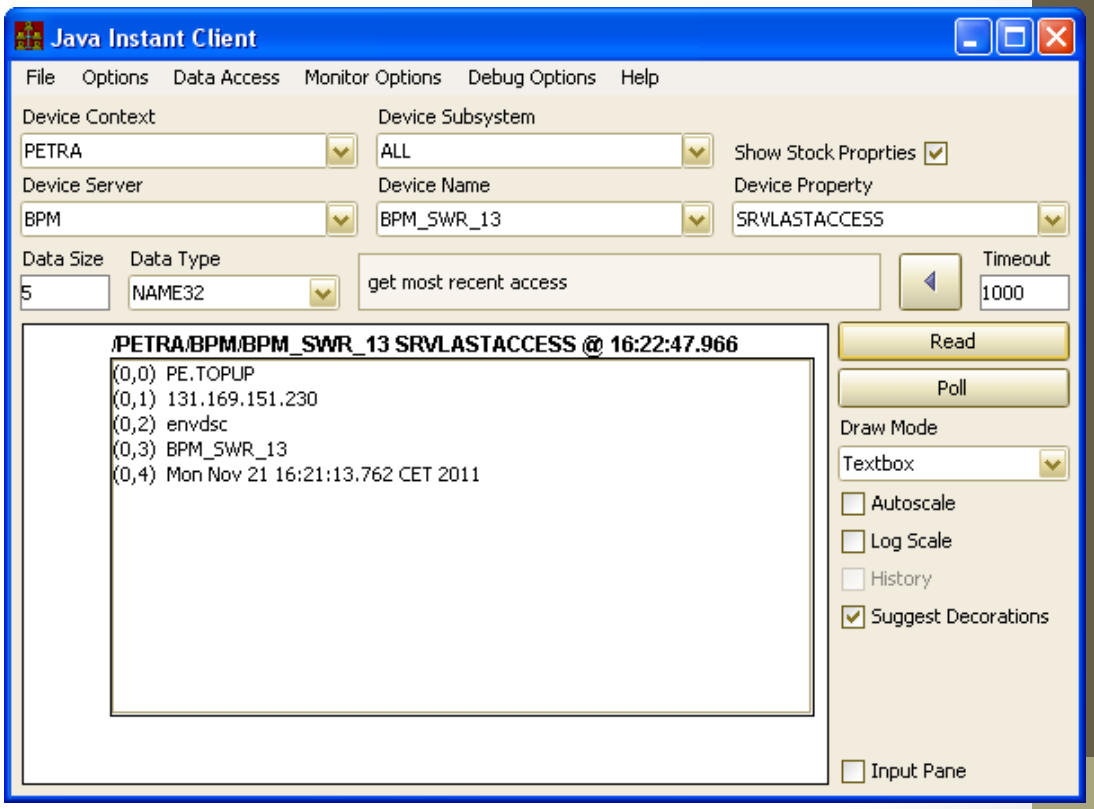
Scope: FEC



The Instant Client

- Stock Properties
  - SRVLASTACCESS
    - READ only
    - return:
      - up to 5 NAME32 values
      - info related to last WRITE access.
        - user
        - net address
        - property called
        - device called
        - access time.

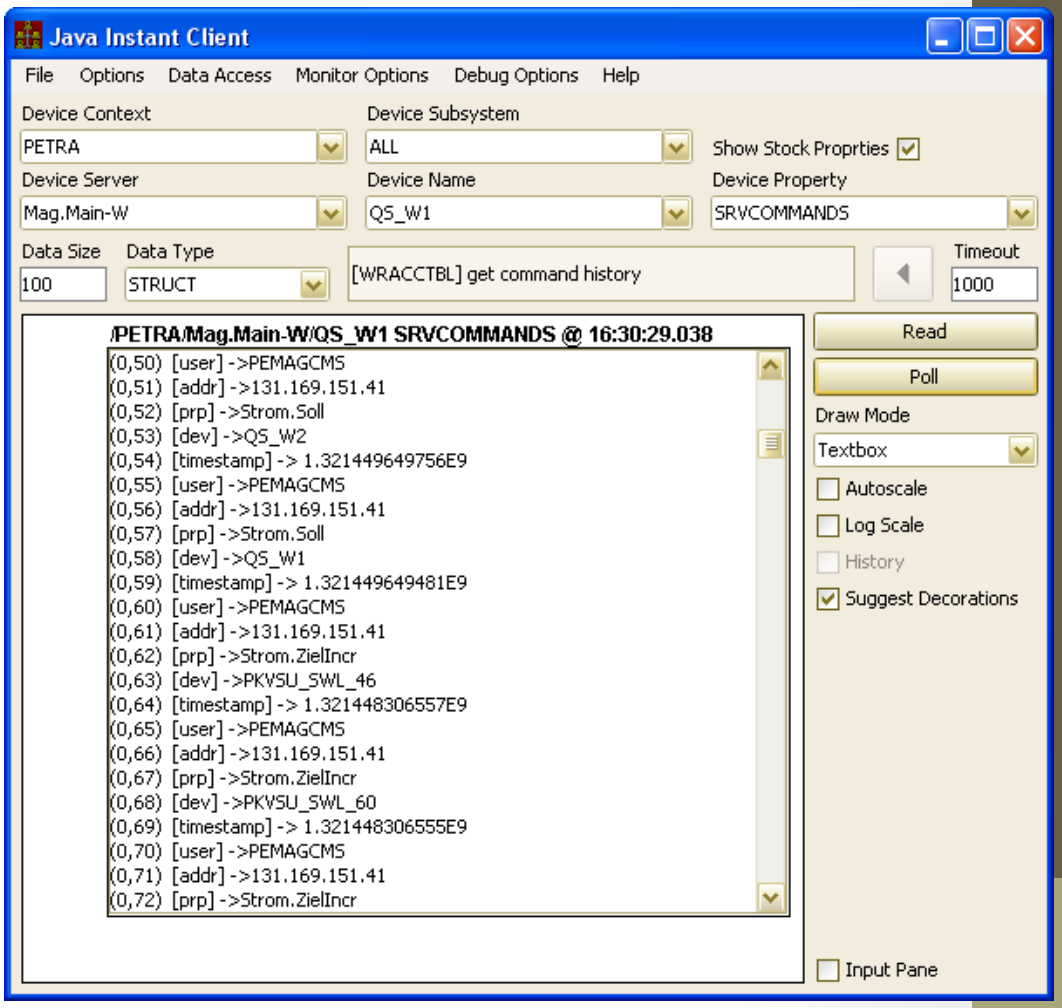
Scope: server



The Instant Client

- Stock Properties
  - SRVCOMMANDS
    - READ only
    - return up to 100 of the most recent WRITE commands (struct “WRACCTBL”)

Scope: server



The Instant Client

- Stock Properties
  - DEBUGLEVEL
    - READ/WRITE
    - Sets/gets the debug level at the FEC
  - SRVEXIT
    - WRITE only
    - must be enabled (!) e.g. “SetAllowRemoteManagement(TRUE)”
    - input:
      - 1 INT32 value gives the exit level
      - no input => exit level = 0

Scope: FEC

The Instant Client

- Stock Properties
  - SRVIDLE
    - READ/WRITE
    - must be enabled (!) e.g. “SetAllowRemoteManagement(TRUE)”
    - input: 1 INT32 value
      - 0 => not idle
      - non-zero => idle
    - an idle server will NOT call any eqm dispatch routines
    - attempts to access the server receive ‘server_idle’
  - SRVINIT
    - WRITE only
    - must be enabled (!) e.g. “SetAllowRemoteManagement(TRUE)”
    - no input
    - Calls a server’s registered ‘init’ routine.
  - SRVRESET
    - WRITE only
    - must be enabled (!) e.g. “SetAllowRemoteManagement(TRUE)”
    - no input
    - de-registers all equipment modules and returns all memory to the heap.
    - Calls any registered ‘PostSystemInit()’ routine.

Scope: server

Scope: FEC

The Instant Client

- Stock Properties
  - DEBUGLEVEL
    - READ/WRITE
    - Sets/gets the debug level at the FEC
  - SRVEXIT
    - WRITE only
    - must be enabled (!) e.g. “SetAllowRemoteManagement(TRUE)”
  - SRVIDLE
    - WRITE only
  - SRVINIT
    - WRITE only
  - SRVRESET
    - WRITE only
  - MESSAGE (write only) text appended to fec.log and on console (e.g. "address in use" from the ENS)
    - WRITE only

Scope: FEC

The Instant Client

- Stock Properties
  - MESSAGE
    - WRITE only
    - input: a text message
    - text appended to fec.log and on console
      - e.g. "address in use" from the ENS

Scope: FEC

The Instant Client

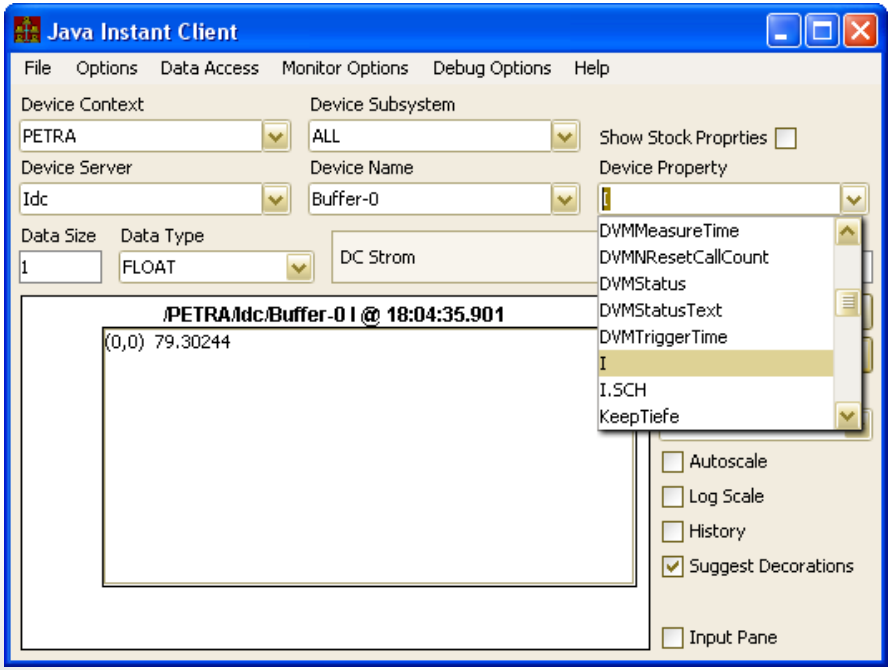
- Stock Properties (abandoned ?)
  - SRVSELFTEST

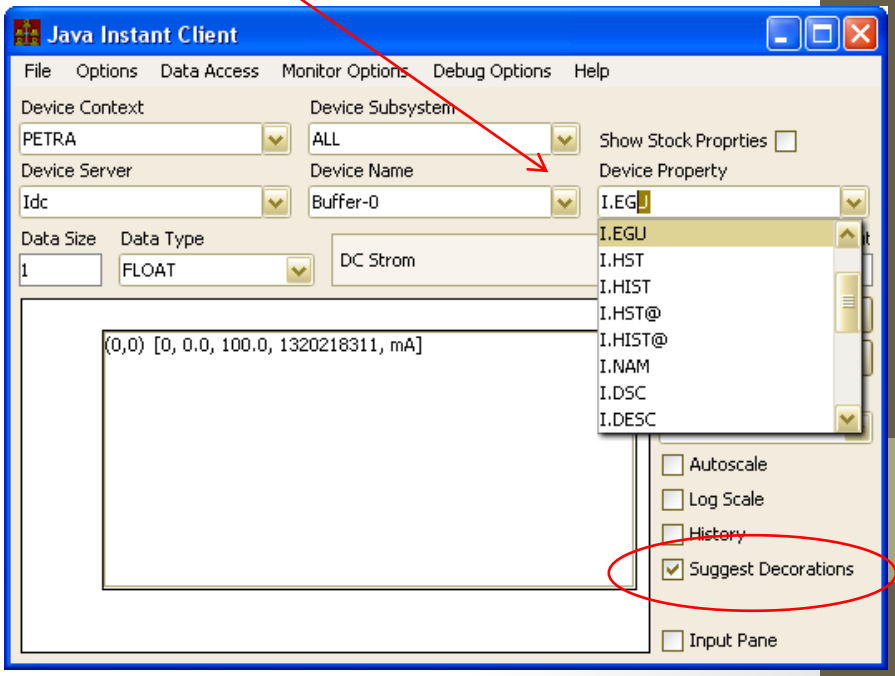
    Scope: Server
    - READ only
    - return: a self-test file with a list of properties/devices (with input) to access
      - each call must return ‘success’ in order to pass the test
  - APPDATE

    Scope: FEC
    - READ only
    - return the last application compile data
    - BUT: information must be provided via API (SetAppDate())
  - APPVERSION
    - READ only
    - return the last application version
    - BUT: information must be provided via API (SetAppVersion())

The Instant Client

- Meta Properties
  - “properties” giving information about other Properties !
  - hidden from general browsing !
  - displayed in java client via supplying a "." in the property combo box





The Instant Client

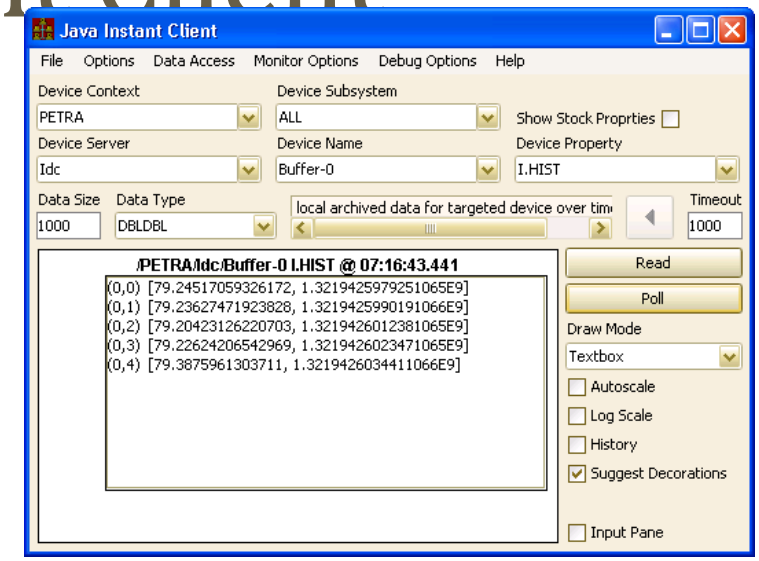
- Meta Properties
  - give a *'mechanism*' for obtaining something
  - do NOT imply 'success'
    - e.g. ".HIST" will return '*not allocated*' if no history stored !
  - .HIST, .ARCH
    - input as shown above (as in the central archive, etc.)
      - array of INT32 or doubles with 'start', 'stop' ('index', 'raster')
      - start, stop in UTC
        - so this is getting '*uncomfortable*' in the instant client!
    - output:
      - DBLDBL (what the viewer uses), or FLTINT, or INTFLTINT
      - CF_HISTORY *not available* in the instant client
  - .HIST@, .ARCH@
    - single input gives the target time.

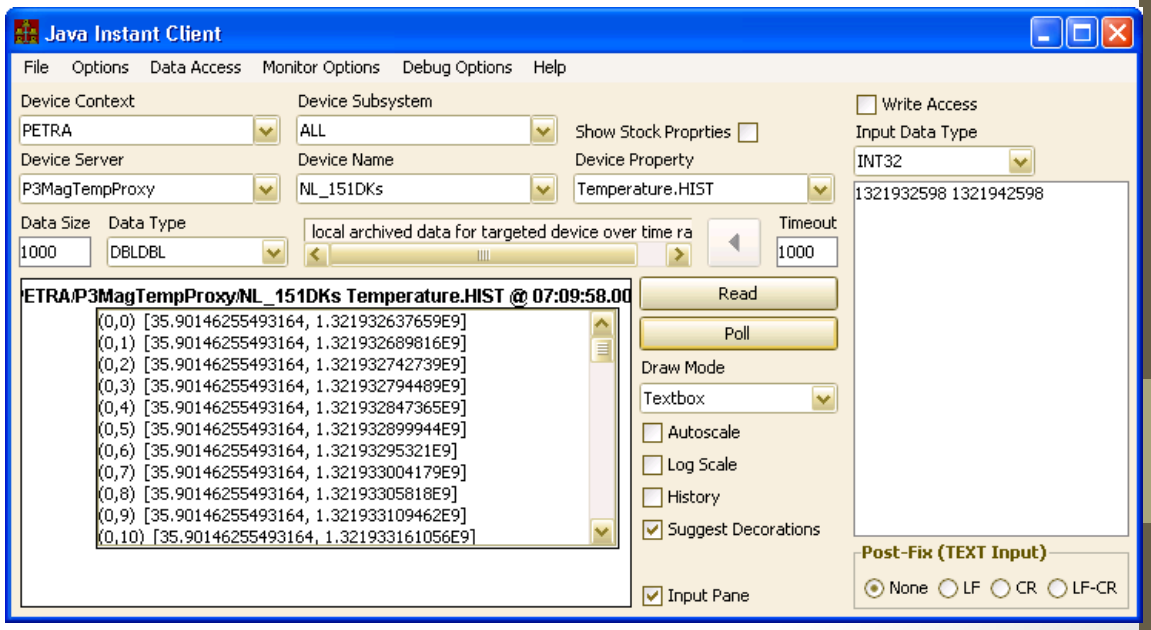
The Instant Client

Some examples:

no input :

start and stop UTC as INT32 :

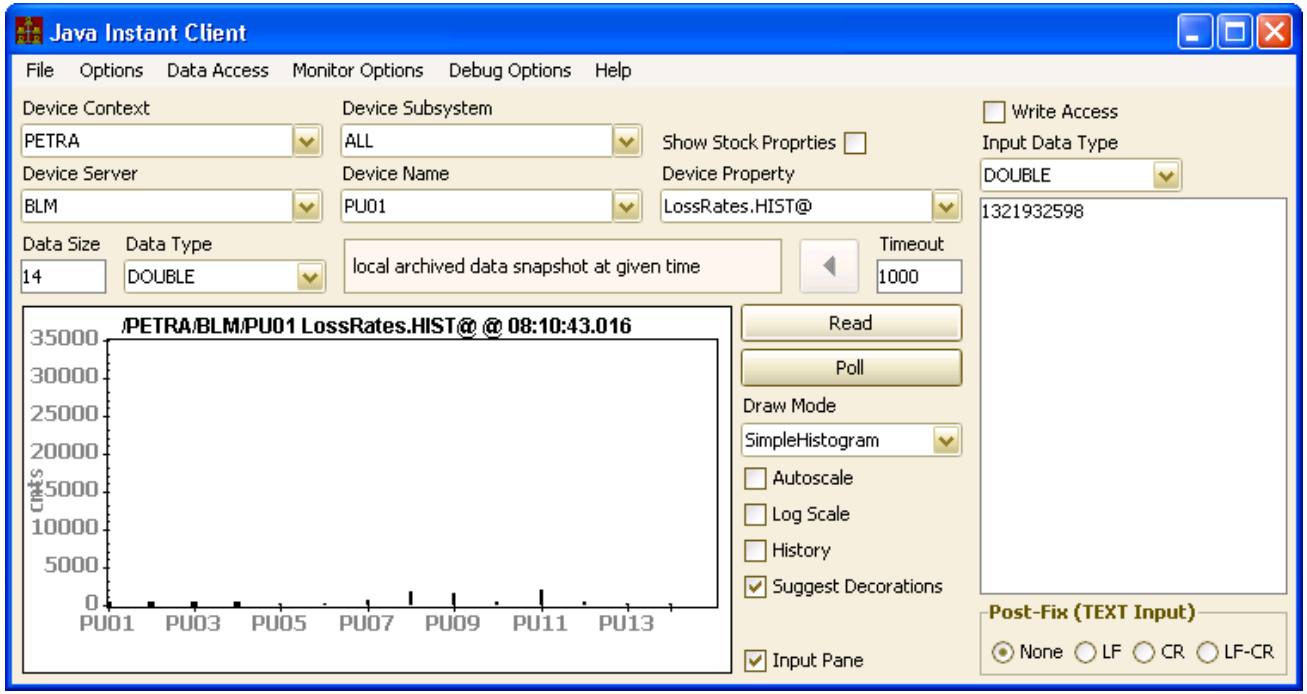




The Instant Client

- History snapshot:

input: target time (UTC)



- 'UTC' input makes this difficult to use with the Instant Client
  - (*better to use the archive APIs let the Archive Viewer handle this*)

The Instant Client

- Maybe add a ‘UTC generator’?
  - calendar + clock ?
  - ‘now’ – N hours ?
  - drag-and-drop to input panel ?

The Instant Client

- Limitations
  - Structure fields via '*suggest decorations*'
    - does not handle 'nested' structures !
    - if a field is itself another structure, it stops 'suggesting'
  - Structure input not possible
    - could provide an input *form* with fields etc. ?
    - could simply try to parse the input according to the known structure type ?