February 16, 1995

To: Stan Hilinski
From: Alex Measday
Subject: C'est Moi!

Since you asked about Tcl, I thought I'd let you know what I've done with it. I've got two Tcl-based applications, one running on the Master Controller card under VxWorks and the other running on a workstation under UNIX:

moi (MEDS Operator Interface) is the user interface program you see running on my workstation sometimes, displaying event messages, status blocks, etc., received from the LZP system. **moi** is programmed in Tk, a superset of Tcl that includes support for X Windows and GUI objects.

gentle (GENeric TcL sErver) is a network server program that provides each network client with its own Tcl interpreter. **gentle**'s Tcl has been extended with networking commands, memory access commands, and various MEDS commands (e.g., *getAlias*, *logEvent*, *mapRam*, *waitInit*, and a whole slew of the TPCE-LZP commands).

moi and **gentle** communicate using ASCII strings (sent as XDR strings in XDR network records). For example, the event logging loop can be tested by:

(1) Pressing **moi**'s "Ping Events" button to send a "logEvent *code message*" command to **gentle**.
(2) **gentle** calls the MEDS *eventmsg()* function to log the message.
(3) The MEDS event server then sends (via my **mbc** program) the event message back to **moi**.
(4) **moi** adds the incoming message to its scrolling display of event messages.

To display a particular status variable on the screen, **moi**:

(1) Sends a "peek *variable*" command to **gentle**.
(2) **gentle** retrieves the value of the specified variable, formats it in ASCII, and sends it back to **moi**.
(3) **moi** reads the returned value and displays it on the screen.

The above is a simplification of the exchange. The actual command sent to **gentle** is "*region* Get *variable*", where *region* specifies the MEDS status block (e.g., "APSTATUS") and *variable* is the name

of the field within the block. **moi** and **gentle** both make use of an object-oriented extension of Tcl: **[incr Tcl]** (the Tcl equivalent of C++). **[incr Tcl]** provides support for C++-like classes. In the command above, *region* is a memory region object that responds to a "Get" message by returning the value of the requested variable. *region* also responds to a "Set" message; for example:

```
$xpstatus Set numRejectedFrames 100000000
```

raises the XP's rejected frame count to 100,000,000.

A Quick Introduction to Tcl

Tool Command Language (Tcl) is a simple, extensible, command language whose interpreter can be embedded in any application. "#"s indicate comment lines. The basic command syntax is:

```
command argument(s)
```

Arguments are frequently specified using UNIX-style "-" options; e.g.,

```
Activate catalog -version N
```

Curly braces and double quotes are used to enclose strings. Square brackets enclose nested commands. The following command:

```
set xpstatus [Region #auto XPSTATUS -mapFunction mapSTS]
```

evaluates the "Region" command (which creates a region object and maps to the XP status block) and stores the resulting value (a region handle) in variable "xpstatus".

The value of a variable is referenced by prefixing a dollar sign to the variable name. In the following command:

```
$xpstatus Get gsHealthMsg
```

"$xpstatus" is replaced by the region handle returned above, so the actual command that will be executed is:

```
regionHandle Get gsHealthMsg
```

(This results in a "Get gsHealthMsg" command being sent to the region object.)

Tcl makes Perl look like a cross between COBOL and APL.

gentle - A Generic Tcl Server

As mentioned before, **gentle** is a network server that provides each client with its own Tcl interpreter. When **gentle** receives a command from a client, it passes the command to that client's

interpreter for execution. A "client write *message*" command allows the interpreter to send responses back to the client. **gentle** can also be run in *tty* mode (i.e., you type commands in at the command line) and on a UNIX workstation. For example, I can initiate a data session from the command line on my Sun:

```
% gentle -tty
set mcr [mcrOpen 3000@vxlzp1]        -- Connect to MC.
$mcr Activate DS8V2                  -- Activate catalog DS8V2.
set sds [mcrOpen 3000@128.183.92.97] -- Connect to Data Generator.
$sds Activate DS8V2                  -- Activate data generation.
exit
%
```

gentle adds a number of commands to the base Tcl command set. In the listing below, optional command options are not shown. For example, the

peek *location*

command supports the following options:

peek *?*-address *format? ?*-format *format?*
?-signed*? ?*-type *type? ?*-unsigned*? ?*-write*? location*

(Since square brackets have meaning in Tcl, Tcl documentation usually encloses optional arguments in question marks; e.g., "*?*-type *type?*".)

As another example, the full network *call* command supports these options:

call *server?*@*host? ?*-nowait*? ?*-connect *command?*
?-error *command? ?*-input *command?*

Networking Commands (UNIX and VxWorks)

call *server?*@*host?*
This command establishes a client connection with a network server.

listen *port*
listener answer
listener close
listener poll
These commands listen for and answer connection requests from clients.

connection close
connection poll
connection read
connection write *string*
These commands read and write information on a network connection.

Memory Access Commands (UNIX and VxWorks)

(A signal handler is used to trap bus errors and access violations.)

address -input *location*
address -offset *numBytes location*
address -delta *base location*
: These commands manipulate "%p" pointers (used in the other memory access commands).

dump *?*-count *numBytes? location*
: This command generates a formatted dump of a memory block.

fill *value start end*
: This command replicates a value throughout a memory block.

find *value start end*
: This command searches for a value in a memory block.

peek *location*
poke *location value*
: These commands read values from and store values in memory locations. Different data types are supported: signed and unsigned bytes, chars, shorts, ints, longs, floats, doubles, and strings.

Shared Memory Commands (UNIX and VxWorks)

map *name*
: This command maps a named, shared memory segment into the **gentle** process's address space. Naming is implemented under UNIX using an NDBM database and, under VxWorks, using the system symbol table. (Commands for mapping to MEDS memory regions are found further down.)

segment address
segment load *file*
segment save *file*
segment size
segment unmap
: These commands perform various operations on a mapped segment.

Miscellaneous Commands (UNIX and VxWorks)

client poll
client read
client write *message*
: These commands operate on the network connection (or TTY interface) to this interpreter's client.

ping
Pong!

timer *interval ?*-command *command? ?*-periodic*?*
timer cancel
This command creates a one-shot or cyclic timer that, each time it fires, executes a Tcl command.

tod
This command returns the current time-of-day.

MEDS Commands (UNIX and VxWorks)

mcrOpen *server?@host?*
This command establishes a network connection (hereafter called *mcr*) with a Master Controller.

mcr Close
mcr Connected
mcr Poll
These commands operate on a connection to a Master Controller.

mcr Activate *file*
mcr Distribute *source destination user?@host?*
mcr ListCatalogs *?wildcard_spec?*
mcr ListDataSets *?*-session *ID?*
mcr LoopBack *text*
mcr SendStatus *?interval?*
mcr Disable *?subsystem(s)?*
mcr Enable *?subsystem(s)?*
mcr Flush *?subsystem(s)?*
mcr NoOp *?subsystem(s)?*
mcr Reset *?subsystem(s)?*
mcr ShutDown *?subsystem(s)?*
mcr Test *?subsystem(s)?*
mcr Zero *?subsystem(s)?*
mcr TimeSpan *start length ?*-repeat*?*
These commands send the respective commands to the Master Controller and wait for responses.

MEDS Commands (VxWorks Only)

getAlias *name*
putAlias *name value ?*-global*?*
showAlias
These commands are used to access MEDS aliases.

logEvent *code message*
This command logs an event message.

mapGPA *?target?*
mapPCA *?target?*
mapSCB *?target?*
mapTCA *?target?*
mapRAM *name ?*-make *size?*
mapSTS *name ?*-make *size?*
These commands return the addresses of the various types of MEDS memory regions.

waitInit
It's a long wait, sometimes!

[incr Tcl] Classes (UNIX and VxWorks)

The region class:

```
class Region {
     member mapFunction       -- Default is "mapRAM".
     constructor()
     destructor()
     method Fill()            -- Defines fill space.
     method Define()          -- Defines typed field.
     method Get()             -- Gets value of field.
     method GetInfo()         -- Indexed access to arbitrary info.
     method IndexedAddress()  -- Computes address into array field.
     method Set()             -- Stores value in field.
     method SetInfo()         -- Indexed access to arbitrary info.
}
```

provides a concise means of mapping to and defining the layout of a MEDS memory region. For example, the following definition of the XP status region is found in "/folks/hilinski/lzp/etc/defineXP.tcl":

```
#    Map to XP status block.
set xpstatus [Region #auto XPSTATUS -mapFunction mapSTS]
#    MEDS status header.
DefineStatusHeader $xpstatus
#    XP information.
$xpstatus Define inProgress {byte -unsigned}
$xpstatus Define dataDirection char
$xpstatus Fill byte 6
$xpstatus Define sessionID    -- Default data type is ULONGWORD.
#    Frame counts.
$xpstatus Define numInputFrames
$xpstatus Define numRejectedFrames
$xpstatus Define ...
```

A specific field in the status block can be retrieved:

```
client write [$xpstatus Get numInputFrames]
client write [$xpstatus Get gsHealthMsg]
```

or modified:

```
$xpstatus Set numInputFrames 1234
$xpstatus Set gsHealthMsg "Never been faster!"
```

The "Info" methods are used to store arbitrary information. For example, to access particular entries in the XP's Virtual Channel Status table, you need to know the size of an individual entry. "defineXP.tcl" computes and saves this information as follows:

```
$xpvcs SetInfo sizeOfVCS \
       [expr [$xpvcs GetInfo nextOffset] - $baseOffset]]
```

This information can be retrieved and passed to the "Get" and "Set" methods in order to access particular entries in the table.

moi - The MEDS Operator Interface

moi is a generic, Tcl/Tk-based, graphical user interface. Tk is a Motif-like, X Windows toolkit that contains an embedded Tcl interpreter; this allows user interfaces to be programmed entirely in Tcl - no C coding required. **moi** adds a couple of Tcl extensions: the **gentle** networking commands (e.g., *call*, *listen*, etc.) and the **gentle** *timer* command.

moi initially reads a Tcl script file that defines the user interface. My file, "/folks/alexm/source/moi/lzp", connects **moi** to the **gentle** server and to the MEDS event server, and then creates the main window of my LZP interface; this window contains:

- a menu bar,
- a scrolling list of event messages,
- an operator input line, and
- several items of information from the segment directory.

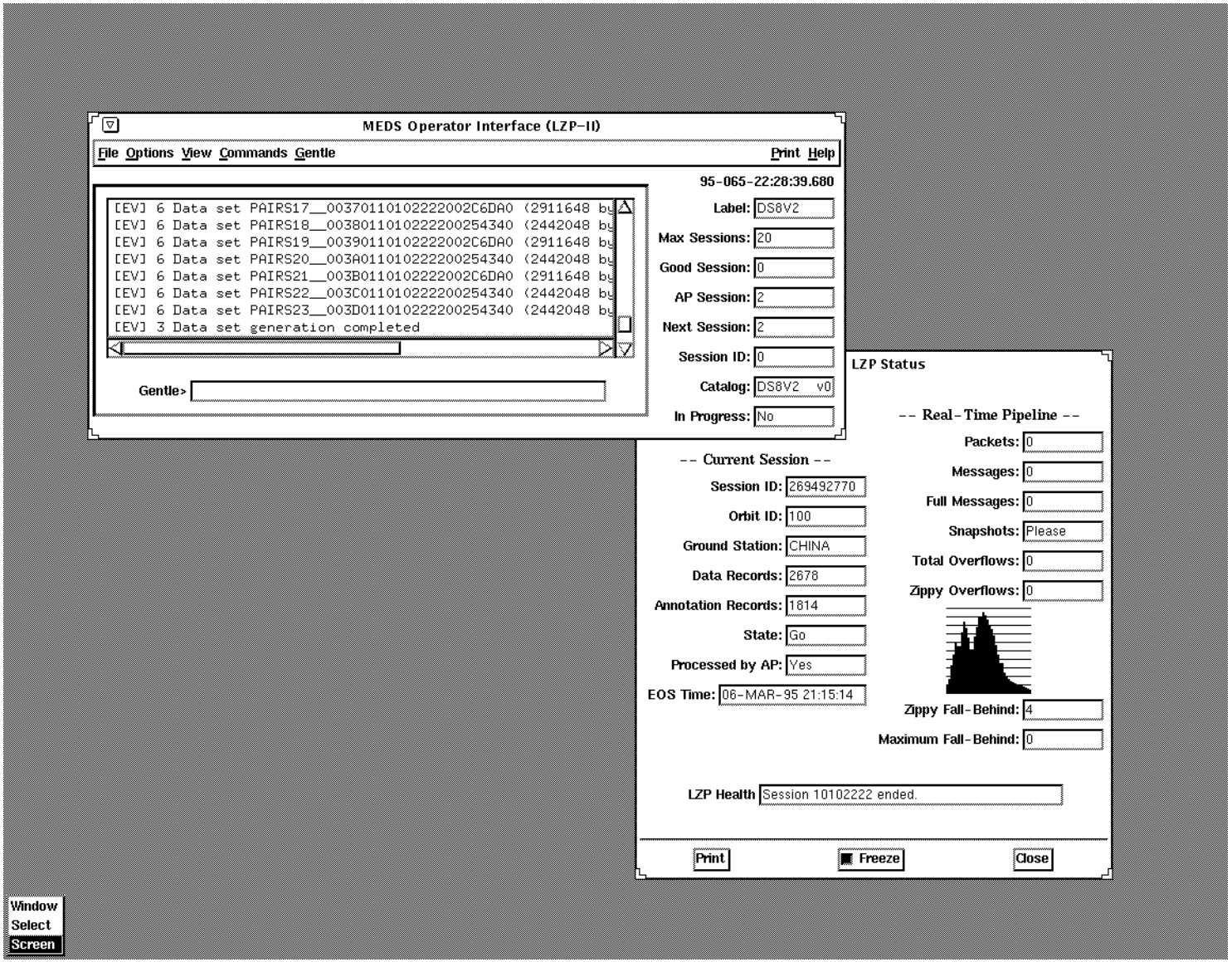
The menu bar has pull-down entries for:

(1) Viewing Pages (e.g., the various subsystem status pages and the segment directory).
(2) Commanding the LZP Rack.
(3) Commanding the Spacecraft Data Simulator.

[incr Tcl] Classes

[incr Tcl] was used to great advantage in my "lzp" application. I defined the following classes:

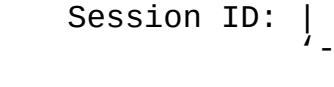
```
Box                     -- For grouping items on a page.
Button
     CheckButton
Dialog
     EntryDialog
Field
```


MEDS Operator Interface (LZP-II)
File Options View Commands Gentle
Print Help
[EV] 6 Data set PAIRS17__00370110102222002C6DA0 (2911648 by
[EV] 6 Data set PAIRS18__00380110102222002543 40 (2442048 by
[EV] 6 Data set PAIRS19__00390110102222002C6DA0 (2911648 by
[EV] 6 Data set PAIRS20__003A0110102222002543 40 (2442048 by
[EV] 6 Data set PAIRS21__003B0110102222002C6DA0 (2911648 by
[EV] 6 Data set PAIRS22__003C0110102222002543 40 (2442048 by
[EV] 6 Data set PAIRS23__003D0110102222002543 40 (2442048 by
[EV] 3 Data set generation completed
Gentle>
95-065-22:28:39.680
Label: DS8V2
Max Sessions: 20
Good Session: 0
AP Session: 2
Next Session: 2
Session ID: 0
Catalog: DS8V2 v0
In Progress: No
LZP Status
-- Real-Time Pipeline --
Packets: 0
Messages: 0
Full Messages: 0
Snapshots: Please
Total Overflows: 0
Zippy Overflows: 0
Zippy Fall-Behind: 4
Maximum Fall-Behind: 0
-- Current Session --
Session ID: 269492770
Orbit ID: 100
Ground Station: CHINA
Data Records: 2678
Annotation Records: 1814
State: Go
Processed by AP: Yes
EOS Time: 06-MAR-95 21:15:14
LZP Health Session 10102222 ended.
Print
Freeze
Close
Window
Select
Screen

```
Object1D
     BarGraph
     Dial
     Stripchart
Page
     DisplayPage         -- Periodically updates.
     MemoryPage          -- For memory dumps.
ScrolledList
Separator
Timer
Variable                 -- Local variables.
     RemoteVariable      -- Remote variables (on the LZP).
```

Variable objects are used to get/set local/remote variables. In the case of a RemoteVariable, a "*region* Get *variable*" command is sent to the **gentle** server on the LZP system; the **gentle** server "peek"s the value and sends its back to **moi**.

The display objects (Field, BarGraph, Dial, and Stripchart) each contain a Variable object, either local or remote. When you invoke the "Update" method of a display object, it "Get"s the value of its variable and updates its display. The most commonly used display object, Field, consists of a label and a display/entry field:

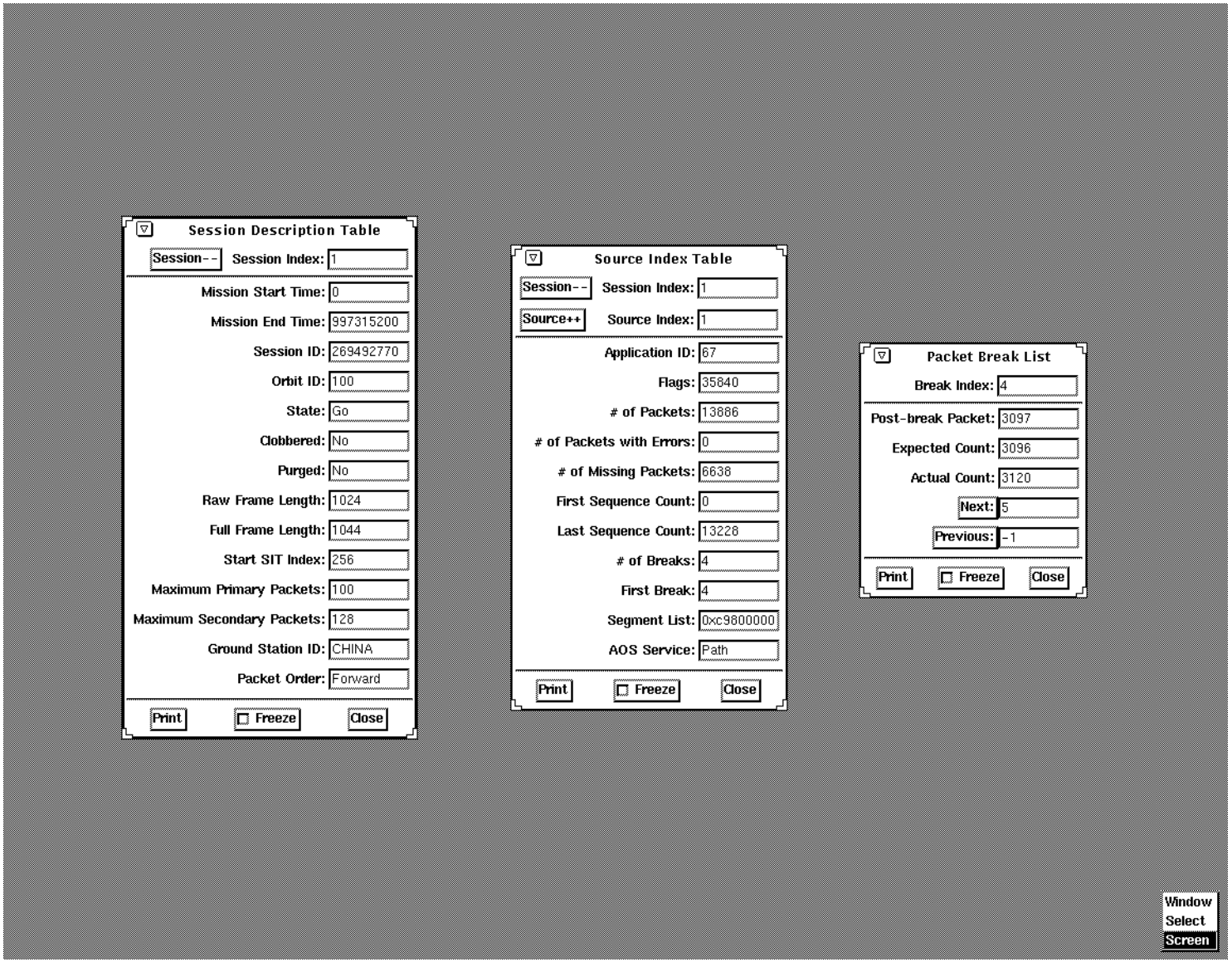


When a Field is "Update"d, it retrieves the value of its variable and places it in the display/entry field. If you type in a new value in the display/entry field, the new value is stored in the variable (e.g., over on the LZP rack). A Field's label can also function as a button. I use this capability in the Packet Break List page: clicking on the "Next Break" label advances the page to the next break entry.

Box objects are used to group display objects. For example, a two-column display is created by placing two boxes side by side. Invoking "Update" on a box results in "Update" being applied to each display object in the box.

Page objects provide a higher-level grouping of display objects. The bare-bones Page class is used for interactive pages; e.g., the LZP and SDS command panels. The DisplayPage class contains a timer that periodically invokes "Update" on the display objects (or boxes) in the page. Display pages are automatically given three buttons at the bottom of the window:

- *Print* - spools a PostScript dump of the page to the printer,
- *Freeze* - suspends/resumes updates, and
- *Close* - closes the page.


Session Description Table
Session--
Session Index: 1
Mission Start Time: 0
Mission End Time: 997315200
Session ID: 269492770
Orbit ID: 100
State: Go
Clobbered: No
Purged: No
Raw Frame Length: 1024
Full Frame Length: 1044
Start SIT Index: 256
Maximum Primary Packets: 100
Maximum Secondary Packets: 128
Ground Station ID: CHINA
Packet Order: Forward
Print
Freeze
Close
Source Index Table
Session--
Session Index: 1
Source++
Source Index: 1
Application ID: 67
Flags: 35840
# of Packets: 13886
# of Packets with Errors: 0
# of Missing Packets: 6638
First Sequence Count: 0
Last Sequence Count: 13228
# of Breaks: 4
First Break: 4
Segment List: 0xc9800000
AOS Service: Path
Print
Freeze
Close
Packet Break List
Break Index: 4
Post-break Packet: 3097
Expected Count: 3096
Actual Count: 3120
Next: 5
Previous: -1
Print
Freeze
Close
Window
Select
Screen

Memory pages display memory dumps from the LZP rack in a scrolling window.

The ScrolledList class provides a generic scrolling list capability. Scrolled lists are used:

(1) For the event message/command history display on **moi**'s main window.
(2) To display catalog lists and data set lists on the LZP and SDS command panels.
(3) To display memory dumps on a memory page.

Individual items in a list can be selected; this capability is used in (1) to recall previous commands and in (2) to select a catalog to activate or a data set to distribute.

Display Pages

The Page class is defined in **[incr Tcl]** as:

```
class Page {
     member displayWindow
     member infoArray
     member objectsArray
     member title
     constructor()
     destructor()
     method AddObj()
     method DeleteObj()
     method GetInfo()
     method GetObj()
     method Iterate()
     method Print()
     method SetInfo()
}
```

objectsArray is a list of the display objects on the page; *infoArray* is used to store and lookup arbitrary information.

The DisplayPage class adds several new methods to the base Page class:

```
class DisplayPage {
     inherit Page
     member updateInterval
     member isFrozen
     constructor()  -- Creates Print, Freeze, Close buttons.
     destructor()
     method Freeze()-- Suspends/resumes updating.
     method Update()-- Updates each object in window.
}
```

The "Update" method "Iterate"s through the *objectsArray*, invoking the "Update" method of each display object in the array. The

"Update" method in an individual object retrieves its variable's value (e.g., from the **gentle** server on the LZP rack) and updates its display.

Creating a display page is a simple matter. The following example is taken from the script file that creates the XP status page (see "/folks/alexm/local/lib/tcl/lzp/xpstatus.tcl"):

(1) Create a blank page:

```
set page [DisplayPage #auto -title "XP Status"]
set window [$page Window]
```

(2) Add display objects (e.g., Fields, ScrolledLists, Stripcharts) to the page:

```
CenteredText [$window] "-- XP Status --"
$page AddObj inProgress \
             [Field #auto $window "Packet Output:" [...]]
$page AddObj sessionID \
             [Field #auto $window "Session ID:" [...]]
... and so on ...
```

"[...]" specifies what variable is bound to the Field and is omitted for the sake of clarity. When the page is created, a periodic timer is started; whenever the timer fires (e.g., every 5 seconds), the page updates.

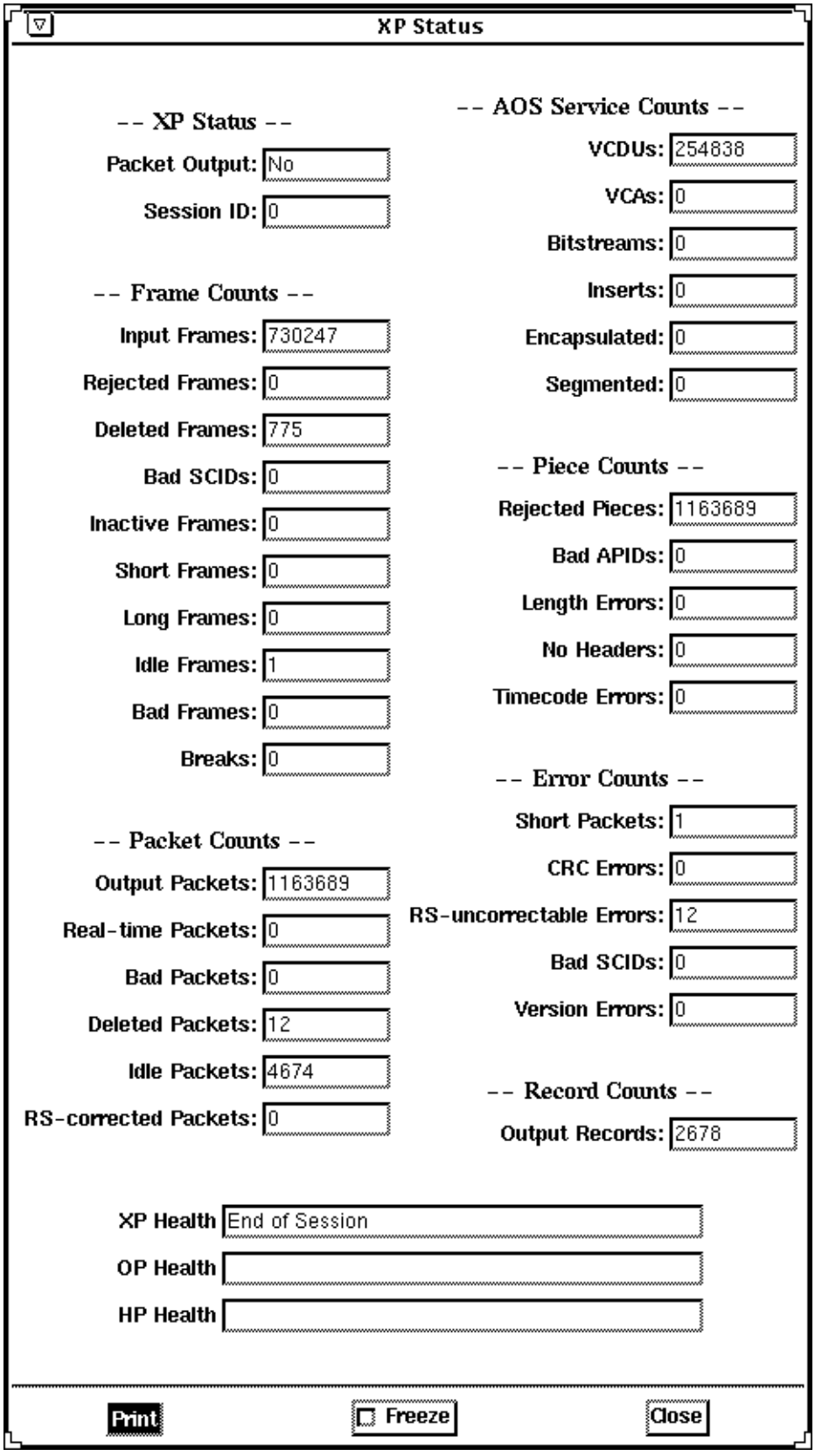
moi currently supports the following display pages:

Subsystem Status Pages -
- MC Status
- LZP Status
- HF Status
- RS Status
- XP Status
- VC Status (XP)
- Source Status (XP)
- RE Status
- AP Status
- DP Status

Segment Directory Pages -
- Segment Directory Header (SDH)
- Session Description Table (SDT)
- Spacecraft Sensor List (SSL)
- Source Reference Table (SRT)
- Source Index Table (SIT)

Packet Break List (PBL)
AP Segment Ordered List (SOL)

Because everything is coded in Tcl and **[incr Tcl]**, you can add a new subsystem status page in 15 minutes or less:


XP Status
-- XP Status --
Packet Output: No
Session ID: 0
-- Frame Counts --
Input Frames: 730247
Rejected Frames: 0
Deleted Frames: 775
Bad SCIDs: 0
Inactive Frames: 0
Short Frames: 0
Long Frames: 0
Idle Frames: 1
Bad Frames: 0
Breaks: 0
-- Packet Counts --
Output Packets: 1163689
Real-time Packets: 0
Bad Packets: 0
Deleted Packets: 12
Idle Packets: 4674
RS-corrected Packets: 0
-- AOS Service Counts --
VCDUs: 254838
VCAs: 0
Bitstreams: 0
Inserts: 0
Encapsulated: 0
Segmented: 0
-- Piece Counts --
Rejected Pieces: 1163689
Bad APIDs: 0
Length Errors: 0
No Headers: 0
Timecode Errors: 0
-- Error Counts --
Short Packets: 1
CRC Errors: 0
RS-uncorrectable Errors: 12
Bad SCIDs: 0
Version Errors: 0
-- Record Counts --
Output Records: 2678
XP Health End of Session
OP Health
HP Health
Print
Freeze
Close

(1) Create the status block definition script (e.g., "defineXP.tcl") that maps to and defines the layout of the status block.

(2) Create the subsystem status display page (e.g., "pageXP.tcl").

(3) Add a menu entry for the new page to the LZP script ("lzp").

Modifying an existing page is just as easy and quick. Although I currently don't do this, such changes could be made while **moi** is still running.

MEDS Command Panels

The MEDS command panels are used to send commands to the LZP and Spacecraft Data Simulator (SDS) racks. (Or to any rack, if desired.) Across the bottom of a command panel are two buttons:

Connect - connects and disconnects from the target Master Controller.
Cancel - exits the command panel; the MC connection, if up, stays up.

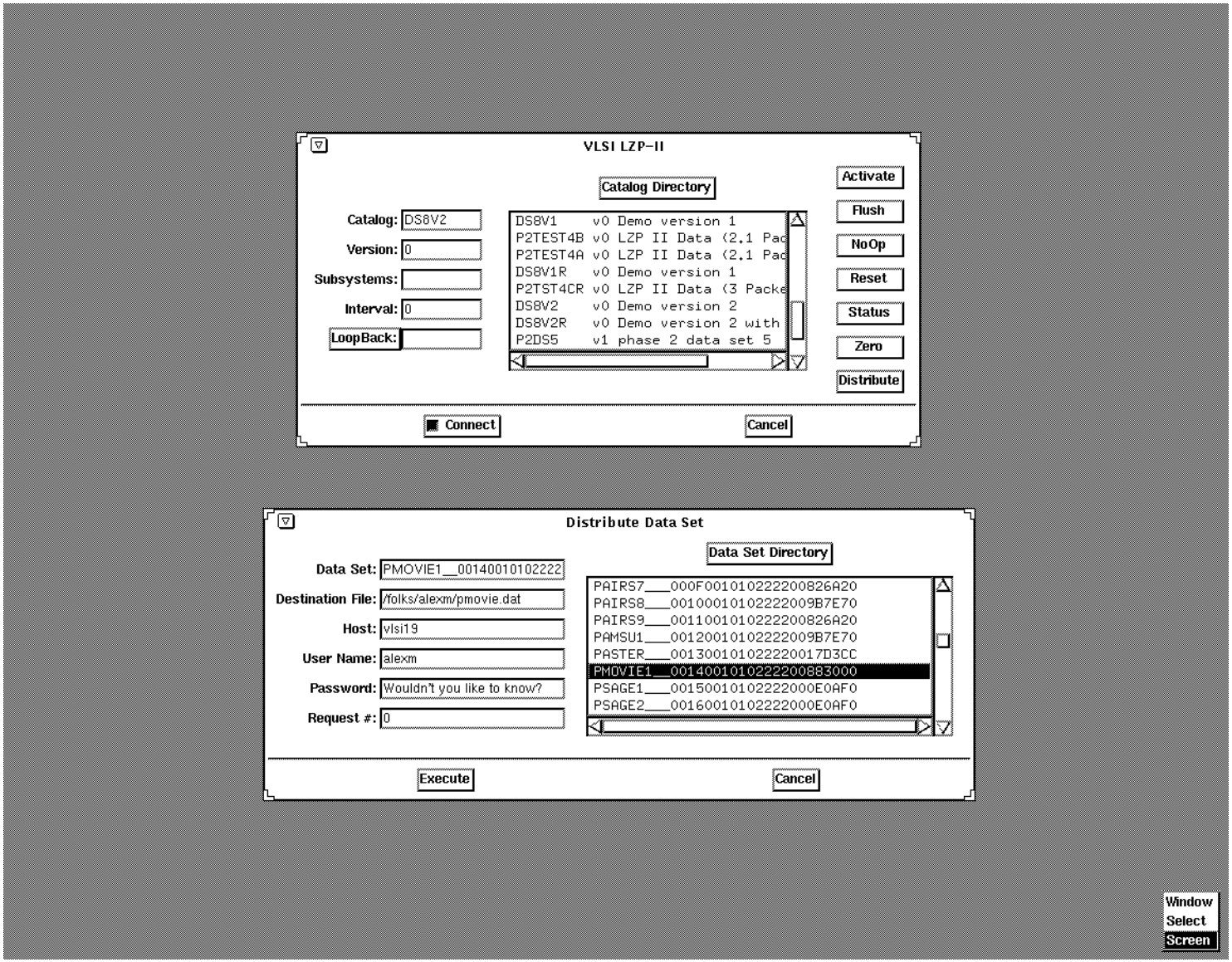
Down the left side of the command panel is a column of Fields for:

(1) Catalog name
(2) Version number
(3) Subsystems (for commands directed to subsystems)
(4) Interval (for status updates)
(5) Loopback text

In the center of the command panel is a scrolling list of the available catalogs; clicking on a catalog moves its name to the catalog name Field. On the right side of the command panel is a column of buttons:

Activate - activates the catalog whose name is found in the catalog name Field.
Flush - flushes the current session.
NoOp - sends a NoOp to the target subsystem(s).
Reset - resets the target subsystem(s).
Status - initiates the sending of status blocks.
Zero - zeros the target subsystem(s).
Distribute - invokes the data distribution panel.

The *Distribute* button brings up the data distribution panel (useless for the SDS). This panel is similar to the command panel: the available data sets are displayed in a scrolling list and the Field entries on the left side are used for entering the information needed for FTP transfers. As with the command panel, clicking on a data set moves its name into the data set name Field.


VLSI LZP-II
Catalog Directory
Activate
Flush
NoOp
Reset
Status
Zero
Distribute
Catalog: DS8V2
Version: 0
Subsystems:
Interval: 0
LoopBack:
DS8V1 v0 Demo version 1
P2TEST4B v0 LZP II Data (2,1 Pac
P2TEST4A v0 LZP II Data (2,1 Pac
DS8V1R v0 Demo version 1
P2TST4CR v0 LZP II Data (3 Packe
DS8V2 v0 Demo version 2
DS8V2R v0 Demo version 2 with
P2DS5 v1 phase 2 data set 5
Connect
Cancel
Distribute Data Set
Data Set Directory
Data Set: PMOVIE1__001400101022222
Destination File: /folks/alexm/pmovie.dat
Host: vlsi19
User Name: alexm
Password: Wouldn't you like to know?
Request #: 0
PAIRS7___000F001010222200826A20
PAIRS8___0010001010222009B7E70
PAIRS9___0011001010222200826A20
PAMSU1___0012001010222009B7E70
PASTER___0013001010222017D3CC
PMOVIE1__001400101022220088300O
PSAGE1___0015001010222000E0AF0
PSAGE2___0016001010222000E0AF0
Execute
Cancel
Window
Select
Screen