Created: October 23, 2001
Modified: January 29, 2002

Gemini Telescope Control System Report

An interface for Visitor Instruments to the Gemini TCS

Chris Mayer

tcs_cjm_069.fm/13

This document sets out the requirements and the design for an interface to the Gemini TCS for visitor instruments.

1.0 Introduction

The Gemini Control System is designed as a distributed network of cooperating subsystems [1] where each subsystem is based around a VME crate running VxWorks and EPICS. The communication protocol between these crates is Channel Access [2]. An instrument that is built around this same architecture and implements the protocols set out in a series of ICDs is known as a conforming Gemini instrument. It has long been recognised that not all instruments mounted on the Gemini telescopes will be “conforming”. In particular, some instruments will have been originally built for other telescopes or will be mounted on other telescopes in the future. These instruments are known as visitor instruments and this note describes the method by which such instruments can be integrated into the Gemini Control System.

1.1 Intended audience

This document is aimed at

- Gemini Software and Controls Manager
- Gemini Software Support Staff
- Head of Gemini Instrumentation
- Visitor Instrument Development Groups

1.2 Scope

This document only addresses the software aspects of interfacing a visitor instrument to Gemini. Mechanical and electrical interfacing are described elsewhere. Note also that only control and status information is described here, data handling is dealt with separately.

1.3 Definitions, Acronyms and Abbreviations

EPICS	Experimental Physics and Industrial Control System
TCS	Telescope Control System

Command completion - a command is considered complete when it has been received, verified and acknowledged.

Action completion - as a result of a valid command, actions are started within the TCS. These actions are considered complete when all TCS mechanisms reach their demanded positions

1.4 References

[1] SPE-C-G0037 - *Software Design Description*, Gemini 8m Telescopes Project

[2] *The Channel Access Reference Manual*, Jeff Hill, LANL

[3] tcs_pbt_001.fm - *ICD 3.1/1.1.11 OCS to TCS*, Chris Mayer, Philip Taylor and Dave Terrett

[4] ocs.015-OcsWishUsers - *The Ocswish User's Manual*, Shane Walker and Kim Gillies

[5] *Practical Programming in Tcl and Tk*, Brent B. Welch

[6] gemProcedure.fm - *Installation of new releases of the Gemini Real-Time Systems*, Andy Foster

1.5 History

V7.0 - Extend description of how to start server. Update in light of experience with OSCIR. Add section on installation.

V8.0 - Add commands to do x, y offsets and status items instrpa, nodtype, targetframe, targetradecsys and targetepoch. Released with V2-1 of the software.

V9.0 - Extended section on nodding plus extra commands mountGuide and m2GuideConfig and status items. Released with V2-2 of the software.

V10.0 - Add sections on accessing NIRI parameters. Remove early version comments from this section.

V11.0 - Add commands to control probes. Add probe angle parameters. Add interface to handset commands and description of different offset types plus control of M1 corrections.

V12.1 - Released with V2-8 of the software. Removed earlier history and expanded descriptions of offsets. Added description of Java demo

V13.0 - Add commands cleartargetoffsets, pwfs1observe, pwfs1stop etc.

2.0 User Requirements

The following requirements for the visitor instrument interface have been gleaned from various meetings and e-mail exchanges

2.1 General Requirements

UR0005 The interface must support visitor instruments that run on a variety of operating systems and are written in a variety of languages

UR0010 The interface must be capable of operating in a distributed networked environment.

UR0015 The interface must allow the visitor instrument to command the telescope and its sub-systems as well as request status

UR0020 The interface must send acknowledgements when commands are received to show whether they have been accepted or rejected.

UR0025 The interface must send a message to the instrument when the actions initiated by a command have been completed

UR0030 It must be possible to configure the interface so that command acknowledgements and/or action completion messages can be disabled.

UR0035 It must be possible to configure the interface to accept or reject commands from specified locations

UR0040 It should be possible to monitor a status item in the TCS such that updates are sent to the instrument whenever the item changes. This relieves the instrument of polling.

UR0045 Potentially all public commands and status as described in the TCS/OCS ICD [3] must be available through the interface.

UR0050 The visitor instrument will be the master in the system and the Gemini TCS will be the slave

UR0055 The interface must support connections from multiple instruments concurrently

UR0060 The number of concurrent connections to the interface should be configurable

UR0065 A means must be available to test the interface at the instruments home institute prior to its arrival at Gemini

UR0070 Example client code should be provided to illustrate how to use the interface to connect an instrument to the TCS.

2.2 Constraint Requirements

UR01000 There is no requirement that the interface operate over RS232 lines

UR01010 If two commands are sent to the TCS such that the action started by the first has not completed before the next one arrives then there will be only one action completion message when both complete.

2.3 Specific Requirements

The requirements here are those that have been specifically mentioned as capabilities that are needed from day 1. There is no intention to limit the interface to only the commands and status mentioned here but simply that due to time constraints these will be implemented first.

UR02000 The status items listed in Section 5.0 must be available via the interface

UR02005 The commands listed in Section 6.0 must be available via the interface

3.0 Implementation

Given the requirements listed in Section 2.0 we have chosen to implement the interface as a server process that accepts ascii strings via a socket connection. The advantages of this approach are

1. Gemini does not have to distribute EPICS channel access software nor support it on different platforms nor make it callable from different languages
2. All commands and status pass through a single point making it relatively easy to implement security and safety policies
3. Visitor instrument groups need learn nothing about EPICS and the Gemini specific protocols that are used to communicate between Gemini systems.

The obvious major requirement imposed on the developer of a visitor instrument is that they must be able to open a socket connection from their instrument control system to

the visitor instrument server. To take full advantage of the server it is desirable but not essential that their control system is multi-threaded. They then need not block whilst waiting for an action to complete and can make use of the monitor facility to keep status up to date without polling.

The server is implemented using ocswish [4]. Ocswish is a Tcl/Tk shell that already has built in the ability to talk Channel Access using the Gemini protocols. Although not a requirement at present, it also has the ability to communicate with “services” other than EPICS. Using ocswish for the server will allow visitor instruments to control other non-EPICS systems should the need arise in the future.

3.1 Security

Running a server on the Gemini network that listens on a socket and then executes commands that affect the telescope is both a safety and security risk. Although the underlying Channel Access protocols also rely on sockets, the protocols are much more complex than those proposed here so a hacker would require much more specialised knowledge to do any damage. The following measures will be adopted to guard against unauthorised use.

1. The prime defence against unauthorised use will be the Gemini firewall. This blocks most access from off site.
2. The server will implement a safe interpreter [5]. Only the commands listed in Section 4.0 and Section 6.0 will therefore be executed by the server. All other commands will be rejected.
3. Only a limited number of connections to the server will be allowed at a time. Once that limit is reached attempts by other clients to connect will be rejected.

3.2 Safety

A decision has to be made as to whether to allow or disallow slewing the telescope through the visitor instrument interface. At present it is intended to implement a command to allow slewing but to leave it disabled. If the command is issued it will be rejected with a suitable error message.

4.0 Proposed protocol

In the following sections command strings are given in lower case or occasionally in a combination of upper and lower case. Note that the server itself is case insensitive so that it is up to the instrument what case it uses to send the strings.

The case of responses from the server are exactly as given here and all responses are terminated by a <CR><LF>. Status item and command names will be returned all in lower case.

4.1 Establishing the connection

The details of how to connect to the server socket will depend on what language is being used to write the instrument control system. No matter how it is done, the server will

write a message back down the socket indicating whether the connection is accepted or rejected. After establishing the connection the instrument should therefore read this initial message and take the appropriate action. Possible messages are

```
Connect: Ok
```

or

```
Connect: Busy
```

If the status is Ok then the server is ready to accept commands. If the status is “Busy” then there are already more connections to the server from other instruments than it can handle and the instrument should try again later. Once the server has sent the “Busy” message it will close the socket descriptor at its end.

4.2 Dropping the connection

Once an instrument is finished with the connection to the server then it should simply close the socket descriptor it is using.

4.3 Fetching status

Status items can be retrieved from the server with the get command. The syntax is

```
get <status-item>
```

The server will respond with

```
got <status-item> <value>
```

The complete list of status items is given in Section 5.0

Two conditions need to be checked for when the “got” message is received

1. If the status item is not known to the server then value will be set to “Unknown”. This could come about from mis-spelling a status name for example.
2. If the TCS is not running at the time the get command is received by the server then value will be set to “Unavailable”.

4.4 Monitoring status

Rather than periodically fetching status items from the server, the instrument control system can monitor the item so that it is sent asynchronously whenever it changes. The syntax is

```
monitor <status-item>
```

and the server will send a message of the following format initially and each time the value changes

```
mon <status-item> <value>
```

The initial value is so that an up to date value is obtained immediately even if the monitored item changes very rarely. Monitoring can be switched off with the command

```
monitorOff <status-item>
```

As with the get command, if the status item is unavailable then the value returned is "Unavailable" and if it is unknown then the value will be "Unknown". In this latter case of course there will never be any further monitor messages but if the initial response is "Unavailable" then monitors will start being sent as soon as the TCS starts running i.e. it is not necessary to issue further monitor commands when the TCS is started.

Issuing a monitor command when a monitor is already active has no affect i.e. you only get one update when the status item changes value not two.

4.5 Issuing commands

Commands are issued in the form

```
do <command> <keyword>=<value> <keyword>=<value>.....
```

The full list of commands can be found in Section 6.0 . Parameters to the command are given as keyword value pairs separated by an "=" sign. The keywords can be given in any order and there can be arbitrary numbers of spaces between the tokens and around the equals sign. If the <value> itself contains spaces e.g. a Right ascension like 10 30 24.2 there is no need to enclose the value in quotes. If you do however they will be stripped off before sending the value to the TCS. You should not separate the tokens by tabs.

It is generally not necessary to specify every parameter to a command each time it is issued but only those that have changed since the last invocation; however see Section 7.3.3 for important caveats.

The response to a command is

```
ack <command> <val> <message>
```

If val is < 0 then the command has been rejected and the reason is given in the message field.

If val >= 0 then the command has been accepted and the message field will be "Ok"

If the command has been accepted then when the action initiated by the command completes a further message will be sent. This completion message indicates that all actions in the TCS have been completed. The format is

```
done <val> <message>
```

Again, if the val field is < 0 then an error has occurred and the reason is given in the message field. If the value is >= 0 then the message field is "Ok"

4.6 Turning on/off acknowledgements

Under some circumstances it may be advantageous to turn off the acknowledgement and completion messages from the server that are generated by a `do` command. If this is the case, it is up to the operator to manually verify that commands have been accepted and that the telescope is in the configuration requested. The command to do this is

```
disable < ack | done >
```

To re-enable acknowledgements, use the enable command.

```
enable < ack | done >
```

Neither of these commands generate any responses back to the originator

5.0 Status items

The table below lists all the status items that can be fetched from the TCS with the get or monitor commands. Note at this stage all status items are atomic. It is very likely that blocks of status items built out of these atomic ones will also prove useful. Any items shown shaded are not yet available

Status item	Units	Description
absorbtiptilt	0 | 1	True (= 1) if off loading mean tip/tilt to mount
Airmass	None	Relative airmass, 1 at the zenith
azError	Sexagesimal string	Error in azimuth position (degs:mins:secs)
Azimuth	Sexagesimal string	Current azimuth of the telescope (degs:mins:secs)
baseX	mm	Base pointing origin in focal plane
baseY	mm	Base pointing origin in focal plane
beam	A | B | C	Current nod beam
chopBeam	A | B | C	Current chop beam (only updates at 1 Hz)
chopDutyCycle	%	Secondary duty cycle
chopfreq	Hz	Chop frequency
choppa	degrees	Chop position angle
chopping	Yes | No	Whether currently chopping or not
chopthrow	arcseconds	Chop throw (arcseconds on sky)
Elevation	Sexagesimal string	Current elevation of telescope (degs:mins:secs)
elError	Sexagesimal string	Error in elevation position (degs:mins:secs)
focus	mm	Absolute focus position
framePA	String	Frame of reference of instrPA
guiding	0 | 1	True (= 1) if tip/tilt guiding with M2

TABLE 1. Individual status items available from the server

Status item	Units	Description
HA	hours	Hour Angle
Health	String	GOOD | WARNING | BAD
Humidity	%	Percentage relative humidity
instrAA	degrees	Instrument alignment angle
instrPA	degrees	Instrument position angle
localTime	String	Local time formatted as hh:mm:ss.s
LST	String	Local Sidereal Time formatted as hh:mm:ss.s
MJD	days	Modified Julian Date on UTC timescale
nodmode[a]	standard | offset	Current nod mode
nodpa[a]	degrees	Nod position angle
nodthrow[a]	arcseconds	Nod throw
nodtype[a]	radec | azel | tangent plane	Nod type
offsetDec	arcseconds	Declination offset from base position
offsetRA	arcseconds	RA offset from base position
offsetX	mm	Offset from base pointing origin
offsetY	mm	Offset from base pointing origin
p1ArmAngle	degrees	P1 probe arm angle
p1TableAngle	degrees	P1 rotary table position
p2ArmAngle	degrees	P2 probe arm angle
p2TableAngle	degrees	P2 rotary table position
programID	String	Program identifier as set by the OCS
pwfs1X	mm	Current probe x coordinate in focal plane
pwfs1Y	mm	Current probe y coordinate in focal plane
pwfs2X	mm	Current probe x coordinate in focal plane
pwfs2Y	mm	Current probe y coordinate in focal plane
pwfs1Xdemand	mm	TCS demanded probe x coordinate in focal plane
pwfs1Ydemand	mm	TCS demanded probe y coordinate in focal plane
pwfs2Xdemand	mm	TCS demanded probe x coordinate in focal plane
pwfs2Ydemand	mm	TCS demanded probe y coordinate in focal plane
rotator	degrees	Rotator mechanical angle
rotError	degrees	Error in rotator position
targetName	String	Name of target object
targetRA	degrees	Base RA of target
targetDec	degrees	Base declination of target

TABLE 1. Individual status items available from the server

Status item	Units	Description
targetEpoch	years	Epoch of target position
targetEquinox	years	Equinox of target
targetFrame	FK4 | FK5 | APPT | AZEL_TOPO	Frame of coordinate system
targetRaDecSys	FK4 | FK5 | GAPPT	Conventional FITS coordinate frame description
Telescope	String	Name of telescope (Gemini-North or Gemini-South)
telRA	Sexagesimal string	Current RA of telescope
telDec	sexagesimal string	Current Declination of telescope
userFocus	mm	Offset from nominal focus
UTC	String	Universal Coordinated Time in format hh:mm:ss.s
utcdate	String	UTC date in format yyyy-mm-dd
zd	degrees	Current zenith distance

TABLE 1. Individual status items available from the server

a. These status items are maintained by the server not the TCS

5.1 NIRI status items

Following the installation of NIRI on the telescope it was found convenient to access a range of status items via the vis for the purposes of updating the headers of the data files. The status items available are listed below

Status item	Units	Description
niriFilter1	String	Name of filter in wheel 1
niriFilter2	String	Name of filter in wheel 2
niriPupilMask	String	Pupil or filter name for wheel 3
niriFocalPlaneMask	String	Contents of focal plane mask wheel
niriPupilViewer	String	Pupil viewer position name
niriBeamSplitter	String	Beam splitter
niriSteerMirrors	String	Steering mirror positions
niriDetFocus	mm	Detector focus position
niriWindowCover	String	Window cover
niriHealth	String	Health of system
niriTopStrapTemp	degree K	Top strap temperature
niriBotStrapTemp	degree K	Bottom strap temperature

TABLE 2. NIRI status items available from the server

Status item	Units	Description
niriCryoPumpTemp	degree K	Cryo pump temperature
niriEdgeTemp	degree K	Cold plate edge temperature
niriOiwfsTemp	degree K	OIWFS temperature
niriImageBuffTemp	degree K	Image buffer mass temperature
niriColdPlateTemp	degree K	Cold plate temperature
niriIntegTime	seconds	Integration time
niriProcessMode	String	Processing mode
niriCoadds	Integer	Number of coadds
niriLowNoiseReads	Integer	Number of low noise reads
niriDigitalAvs	Integer	Number of digital averages
niriDataPath	String	Path to location of data files
niriFileName	String	Current data file name
niriDataLabel	Integer	Data label
niriDetHealth	String	GOOD | WARNING | BAD
niriBiasVoltage	volts	Bias voltage
niriDetTemp	degree K	Detector temperature
niriWfsFilter	String	WFS filter name
niriWfsFocus	mm	WFS focus position
niriWfsXpos	mm	WFS probe x position
niriWfsYpos	mm	WFS probe y position
niriWfsXskypos	arcsec	WFS probe x sky position
niriWfsYskypos	arcsec	WFS probe y sky position

TABLE 2. NIRI status items available from the server

6.0 Commands

The table below lists all the commands that can be issued to the server to control the telescope

Command	keyword/value pairs	Description
absorbadjust	No parameters	Absorb adjustment offsets into target position
absorboffset	No parameters	Absorb handset offsets into target position
chop	state =<On | Off>	Turn chopping on or off

TABLE 3. Commands accepted by the server

Command	keyword/value pairs	Description
chopBeam	beam = <A | B | C>	Move to selected chop beam
chopConfig	sync =<SCS | OSCIR>	System that controls the synchronisation of chop
	freq = value	Chop frequency (Hz)
	throw = value	Chop throw (arcsec)
	PA = value	Chop PA (degs)
	frame =<FK5 | FK4 |AZEL>	Chop Frame
	Equinox = value	Chop Equinox e.g. J2000.0, B1950.0
clearadjust	No parameters	Clear any adjustment offsets
clearoffset	No parameters	Clear any handset offsets
cleartargetoffsets	vt = pwfs1 | pwfs2 | oiwfs | mount | source	Clear all offsets for the specified target
focus	offset = value	Change M2 position by specified no. of mm
guide	state =<On | Off>	Close the guide loop with M2
handset	Type =<radec | azel | tangent plane>	Type of offset
	dRA = value	Incremental offset in RA/Az (arcsec)
	dDec = value	Incremental offset in Dec/El (arcsec)
handsetAbsorb	No parameters	Absorb handset offsets into target position
handsetClear	No parameters	Clear any handset offsets
m1Corrections	state = <On | Off>	Turn M1 figure corrections on or off
m1GuideConfig	source = <PWFS1 | PWFS2 | OIWFS>	Set source of M1 figure updates
m2GuideConfig	source = <PWFS1 | PWFS2 | OIWFS>	Source of guide signals
	chopbeam = <A | B | A+B>	Chop beam where guide star will appear
mountGuide	state = <On | Off>	Enable off loading of M2 errors to mount
nod	beam = <A | B>	Nod the telescope to the selected beam
nodConfig	mode = <standard | offset>	Use standard for nodding along the beam
	type = <radec | azel | tangent plane>	Type of nod. Only necessary if mode is offset
	throw = value	Throw (arcsec) Only necessary if mode is offset.
	pa = value	PA (degs) Only necessary if mode is offset
offset	Type =<radec | azel | tangent plane>	Type of offset
	dRA = value	Offset in RA or Azimuth (arcsec)
		N.B. dRA is in secs for type=radec
	dDec = value	Offset in Dec or Elevation (arcsec)
offsetAdjust	off1 = value	RA, x or Az offset depending on frame (arcsec)
	off2 = value	Dec, y or El offset depending on frame (arcsec)
	frame = <tracking | acqcamera | azel | instrumentxy>	Frame of offset

TABLE 3. Commands accepted by the server

Command	keyword/value pairs	Description
oiwfsobserve	numexp = value	Number of exposures (-1 = continuous)
	exposure = value	Exposure time (secs)
oiwfsstop	No parameters	Stop oiwfs integrating
p1Follow	state = <On | Off>	Turn P1 probe following on or off
p1GuideConfig	nodachopa = <on | off>	Guide state for nod A chop A
	nodachopb = <on | off>	Guide state for nod A chop B
	nodbchopa = <on | off>	Guide state for nod B chop A
	nodbchopb = <on | off>	Guide state for nod B chop B
p2Follow	state = <On | Off>	Turn P2 probe following on or off
p2GuideConfig	nodachopa = <on | off>	Guide state for nod A chop A
	nodachopb = <on | off>	Guide state for nod A chop B
	nodbchopa = <on | off>	Guide state for nod B chop A
	nodbchopb = <on | off>	Guide state for nod B chop B
pwfs1observe	numexp = value	Number of exposures (-1 = continuous)
	exposure = value	Exposure time (secs)
pwfs1source	name = string	Name of target object
	frame = <FK4 | FK5 | AZEL>	Target coordinate frame
	RA = value	Right Ascension
	Dec = value	Declination
	Equinox = value	Target equinox e.g. J2000.0, B1950.0
	Epoch = value	Target epoch e.g. J2000.0, B1950.0
	parallax = value	Parallax (arcsec)
	pmRA = value	Proper motion in RA (sec/yr.)
	pmDec = value	Proper motion in Dec (arcsec/yr.)
	rv = value	Radial velocity (km/s)
pwfs1stop	No parameters	Stop pwfs1 integrating
pwfs2observe	numexp = value	Number of exposures (-1 = continuous)
	exposure = value	Exposure time (secs)

TABLE 3. Commands accepted by the server

Command	keyword/value pairs	Description
pwfs2source	name = string	Name of target object
	frame = <FK4 | FK5 | AZEL>	Target coordinate frame
	RA = value	Right Ascension
	Dec = value	Declination
	Equinox = value	Target equinox e.g. J2000.0, B1950.0
	Epoch = value	Target epoch e.g. J2000.0, B1950.0
	parallax = value	Parallax (arcsec)
	pmRA = value	Proper motion in RA (sec/yr.)
	pmDec = value	Proper motion in Dec (arcsec/yr.)
	rv = value	Radial velocity (km/s)
pwfs2stop	No parameters	Stop pwfs2 integrating
rotator	PA = value	Position angle (degs. east from north)
	frame =<FK5 | FK4 | AZEL>	Rotator frame
	Equinox = value	Equinox of frame e.g. J2000.0
	iaa = value	Instrument alignment angle (degs)
source [a]	name = string	Name of target object
	frame = <FK4 | FK5 | AZEL>	Target coordinate frame
	RA = value	Right Ascension
	Dec = value	Declination
	Equinox = value	Target equinox e.g. J2000.0, B1950.0
	Epoch = value	Target epoch e.g. J2000.0, B1950.0
	parallax = value	Parallax (arcsec)
	pmRA = value	Proper motion in RA (sec/yr.)
	pmDec = value	Proper motion in Dec (arcsec/yr.)
	rv = value	Radial velocity (km/s)
targetAdjust	frame = <tracking | acqcamera | azel | instrumentxy>	Frame of offset
	off1 = value	RA, x or Az offset depending on frame (arcsec)
	off2 = value	Dec, y or El offset depending on frame (arcsec)
	vt = <Mount | SourceA | SourceB | SourceC | PWFS1 | PWFS2 | OIWFS>	Name of virtual telescope to apply offsets to
wfsguide	state = <On | Off>	Turn guiding on or off
	probe = <PWFS1 | PWFS2>	Probe to guide
xyOffset	dx = value	X offset from base pointing origin (mm)
	dy = value	Y offset from base pointing origin (mm)

TABLE 3. Commands accepted by the server

Command	keyword/value pairs	Description
xyOffsetClear	No parameters	Clear user x, y offsets
xyOffsetAbsorb	No parameters	Absorb user x, y offsets into base pointing origin

TABLE 3. Commands accepted by the server

a. Only available through the simulation server

7.0 User guide

7.1 Connection to “simulated” TCS

This section describes the protocol to connect to the Visitor Instrument Interface simulation server. For security reasons, access to the simulation server will only be allowed to currently active developers of Gemini visitor instruments. See Section 7.2 for access to the real server to be used after a visitor instrument has arrived at Gemini.

To gain access to the Visitor Instrument Interface simulation server, contact Jim Wright (`jwright@gemini.edu`) to coordinate access through the Gemini firewall.

The machine running the server is vii-sim.hi.gemini.edu; the port number is 7283. Prior to connecting an instrument, the interface should be tested using the telnet command to connect to the above machine and port and then issuing a command from Section 4.3 or Section 4.4 . This is illustrated below:

```
% telnet vii-sim.hi.gemini.edu 7283
  Trying 128.171.188.202...
  Connected to vii-sim.hi.gemini.edu.
  Escape character is '^]'.
  Connect: Ok
```

If you see the message “Connect: Ok” then the connection was successfully established to the server.

If you see the message “telnet: Unable to connect to remote host: Connection refused” or something similar, this indicates that the Visitor Instrument Interface simulation server is not running.

If you wait a long time and eventually see the message “telnet: Unable to connect to remote host: Connection timed out” or something similar, this indicates that your connection attempt is being discarded by the Gemini firewall.

If you see the message “Connect: Busy” then the server already has connections from the maximum number of instruments it is configured to handle.

If you find yourself in any of these error situations then contact one of the Gemini software support staff.

Assuming you have connected successfully then type:

```
get telescope
```

The response should be:

```
got telescope Gemini-North
```

If you receive “got telescope Unavailable” then the server is running but the telescope control system (simulator) is not running. In this case get an SSA to start the TCS simulator for you.

To disconnect from the server simply drop your telnet connection. For the Unix telnet command, type `CTRL-]` and at the “telnet>” prompt type “`quit`”.

7.2 Connection to “real” TCS

Once on-site, visitor instruments will have access to the Visitor Instrument Interface connected to the real TCS. The procedures are identical to those outlined above, with the following exceptions.

1. The host name is vii.hi.gemini.edu.
2. Access to vii.hi.gemini.edu is only allowed from within the Gemini firewall

7.3 Guidelines on writing client applications

The following sections provide some guidelines on the events that a visitor instrument will need to handle when it connects and sends commands to the server process.

7.3.1 Connection/disconnection events

The first requirement is to open a socket connection to the server both for reading and writing. How this is done will depend very much on the language used to write the instrument control system and so will not be described here. Section 8.0 provides some example clients in a number of different languages that may help.

Having created the socket, then the application should immediately read the Connect message that is sent by the server (Section 4.1). The server only permits a limited number of simultaneous connections and when this limit is reached it will reject any further connections. If the message returned is “Connect: Ok” then your connection has been accepted. If it is “Connect: Busy” then your connection has been rejected. In this latter case the message will immediately be followed by an end of file (EOF) indicating the socket had been closed by the server.

Applications should close the socket at their end if they read an EOF. This can happen in the circumstances described above and if the server crashes or is shut down.

Although not essential it is a good idea to make the connection and disconnection to the server an explicit command within the instrument control system rather than part of the startup process. This will make it much easier to recover in the event that there is a problem with the server

7.3.2 Fetching status

Status may be fetched in two ways, either synchronously via the get command or asynchronously via the monitor command (see Section 4.3 and Section 4.4). In either case, there are a number of points to watch out for when the command is first issued

1. The command `get` or `monitor` is misspelled. This is only likely to be a problem if the control system provides a means for the user to interactively send commands.

For example if the command sent is "`got utc`" rather than "`get utc`" then rather than the expected response you will get

Error: invalid command name "got"

The server only understands a very limited number of commands. In general if at any time you give it a command it doesn't recognise or an internal error is generated you will get back a message prefixed by "Error:"

2. The parameter specified doesn't exist. This may also come about through a misspelling. The only parameters available from the TCS are those in Section 5.0 . If you send a command to fetch something else you will get

   got <param> Unknown

3. The TCS isn't running. In this case the server can't fetch the status item asked for and so it returns

   got <param> Unavailable

   The instrument program should be able to handle the string "Unavailable" as well as the normal response sent when the TCS is running.

   Note that if the command was `monitor` then when the TCS is booted and starts running updates will be sent automatically. There is no need to issue another `monitor` command

7.3.3 Issuing commands to the TCS

Commands to the TCS are executed by sending a `do` command to the server. The available commands are given in Section 6.0 . By default the instrument will receive two responses to a `do` command. The first response is an acknowledgement. If the value received with the acknowledgement is 0 then the command has been accepted. If it is less than 0 then it has been rejected and a message string will give the reason.

Commands can be rejected for the following reasons

1. The instrument isn't permitted to send commands. The server can be configured such that it will accept more connections for status than for commands. When the maximum number of command connections is reached any client that tries to issue commands will get back the message "Commands not permitted"
2. The command name is not recognised. Only the commands in Section 6.0 are permitted
3. One of the parameter names was not recognised.
4. The syntax of the command is incorrect. Command syntax is fairly free e.g. arbitrary amounts of white space, parameters in any order etc. but forgetting the "=" sign for example between keywords and values will confuse the parser.
5. The command was rejected by the TCS. For example a perfectly valid command to the TCS will be rejected if there is an interlock present.
6. The command is currently disabled

If the command is accepted then some time later a completion message will be received. If the value associated with the completion message is 0 then the actions started by the command have all completed successfully. If the value is less than 0 then an error has occurred and a reason will be given in the message field.

If a client instrument does not want to handle the completion messages from the commands e.g. if it is single threaded and doesn't want to block whilst the command is being executed then the completion message can be switched off with the command `disable done`. Of course in this case the instrument must use some other means of determining when it is safe to continue e.g. waiting a pre-measured interval of time. It is possible to also turn off the acknowledgement messages but this is not recommended as there is then no way of telling whether the command has been accepted or not.

Although keyword values are in general persistent you should not rely on this for the correct operation of your client application. For example, if you issue the command

```
do chopConfig sync=OSCIR freq=3.0 throw=20.0 frame=FK5
```

and later decide to change the chop throw to 22.0 then you could simply issue

```
do chopConfig throw=22.0
```

This will work provided that your application is the only source of commands. Note that it is not good enough that your application is the only source of commands through the server. This is because the persistence arises through the underlying TCS. If for example after the first command the chop frequency is modified through the TCS engineering screens then the second command will still work it is just that the frequency might be different from the one you expect. If you are in any doubt then the safest course is always to send all the keyword/value pairs with the command.

7.4 The offset commands

Although the following sections discuss individual commands, an overview is given here of the various ways of offsetting the telescope as there are quite a number of different ways of doing this and the large number of different commands are potentially confusiong.

The commands available through the vii are in the end set by the facilities of the TCS. For any target the TCS keeps track of three separate offsets. Ultimately all three offsets are simply added together along with the base position to generate the total demand. The offsets are numbered 0, 1 and 2 and are reserved as follows

1. Reserved for use by the handsets
2. Reserved for use by the user
3. Reserved for use by the TCS during nodding

The various vii commands map to these offsets as follows:

offset 0 - targetAdjust, offsetAdjust, absorbAdjust and clearAdjust

offset 1 - handset, handsetAbsorb, handsetClear, offset, absorbOffset and clearOffset

offset 2 - nod

So, why are there still so many different commands to manipulate these offsets? This is partly historical and partly practical. The historical reason is that the TCS itself has changed over the lifetime of the vii but old commands have been retained so as not to affect visitor instruments that already make use of them. The practical reason is that sometimes it is convenient to have commands that work incrementally and sometimes it is convenient to have commands that work absolutely. Section 7.5 below provides details on each of the commands that affect the offsets.

7.4.1 Size of nods/offsets

There are some additional points to consider depending on the size and direction of any offset (or nod) when guide stars are being used. The problem arises if the guide probes can't reach the new offset position. Note that this can happen for quite small offsets if the guide probe is already very close to the edge of its travel and the offset is along the line between the target and guidestar but away from the guidestar.

If the probe can't physically reach the offset position then the only options are

1. Use a different guide star for the offset position
2. Leave the probe where it is and don't guide in the offset position

If the second option is chosen then prior to doing the offsets the following commands should be issued

```
do p1GuideConfig nodachopa=off
do p1Follow state = off
```

There are corresponding commands for p2. Once the offset has been cleared then probe following can be restarted with

```
do p1GuideConfig nodachopa = on
do p1Follow state = on
```

Note that two commands are used here. The first is an instruction to the TCS not to bother generating new demands and the second an instruction to the guide probes to ignore the stream of demands from the TCS. If you don't mind leaving the probe following then you can just use the first command. What the TCS does is continue to send demands but they are now fixed numbers being the last demands sent prior to receiving the p1GuideConfig command. The probe is left following but now follows a fixed constant demand.

7.5 Details of individual commands

7.5.1 absorbAdjust

Absorb any offsets set with the targetAdjust or offsetAdjust commands into the base target position

7.5.2 clearAdjust

Set any offsets set with targetAdjust or OffsetAdjust commands to 0.

7.5.3 cleartargetoffsets

This command is a more general form of the commands clearAdjust, clearOffset etc. It can be used to clear all the offsets of a particular target (see Section 7.4) including any offsets applied to guide stars.

7.5.4 handset

This command applies incremental offsets to the current position. Once an offset increment in RA and Dec has been set up, the same increments can be repeatedly applied simply by sending `do handset`. Be particularly careful here to ensure that you only apply an increment in the direction you want. Consider the following sequence

```
do handset type = tangent plane dRA = 1.0
do handset type = tangent plane dDec = 1.0
```

The second command will apply a 1 arcsec increment in *both* RA and Dec. The reason is that the RA increment is still set to 1.0 from the preceding command. To be sure you get the effect you want (particularly if the commands are executed from a handset type display) issue the commands as follows

```
do handset type = tangent plane dDec = 0.0 dRA = 1.0
do handset type = tangent plane dDec = 1.0 dRA = 0.0
```

If you want to send absolute offsets see Section 7.5.11 .

7.5.5 handsetAbsorb / absorbOffset

These commands have identical functions and simply absorb the current offsets into the base target position at the same time as resetting the offsets to 0. Note that they only absorb the offsets specified with the handset or offset commands. Offsets specified with the targetAdjust or offsetAdjust are not affected.

7.5.6 handsetClear / clearOffset

These commands also have identical functions. They reset the current offsets to 0 and once again only affect the offsets specified with the handset or offset commands.

7.5.7 m1Corrections

This command turns on and off the corrections from the WFS to the figure of M1. To set up which WFS to use, see the m1GuideConfig command below. Turning the corrections off and then on may be advisable during a nod or offset when the star is lost from the probe during the transition from one target position to another.

7.5.8 m1GuideConfig

Use this command to select which WFS is to be used to provide aO corrections to the PCS. For PWFS2 and OIWFS only astigmatism will be provided due to the low order of the Shack Hartmann array. For PWFS1 a much larger number of modes is available.

7.5.9 m2GuideConfig

This command configures the secondary control system to accept guide signals from a particular WFS in a particular chop beam. It is possible for M2 to receive guide signals from several sources for the same chop beam but it is more normal to assign a single WFS.

It may be necessary issue this command during a nod if the guide probe has been configured to follow the nod

7.5.10 mountGuide

Averaged over several seconds, the mean displacement of M2 due to the atmosphere when guiding is zero. Other effects e.g. tracking errors can cause a steady buildup of tilt on the mirror. If this tilt becomes large then it will start to affect the image quality. The mountGuide command can be used to off load any DC components of tilt from M2 to the mount so that the mean tilt of M2 remains zero. Mount guiding should be switched on after M2 guiding is enabled and switched off before M2 guiding is disabled.

7.5.11 nod / nodConfig

To cancel out telescope effects when chopping with an IR instrument the telescope usually nodded periodically in such a way that the target appears in the other chop beam. Nodding is achieved simply by issuing the nod command with beam parameter set to the beam in which you want your science target to appear. The telescope takes care of all the details of how to move the telescope to achieve this affect.

Although making the telescope perform the nod is easy it does introduce several complications due to the fact that the telescope is effectively moving to a different target. The first problem is how to determine when the nod is complete and the second is what to do about guiding.

At present the only solution to the problem of knowing when the telescope has completed the nod is to measure the time taken directly and then code this predetermined wait into the application that sent the command. The reason that the “done” signal can’t be used is that at the time of writing the mount does not correctly set this signal. This problem is being addressed but in the meantime the only safe method is to measure the time taken prior to setting up the integrations.

The solution to the guiding problem depends on how the guide target has been set up. It is assumed below that the most common method is used i.e. the guide probe has been set to follow chop state A in nod state A and chop state B in nod state B. This will result in essentially zero probe movement during the nod and the star will appear on the WFS during chop state A of nod A and chop state B of nod B. (see Section 7.5.12)

With the above setup then the sequence of operations to nod the telescope is

- open the guide loops
- configure M2 to look for the guide star in the other chop beam
- nod to the other beam and wait for completion
- close the guide loops

In the case of nodding from beam A to beam B and assuming we are guiding with PWFS2, the exact sequence of commands that would be sent would be:

```
do mountguide state=off
do guide state = off
do m2guideconfig source = pwfs2 chopbeam = b
```

```
do nod beam = b
<wait for completion>
do guide state = on
do mountguide state = on
```

The above sequence (and the one to nod back to beam A) is coded into the demo tcl program (see Section 8.1)

Nodding is usually done along the chop throw but it is possible to nod in arbitrary directions by using the nodConfig command. In this case you must specify the mode of the nod to be offset as well as giving the size, direction and coordinate frame of the nod. To switch back to nodding along the chop direction set the mode parameter back to standard.

7.5.12 offset

If the chosen frame type is radec then the RA is in seconds of time. For all other cases the RA offset is in arcsec.

Note that the offsets given are absolute. Issuing two offset commands in succession each with offsets of say 10 arcsec will result in a total offset of 10 arcsec not 20 arcsec. For incremental offsets use the handset command.

7.5.13 p1/p2GuideConfig

These commands are instructions to the TCS to tell it for which nod and chop states it should be generating demands for the guide probes. For example, if you want the probe to follow the nod i.e. for it to reposition itself back onto the guide star then you would issue the command

```
do p1GuideConfig nodachopa = on nodbchopa = on
```

Some combinations of parameters are not sensible e.g.

```
do p1GuideConfig nodachopa = on nodachopb = on
```

This would be an instruction for the probe to try and follow the chop. This would not be possible as the probes physically can't move fast enough to match the image motion. One combination that is very common is

```
do p1GuideConfig nodachopa = on nodbchopb = on
```

This is useful during "standard" nodding (see Section 7.5.10) in that the demands to the probe are the same for these two states so you can guide in the other nod state without having to move the probe.

7.5.14 pwfs1/pwfs2/oiwfsObserve and Stop

These commands are used to start and stop integrations of the wave front sensors. The observe commands can take two parameters which are the number of exposures and the integration time. Once these have been set for the first time then, provided you don't want to change these you can simply issue the command without parameters.

7.5.15 source

It is likely that for safety reasons this command will only be available in the TCS simulator. The minimum number of parameters to specify to make a valid command is the RA and Declination.

7.5.16 targetAdjust / offsetAdjust

These commands are very similar. The only difference is that targetAdjust has an extra parameter that allows you to specify which of the virtual telescopes within the TCS you want to apply the offset to. The offsetAdjust command is hard wired to always offset "source A" which is the most common requirement.

These commands differ from the offset and handset commands in that the selection of frames in which the offsets can be applied is more sophisticated. For example, if the frame "acqcamera" is selected and an offsetAdjust command sent then the offsets will be aligned with the acquisition camera axes. Similarly, if instrumentxy is chosen then the axes will align with the instrument principal direction. If this is the slit of a spectrograph then the target will offset along the slit.

These commands all work incrementally and are very closely tied the facilities of the TCC handsets.

7.5.17 xyoffset

This command changes the position of the target in the focal plane by the amounts specified. Note that like the offset command the parameters are absolute and represent the offsets from the base pointing origin. Thus, if you have offset the x position by 1 mm and you want to offset it by another 1 mm you would issue the command

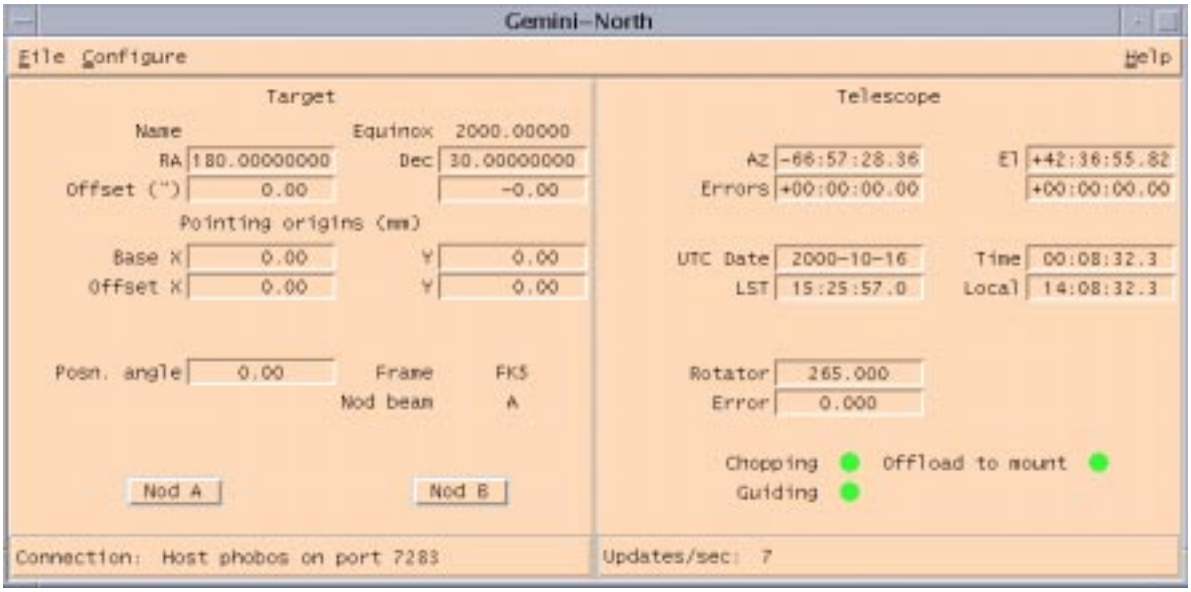
```
do xyoffset dx=2.0 dy=0.0
```

8.0 Example instrument clients

This section describes a number of example instrument clients that have been written to demonstrate how to interface to the Telescope Control System server. It is hoped that these examples will make it easier for instrument groups to interface their instrument to the Gemini system. Copies of these clients are freely available from the `vii/demos` directory of the Gemini CVS repository. Contact a member of the Gemini software support group if you would like a copy.

8.1 Tcl/Tk client

This client provides a display that shows the current status of the Gemini TCS. Parameters of the current target are shown on the left hand side of the window and parameters of the telescope on the right hand side. A screen shot is shown below



The buttons nod A and nod B can be used to nod the telescope to the relevant beams using the sequence of commands given in Section 7.5.10 . The configure option on the menu bar allows you to set the wait time for the nod to complete. The option is also provided of waiting for the done signal but this will currently only work reliably with the simulator.

By default the client is set up to connect to the simulation server in Hilo but menus and dialog boxes are provided from the file menu to allow connection to any machine running the server. The client can gracefully handle disconnect events when the server is shut down as well as situations where the server is busy.

To run the client simply start up wish and then at the prompt type

```
source <dir>/demo.tk
```

where `<dir>` is the path to wherever you have placed the file demo.tk

The file demo.tk does not use any tcl/Tk extensions and has been tested on a Linux system under versions tcl/Tk 7.6/4.1 and 8.0 as well as under Tk 8.2 on solaris.

8.2 C language client

This client makes a connection to the vii server and if successful fetches the telescope name and then exits. By default the program will try and connect to vii-sim.hi.gemini.edu on port 7283 but these can be overridden on the command line e.g.

```
./demo hobbit 8300
./demo
```

In the first example the program will try to connect to a host called hobbit on port 8300. In the second it will connect to vii-sim.hi.gemini.edu on the default port 7283.

8.3 Java clients

Two java clients have been written. The first is very simple and mimics a telnet connection. The second is an implementation of the Telescope Control Consoles target handset.

8.3.1 ViiDemo

This very simple client acts similarly to a telnet session. The application can be built by typing gmake in the directory into which you have placed the source code. Check the Makefile first to make sure that the variables JAVA_DIR, LINK_DIR and JAVA_BIN are appropriate for your installation. By default the class files will be placed in the directory `../classes` relative to your source directory. If you want to generate the documentation then you should also run `gmake docs` after compiling the application. The documentation is placed by default into `../html` relative to your source directory.

To run the client type

```
java -classpath ../classes ViiDemo <host> <port>
```

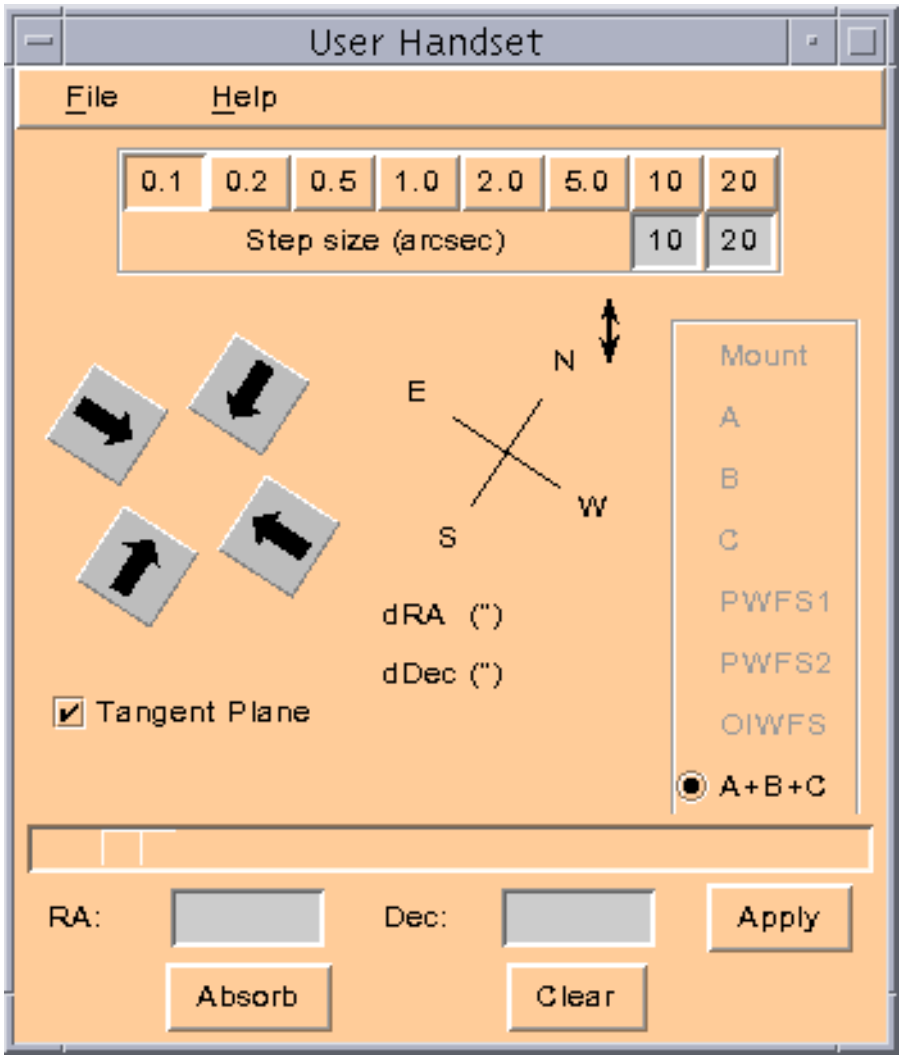
`<host>` and `<port>` are optional and default to vii-sim.hi.gemini.edu and 7283 respectively.

If the client connects successfully you'll see the "Connect: Ok" response. You can now type in any of the commands get, monitor, do etc. and the response will be printed to the terminal. To shut down the program simply type quit or exit.

8.3.2 Handset

The application tries to mimic as closely as possible the handset interface provided by the tcc screens (in fact the tcc handsets are now rather different but earlier versions looked very much like this Java application)

The handset interface is shown below



In order to connect to the vii server, use the connect button on the File drop down menu. A dialog box will pop up that allows the name and port used by the server to be entered. If the server refuses the connection or exits whilst the handset is connected a warning dialog will be displayed.

You can set the size of the offset sent to the telescope via the row of push buttons along the top of the display. Two of the buttons can be customized to set user specified offset sizes by typing into the text entry boxes below them.

Once an offset has been selected, it can be applied in the required direction by clicking the arrow buttons. A slider can be used (as has been done in the screen shot) to orient the buttons in line with the acquisition camera image if this is required. A small double

headed arrow symbol can be clicked to reverse the N-S direction and this, coupled with the rotation of the whole button set allows any orientation to be achieved.

Offsets are normally applied in the tangent plane but this can be changed by clicking the check box. Similarly, if a specified offset is required the numbers can simply be typed into the text entry boxes in the lower parts of the screen and the apply button pressed. Buttons are also provided to absorb the offsets into the target positions or to clear them down to zero.

On the right hand side of the handset there is a column of systems to which the offset will be applied. For this example all options except A + B + C are disabled.

The example is provided with a Makefile which may need editing to point a few definitions at your local Java environment. The example has only been tested on Java 1.2.2 but should work on later versions. It will not work on Java 1.x environments.

To build the application simply type `make` (or `gmake`). To generate the documentation type `gmake docs`.

The application can be run either by

```
gmake run
```

or

```
java -classpath ../classes Handset
```

9.0 The server processes

This section describes the server applications that implement the functionality and requirements of Section 2.0 to Section 4.0 . It is mainly aimed at the Gemini Support staff who must maintain and possibly extend the server at a later date.

The interface to the TCS is made up of two processes “vis” and “visControl” (the vis stands for **v**isitor **i**nstrument **s**erver). Vis implements the server used by visitor instruments and visControl controls and configures vis. Only vis needs to be running to provide the interface that an instrument uses to talk to the TCS.

The main reason for splitting the interface into two separate applications was that vis must run as a background task. There is then a question of what to do about its control terminal or graphical front end when you log out of the work station where it was started from. Rather than handle this complication it seemed easier to completely split the control aspect off into another process that is only run up when needed. This separate process is visControl.

Both vis and visControl are currently run under the Tcl/Tk extensions provided by the ocs [4]. Vis uses ocssh and visControl ocswish. Vis must use ocssh as it requires access to the epics service that extension provides. VisControl could be run under any Tk wish as it doesn’t make use of any extensions. It is simply convenient to use the same environment for both applications.

9.1 Starting the “real” server

Note that vis **must** be started on the machine to which the name vii.hi.gemini.edu is mapped. You can get to that machine with the command

```
% telnet vii.hi.gemini.edu -l gemvx
```

The machine vii.hi.gemini.edu is currently set to be lepus at Gemini North (there is currently no alias set up on Gemini South but the machine used is usually antu) Be sure to use the above command so that if the machine hosting the server is changed the rest of this procedure will work correctly. Vis can be started from the `/gemini/vii/vii` directory with the command

```
% ./vis &
```

The default port that the server opens up is 7283. If it is necessary to change the default port number then this can be done by changing the variable sockNo in `visMain.tcl` and restarting vis. The Gemini copy of the /etc/services file also notes the port assigned to vii, and should be changed if the port changes.

VisControl can be started from the same directory as vis with the command

```
% ./visControl &
```

Note that it is not necessary to start this on the machine vii.hi.gemini.edu. All its communications with vis are via a socket (the number allocated is one greater than that used

by the instruments to talk to the TCS). Potentially visControl could therefore run anywhere on the internet but in practice the machine must be within the Gemini firewall. There are no plans to open the port visControl uses to off site access.

Both vis and visControl require the environment variable HLPG_INSTALL_BASE to be defined. This variable is used to locate the ocssh and ocswish binaries. Typically it should be set to /gemini/ocs/ocs. You also need to define GEMINI_BASE which determines where the log file is written. It is typically set to /gemini. As with all channel access applications, the variable EPICS_CA_ADDR_LIST must to set to broadcast on the appropriate subnet. If you are connecting on the summit it should be set to 10.2.2.255.

9.2 Starting the “simulation” server

To run a server that connects to the TCS simulator in Hilo you must first log in to vii-sim.hi.gemini.edu. The server must be run on the machine to which the name vii-sim.hi.gemini.edu is mapped and this is the only name that off-site instruments can use to pass through the firewall. Set EPICS_CA_ADDR_LIST to 10.1.2.255 and define the environment variable VIS_TCSNAME to be tc1 (note there is no colon). Set the GEMINI_BASE and HLPG_INSTALL_BASE variables as above. You can then start the server as described in the previous section.

9.3 Starting a NIRI server

With the addition of extra status parameters for NIRI it was found convenient to have a separate vii server running on the machine called “neagle” when testing NIRI in Hilo. Having a dedicated server means there are no implications about stopping and starting it for test and diagnostic purposes whilst observing is in progress.

Special startup instructions are required as neagle mounts the gemini directory read only in a default location rather than in /gemini

To start a server on neagle:

1. Log in as gemvx. Any user can start the server but if gemvx is used most of the environment variables are set and it is possible for someone else to kill the server if this proves necessary.
2. Set the environment variable GEMINI_BASE to /home/gemvx. This just determines where the log file is written. The standard location can’t be used as neagle doesn’t have write permission.
3. Set EPICS_CA_ADDR_LIST to 10.1.5.255 This is important so that the server can’t connect to any summit systems.
4. cd to `/net/scott/export/gemini/vii/vii` and type `vis &`

9.4 Stopping the server

The server process can be stopped in two ways

1. With the standard kill command
2. Via visControl

9.4.1 Use of the kill command

Log on to the machine where the server process is running. This should probably be as gemvx as this is the account that should have been used to start the process. Issue the command

`ps -ef | grep ocssh` (if you are using /usr/bin/ps)

or

`ps -ax | grep ocssh` (if you are using /usr/ucb/ps)

in order to identify the process id and then type `kill <process id>.`

Be careful to correctly identify the process running the server, the full process listing should be something like this.

gemvx 4389 1 0 00:18:20 ? 0:04 /gemini/ocs/bin/solaris/ocssh.exe vis

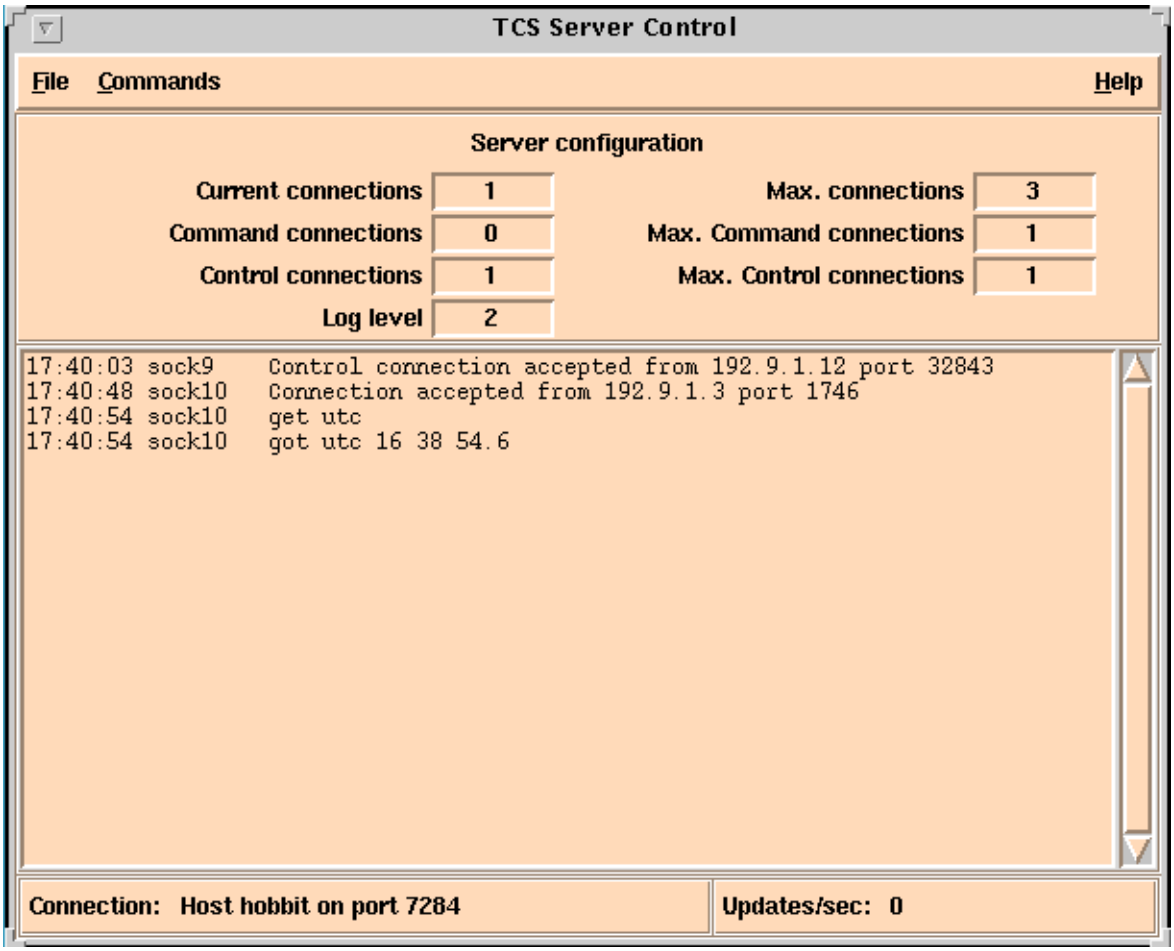
9.4.2 Stopping the process with visControl

The visControl process is described in the next section. It provides a means of interacting with the server whilst it is running. Use the ‘Commands” menu item to send a shutdown request to the server (see Section 9.5.1)

9.5 Use of visControl

VisControl provides a graphical interface to control and monitor vis. A screen shot of the current implementation is shown below.

If you find that no messages appear in the scrolling window it may be because the machine or user that was used to start the vis server didn’t have permission to write the log file. (see Section 9.6.1) If you find this is the case then the server should be restarted from a suitably privileged user or machine.



The interface consists of 4 main parts

- A menu bar with drop down menus
- A status area showing the current configuration of the server
- A scrolling window that receives copies of the server log messages
- A status bar showing the current connection and update rate

Each of these will be considered in turn

9.5.1 Menubar

The menubar consists of the entries File, Commands and Help. Each of these has a drop down pane

- **File** - this pane has commands to connect to and disconnect from a server along with the standard Exit function to shut down visControl. On startup visControl does not try to automatically connect to a server. You must connect manually via this menu.

- **Commands** - the commands currently available allow you to set the logging level and the number of connections the server will accept as well as to shut down the vis server program itself.

  The logging level can be set in the range 0 to 4. Further details about logging can be found in Section 9.6.1

  The server classifies clients under three headings:

1. Clients that only want to fetch status
2. Clients that fetch status and send commands
3. Clients that control the server itself

   The commands menu can be used to control how many of each type of client is allowed to connect to the server.

- **Help** - this brings up a simple text window with a basic description of the program. It also allows activation and deactivation of “balloon help” i.e. pop-up windows that appear when the mouse passes over a particular widget.

9.5.2 Status area

The status area shows the number of current connections to the server together with the maximum number of permitted connections. The maxima can be altered using the commands menu previously described. Also shown is the current log level.

Note that connections are classified as different types. The server keeps track independently of the number of instruments connected to it and the number of connections from control programs like visControl. Within the number of instrument connections there can be clients that only read status and clients that both read status and send commands. If a client is permitted to send commands it is automatically permitted to also read status. The number of clients permitted to send commands is always less than or equal to the total number of instrument clients. Attempts to have more command connections than total connections are trapped by the command menus.

The log level determines the number of messages that will appear in the scrolling window below. Higher numbers indicate more log information is required (see Section 9.6.1 for a description of the current log levels)

9.5.3 Scrolling window

This window shows the messages that are currently being sent to the vis log file. Each log message is time stamped and is followed by the socket number on which the message was received or sent.

9.5.4 Status bar

This area shows whether or not visControl is currently connected to vis and if so on which machine and which port it is running. For information the number of update messages per sec received by visControl is also shown.

9.6 Vis

This section provides an extended description of the vis server process. It describes the functionality of the process as well as some implementation details sufficient to allow the program to be extended and enhanced. The primary debug/test facility available with

vis is access to an interactive terminal. In operational use the commands that invoke this terminal are commented out but for development the following lines should be uncommented in the file visMain.tcl

```
# fileevent stdin readable process_cmdline
# prompt
```

Vis is implemented in Tcl/Tk using the ocssh extension. It operates as a concurrent server and thus permits multiple simultaneous connections. It maintains two completely independent socket channels. The first is used by visitor instruments that want to send commands to the TCS or fetch status from it. The second is a control channel that can be used to monitor and change the configuration of the server.

The messages received on both channels are handled by Tcl/Tk “safe” interpreters i.e. they will only respond to a very limited set of commands. This has been done for security reasons to prevent arbitrary ascii strings being sent through the firewall by port sniffers etc. The currently implemented set of commands are contained in the file parser.tcl

The operation of the server is actually quite simple, it receives a command, checks it for validity and executes it and then returns a response to the originating sender. The complications arise from the need to handle multiple connections and what to do about asynchronous messages. The asynchronous messages arise in two ways. Firstly if a monitor has been placed on an EPICS variable then updates must be sent every time the value changes. Secondly, when a command is sent to the TCS that is accepted then some time later a completion message must be sent back to the originator.

The server copes with these problems by maintaining lists of the number of concurrent connections. If a connection is received on the instrument channel then a unique identifier is added to the variable cidList, if a connection is received on the control channel it is added to cidControlList. Whenever a monitor is placed on a variable a new element is added to the array attrib indexed by the name of the variable e.g. if the variable is “utc” then the array element created will be attrib(utc). Each element of this array is itself a list. The list consisting of the unique identifiers from cidList from those clients that have put a monitor on this variable. When EPICS notifies the server that a variable has changed the server just looks at the attrib array for that element and sends messages to all the identifiers in the list. (in fact the process involves one more step, see next paragraph) In this way only those clients that have placed monitors on the variable get notified. If a client turns off a monitor then its identifier is simply removed from the list for that variable.

A complication arises when a client disconnects completely. Rather than search through the whole attrib array and remove the identifier from every element, the identifier is simply removed from cidList. This leaves identifiers in the attrib array lists that are no longer valid. This is handled as follows. Every time a variable changes and the identifiers are extracted from the attrib array element and before the message is sent out the list cidList is searched to make sure the identifier is still valid. If it isn’t then the identifier is removed from that variable and the message is discarded.

The connections on the instrument channel can be either status only or status and commands. This was done so that it would be possible to limit the number of clients that can

send commands independently of the number that could read status. It is likely that only one or possibly two clients would be sending commands at any one time but there may be many more that would want to know the current telescope status. Note that the server provides no interlocking of commands in the case that more than one client is sending commands. It is up to the clients to ensure they don't send conflicting or contradictory commands.

9.6.1 Logging

The server provides logging at different levels for diagnostic purposes. The data are logged to a file in the directory /gemini/vii/vii/log. The log file name is of the form visyyyymmmdd.log where dd is the current day, mmm is a three letter abbreviation of the month and yyyy is the year. If the log file already exists when the server is started then the data is simply appended to the end. In the case of the simulation server the log file name is visSimyyyymmmdd.log.

If the user or machine that is used to start the server doesn't have permission to write to the log file then vis will still start but no log will be created.

As well as logging to the file, the server sends a copy of the message to any connected control clients. In the case of visControl these messages are displayed in the scrolling window (see Section 9.5.3)

The current log levels are as follows

0 - all connection/disconnection events, all errors and all control commands

1 - All do commands plus disable/enable commands

2 - All get and mon commands, for mon the initial response only

3 - All command acknowledgements and completion messages

4 - All messages i.e. monitor updates, call backs added etc.

Note that setting the log level to a particular number will log all the messages below that level as well.

9.7 Installing new versions

Installation of a new version of the visitor instrument interface software is somewhat different from that of installing new versions of EPICS software. In particular it does not make use of the `instRelease` process [6]. The main reason for this is that the directory structure is different from the standard EPICS application but more importantly there is nothing to compile. Installation is therefore just a matter of checking out the software from CVS into an appropriate directory.

Assuming that the software version to install is called V1-0 then execute the following:

```
cd /gemini/vii
cvs checkout -d V1-0 vii
```

```
rm vii
ln -s V1-0 vii
```

Note that the above checkout command will extract the latest versions of all the files. If you have tagged a release and you want just the files from that release then instead use

```
cvs checkout -r V1-0 -d V1-0 vii
```

10.0 Acknowledgements

Special thanks go to Jim Wright for many useful comments on all versions of this document as well as many suggestions for the improvement of the functionality of the server.

Thanks also to Pat Wallace for providing the example C program described in Section 8.2