

NexPath Telephony Server Interface

Software Engineering Department
NexPath Corporation
Version 2.10

Telephone Operations API

1. Introduction

The NexPath Telephony Server Interface is modeled after the JTAPI object specification. In JTAPI, there are several objects which model basic telephony. Therefore, all messages sent to the server can be divided into the following objects:

- Provider
- Call
- Connection
- Address
- Terminal Connections
- Terminals

The Provider represents the service provider software. The Call Object represents a telephone call, the information flowing between the service provider and the call participants. The call can have zero or more connections. The Address object represents a telephone number. The Connection object models the communication link between the Call object and Address object. The Terminal object represents a physical device such as a telephone. Finally, the TerminalConnection object models the relationship between a call and the physical endpoint of the call, which is represented by the Terminal object.

NexPath has extended the JTAPI concept to include park orbits, and added the Park object to the model.

The NexPath model does not include the Terminal or Terminal Connection object. Instead, the interface to the telephone is modeled as a *line*, also called a physical line. One or more addresses may be associated with a *line*. The first address (i.e., extension number) assigned to a *line* is called the primary extension. The primary extension in this model is synonymous with the *line*. Whenever a telephone goes off-hook, it is assumed in this model that the primary extension address went off-hook.

There can be at most one active call object per *line*. Additionally, associated with each *line* there may be one call object for a waiting call. There may be up to three hold call objects per *line*.

2. References

More information on the JTAPI model can be obtained from the JavaSoft web site:

Java Telephony API, Version 1.1, [http://www.javasoft.com/products/javatel/Overview.htmlSoftware Interfaces](http://www.javasoft.com/products/javatel/Overview.htmlSoftware%20Interfaces).

3. Telephony Server Version

The commands described in this document are compatible with the following version:

NexPath Telephony Server Release 7.

4. Messages

Communication with the NexPath Telephony Server is via a socket connection using the TCP/IP protocol. Commands messages are sent to the Server using simple ASCII commands, terminated by a new-line (line feed, ASCII 10 decimal), and similarly responded to by the Server. Every command sent to the Server is acknowledged except the **NOP_** command, which is the response to the keep alive request (See “Keep Alive” at 5.3 on page 4). The Server may send asynchronous event messages to the client indicating a change in state of an object in the system, if a command has previously been sent to the Server registering for that asynchronous message class. A message must complete to a newline within two seconds or it will be rejected, so messages should be blocked into complete lines before sending.

More than one socket interface may be opened on each line (i.e., extension or address) by one or more clients. Multiple objects may be monitored with the asynchronous event feature.

All commands are terminated with an end of line. Those that expect the return of indeterminate length data have the number of bytes being sent embedded in the response, and contain that many bytes following the end of line. Data is at most 4096 bytes per request. Command parameters are space separated. The server assigns all call ids and handles.

The server response messages will always be in the following format:

```
<msgid> <number> <conversational text> "<variable data>"\n
```

where the “< . . . >” notation indicates data that is filled in depending on the message. The “<” and “>” do not actually appear in the response, but are used to indicate data that depends on the specific instance of the message. The <number> is a message class identifier, attempts to follow the *ftp* numbering standard to distinguish between error responses and commands that were accepted as valid. The <conversational text> provides easily readable information about the message and data. The <variable data> is always contained within double quotes and contains multiple keyword-data pairs which are outlined in detail below. The msgid, number, and beginning of conversational text are separated from each other by one or more spaces. The conversational text may (and usually does) contain embedded spaces. The variable data is always enclosed by double quotes and uses tags with equal signs to provided specific parameter values.

Both verbose and terse forms of command mnemonics are presented in this document. Either may be used and both are recognized by the Server.

4.1 Message Identifiers

Each message sent to the server must have an 1-10 character alpha-numeric identifier in the first space parsed field. This identifier will be returned with the response message. For instance,

```
55643 lgon 300 1000
```

would return:

```
55643 201 User logged in. " Name=Administrator UserId=0 ExtOwner=99
Priv=admin "
```

where 55643 is the identifier for this message. If the message is unsolicited from the server, “*” will be placed in the first field as a general message identifier.

In the remainder of this document, the message identifiers are assumed, and for clarity are not shown in each message example.

4.2 Keyword-Value Data Pairs

The variable data enclosed in quotes contains keyword-value data pairs separated by equal signs. The keyword-data pairs contain no embedded spaces, and are separated from each other by one or more spaces.

Example:

```
* 223 CALE      ` CallId=10  CallState=Active CallType=NormalCall `
```

4.3 URL Encoding *Cname*

The *cname* field used to send the name decoded from the Called Named Delivery message will use modified URL encoding. Each unsafe character in the *cname* will be encoded with %xx, where xx are two hexadecimal digits representing the character, from the set "01234567890ABCDEF". Lower case "abcdef" may also be used. Unsafe or reserved characters are (for this implementation):

```
unsafe = <"> | "%" | "=" | " " | ","
```

where the above is written as <delimiter> char <delimiter>, and the "|" symbol means "OR". Any non-printing or graphics character is also unsafe. The SPACE is URL encoded with %20, not replaced with a "+" symbol.

Characters that are reserved and have special meaning, such as "=", are not URL encoded if their special meaning is to be preserved.

Note that user names and file names may **not** contain these or any other non-alphabetic characters. Additionally, a conversion of "_" to " " (underbar to space) takes place in the display of user names and file names in AdminTool.

4.4 Error Responses

All error responses are documented in Appendix A and associated to commands with the terse mnemonic. Therefore, these error responses will not be repeated in the message specification below.

5. Provider

5.1 Port Number

Connections are made via TCP/IP at port **5000** on the NexPath Telephony Server.

5.2 Login

Each connection to the server must first log in with the `lgon` command. The login requires a valid extension number on the system, and a valid user password. The client logs out and closes the socket connection using the `close` command.

Commands:

```
lgon      ext      pswd
LGON     ext      pswd
```

```
close
CLOS
```

The server responds with:

```
201 User logged in. "Name=username UserId=usernumber
ExtOwner=<owner_number> Priv=[admin|normal]"
```

where

ext - the address that this socket will control. Usually the user's extension. Must be a valid inside extension. Usually, this is the extension next to the computer controlling it. The *ext* can be an inside or outside line.

pswd - The user's 4 digit password.

If the login attempt fails with any of the above errors, the Server closes the socket.

For third party call control, the ext must be 3PCC. The password must be a valid user with admin privileges.

Commands:

lgon	3PCC	<i>pswd</i>
LGON	3PCC	<i>pswd</i>

The server responds with:

201 User logged in. "Name=3PCC UserId=*username* ExtOwner=*username* Priv=admin"

A limited set of commands are available for third party call control including playAudio and transfer.

5.3 Keep Alive

Every two minutes a heartbeat, or keep alive signal is sent from the server to all connections. The client must respond within two minutes, or the connection is closed.

Server sends command

340 ATST

and expects the client response

NOP_

Any other command will also count as a response and the keep alive timer will be reset. The NexPath Server does not respond to or acknowledge the **NOP_** command.

5.4 Set Day and Night Ringing

Command to set day ringing mode:

setDayRing
DRNG

Server Response:

218 Day Ringing is now on

Command to set night ringing mode:

setNightRing
NRNG

Server Response:

219 Night Ringing is now on

Current status about day and night ringing can be obtained with the command **GETS PROV_EV**. See "Requesting Status Messages" at 8.2 on page 23. The logged in user must have permission to set the day/night ringing mode.

5.5 Change Password

changePwd	<i>old</i>	<i>new</i>
CPWD	<i>old</i>	<i>new</i>

Server Response:

233 Your password is now changed
where:

old - valid password for this user

new - desired password, valid. Changing a user's password will not cause any event in the system.

5.6 Extensions for Line

The primary extension, virtual extensions, and physical line number associated with any normal extension can be found with the following command.

getAddr *ext*

GETA *ext*

Server Response:

203 OKOK "PriExt=*ext* Line=*line_no* Exts=*ext,ext,...*"

where:

ext - any extension of interest.

line-no - Is the line number in the form OL-*nn* or IL-*nn*.

Examples:

getAddr 398

203 OKOK "PriExt=315 Line=IL-15 Exts=315,398"

getAddr 303

203 OKOK "PriExt=303 Line=IL-3 Exts=303,300,411,611"

6. Address

6.1 Call Forwarding

setForward *FromAddress* *ToAddress*

FWRD *FromAddress* *ToAddress*

cancelForwarding *FromAddress*

CFWD *FromAddress*

Server Response:

200 OKOK

where:

FromAddress - Extension from which to forward

ToAddress - Extension or phone number (offsite) to which to forward.

The user must own the extension from which calls are forwarded or be the administrative user. If the calls are to be forwarded offsite, the extension must have offsite forwarding enabled.

6.2 Do Not Disturb

doNotDist *address*

OFLN *address*

cancelDnd *address*

ONLN *address*

Server Response:

231 Do not Disturb mode for "Addr=address".

232 Now accepting calls "Addr=address".

where:

address - Extension to set "Do Not Disturb" or set to accepting calls.

6.3 Call Waiting

callWaitingOn *address*

EWTG *address*

CallWaitingOff *address*

DWTG *address*

Server Response:

234 Call waiting is now on.

235 Call waiting is now off.

where:

address - extension for enable/disable call waiting

The line must not be a secure line.

6.4 Set Voicemail Preferences

This command is used to set numeric pager parameters, to select standard or alternate greetings, to set email parameters, and to set voice mail distribution groups. Numeric pages or email are sent after receipt of a voicemail message. The duration between the pager connection and the sending of digits may also be specified. This is provided to optimize the interface to various pager companies. Additionally, if the pager company does not recognize "*" as a separator, a "#" or nothing can be designated as a separator. Separators are used to separate the caller ID of the telephone call, from the voice mail box number and the number of unread messages, which are transmitted to the pager. A separator should show up as a "-" in the display of the pocket pager. Most companies use "*" for the separator. Finally, if the messages in the voicemail box are to have/not have the caller id or time/date voice announcements prepended, the option can be set here.

To view the voice mail preference settings of a mail box; query the address status of the voice mail box extension: GETS ADDR_EV *ext*. See "Requesting Status Messages" on page 23 and "Address Status" on page 24

Numeric pager commands:

To specify a pager number:

setVmPref *ext* **PAGER** *address*

SVMP *ext* **PAGER** *address*

To clear a previous pager number designation:

setVmPref *ext* **PAGER**

SVMP *ext* **PAGER**

To specify silence duration between calling pager and sending digits:

setVmPref *ext* **DUR** *seconds*

SVMP *ext* **DUR** *seconds*

To clear silence duration:

setVmPref *ext* **DUR**

SVMP *ext* **DUR**

To specify numeric pager separator (blank or empty means no separator):

setVmPref *ext* **SEP** [# | * | <blank>]

SVMP *ext* **SEP** [# | * | <blank>]

To specify that caller id header (from outside lines only) or time/date header is to be prepended to voice mail messages:

setVmPref *ext* **CID_H** [on | off]

SVMP *ext* **CID_H** [on | off]

setVmPref *ext* **TD_H** [on | off]

SVMP *ext* **TD_H** [on | off]

To specify playing of the standard/alternate greeting:

setVmPref *ext* **GREETING** [alternate | standard]

SVMP *ext* **GREETING** [alternate | standard]

Email can be sent with just the header (time, date, length, etc.), or with the audio file attached (ENTIRE option), and either with a regular or short text message (ELEVEL option). To delete the voice mail file after sending the email, select the EDELETE option.

To specify email :

setVmPref *ext* **ENAME** *email_address*

SVMP *ext* **ENAME** *email_address*

setVmPref *ext* **EHOST** *email_host*

SVMP *ext* **EHOST** *email_host*

setVmPref *ext* **EATTACH** [entire | header]

SVMP *ext* **EATTACH** [entire | header]

setVmPref *ext* **EDELETE** [yes | no]

SVMP *ext* **EDELETE** [yes | no]

This command sets the length of the text message, where *short* is an abbreviated text message for suitable for cell phones:

setVmPref *ext* **ELEVEL** [reg | short]

If the SMTP host requires a login, the login username and password can be set with the following commands:

setVmPref *ext* **EUSER** *username*

setVmPref *ext* **EPASS** *password*

The SMTP login password is stored encrypted on the NexPath Telephony Server, and is never echoed back in cleartext. Only “***” are returned when querying the voice mail box address status.

To specify voice mail distributions groups:

```

setVmPref ext VMDG add_group group_au_file ext1,ext,... extn
SVMP      ext  VMDG add_group group_au_file ext1,ext2,... extn

```

```

setVmPref ext VMDG del_group group_au_file
SVMP      ext  VMDG del_group group_au_file

```

Server Response:

200 OKOK

where:

ext - voicemail extension
address - offsite pager number
seconds - enter between 0 and 10 seconds for silence duration.
email_address - email addresses. This replaces any previous email address.
email_host - host computer sending email.
group_au_file - audio file saying the identity of the group
ext1,...,extn - voice mail extensions comprising the voice mail distribution group

Note: No spaces allowed between extensions when adding a voice mail distribution group.

To associate the group with all voice mail extensions, use the keyword 'all_vm' in place of the extension list.

Add group will add the group to the user's local voice mail directory. To add a system group, only the *group_au_file* is needed (use a full path to the system distribution group directory, /var/cbts/config/system_vm_groups). The user is not allowed to modify the extensions defined for system groups. Deleting the group removes it from the local directory, therefore, removing the ability for that user to forward voice mail to that group. For personal groups, the *group_au_file* must be the filename relative to the /var/cbts directory. The .au file must already exist before adding the group. If the file is removed and the group still exists, then the option will not be given to the user when the attempt to forward is made

Example:

```

setVmPref 501 VMDG add_group /var/cbts/config/system_vm_groups/somevm.au
setVmPref 501 VMDG add_group /vmail/501 group1.au 502,503.

```

6.5 Copy Voice Mail

This command is used to copy voice mail messages. The voice mail filename is relative to the /var/cbts directory. Currently, if the short name is available it will be announced as the forward-from name. Otherwise, the forward-from extension will be announced.

```

cpVm      from_ext to_ext vmFileName
CPVM     from_ext to_ext vmFileName

```

Server Response:

200 OKOK

where:

from_ext - voicemail extension where the file is copied from
to_ext - voicemail extension where the file is copied to
vmFileName - voice mail file name relative to */var/cbts*

A new filename is created where the first 3 characters are the extension that this file is forwarded from. If there is a short name, “Forwarded from”+*short_name* is played, otherwise “Forwarded from”+*ext* is played. If there is a prepend message identified by the original vm file name with ‘.pre’ added to the end, then that file is copied. Finally, the forwarded message is copied.

If the vm file name is: *Wed_Mar__1_08:33:09_2000a.au*

Then the prepend message filename is: *Wed_Mar__1_08:33:09_2000a.au.pre*

6.6 Copy Voice Mail to Distribution Group

This command is used to copy voice mail messages to a distribution group. The voice mail filename is relative to the */var/cbts* directory. Currently, if the short name is available it will be announced as the forward-from name. Otherwise, the forward-from extension will be announced. The distribution group name (*vm_group_name*) is the .au audio file associated with the group. For system groups, the full path must be provided to the system group directory */var/cbts/config/system_vm_groups/*; for personal groups, the path is relative to */var/cbts/*.

```
vmToGroup    from_ext vm_group_name vmFileName
CPDG        from_ext vm_group_name vmFileName
```

Server Response:

200 OKOK

where:

from_ext - voicemail extension where the file is copied from
vm_group_name - group audio file name relative to */var/cbts* or full path for system group
vmFileName - voice mail file name relative to */var/cbts*

A new filename is created where the first 3 characters are the extension that this file is forwarded from. If there is a short name, “Forwarded from”+*short_name* is played, otherwise “Forwarded from”+*ext* is played. If there is a prepend message identified by the original voice mail file name with ‘.pre’ added to the end, then that file is copied. Finally, the forwarded message is copied.

If the voice mail file name is: *Wed_Mar__1_08:33:09_2000a.au*

Then the prepend message filename is: *Wed_Mar__1_08:33:09_2000a.au.pre*

This message is then sent to all extensions in the voice mail distribution group.

Examples (note the full path for the system voice mail group):

```
vmToGroup 501 group1.au vmail/501/Fri_Apr_14_12:51:14_2000a.au
```

```
vmToGroup 501 /var/cbts/config/system_vm_groups/allvm.au vmail/501/
Fri_Apr_14_12:51:14_2000a.au
```

6.7 Push Address Event

This command is used to cause the system to re-examine the voice mail box indicated by *ext* to determine if new messages are present. If so, the appropriate events are triggered and voice mail indicators refreshed for the affected lines. The **pushAddrEv** command is required when direct file system oper-

ations are performed on voice mail directories, to notify the telephony software that a change has occurred.

```
pushAddrEv    ext
PSHA          ext
```

Server Response:

```
200 OKOK
where:
    ext - voicemail extension
```

6.8 Set Paging Address (obsolete)

```
setPaging    ext    address
SETP         ext    address
```

Server Response:

```
200 OKOK
where:
    ext - voicemail extension
    address - offsite pager number
```

7. Call

A Call can be in one of three states:

- Active - indicates that the call has one or more connections, none of which are in the disconnect state.
- Idle - indicates that the call has zero connections.
- Invalid - indicates that the call has lost all of its connections.

For a call id which is not known to the system, such as prior to the call's existence and after it has been removed from the system, a query of the call id will respond, "503 Cannot get status for this call id".

7.1 Call Creation

Calls can be created (*computer generated calls*) by the client:

```
createCall
```

CCAL

Server Response:

```
202 Create Call OK "CallId=cid".
```

This command reserves a call id (returned) and sets this client as the controller for the call. Upon completion of this command, no connections are established to this call. All subsequent commands are sent to the "other" line. Therefore, if a command to connect is made, it is as though the computer is the near end and the phone is the far end. To complete a computer generated call, after connecting to the far end, the call must be transferred to the desired party or disconnected from the client. The following is an example of how to create and complete a call between extension 303 and extension 305:

```
createCall
202 Create Call OK "CallId=2".
callAddress 2 305
204 Address Called "CallId=2".
transfer 2 303
234 Call transferred "CallId=2".
```

There is a restriction of one computer generated call per client. Requesting a second active computer generated call will disconnect an active one or return an error. The following commands can be used with computer generated calls:

CLEX, **DIAL**, **SZLN**, **HKFS**, **PLAA**, **DISC**, **BXFR**, **CONF**, and **PURC**.

7.2 Call Connect

7.2.1 Call Address

If the client has established a computer generated call, then the call address command attempts to connect the address to the call id of the computer generated call. That is the callid in the command must be that of the computer generated call. In the case where the call is placed for the phone the client has logged into, then the phone must be off hook and receiving dialtone.

```
callAddress    callId address  
CLEX          callId address
```

Server Response:

```
204 Address Called "CallId=callId".
```

7.2.2 Send DTMF Digits (Touch Tones)

This command is supported for either computer generated calls or calls on the line logged into. If the computer is controlling the telephone line, this command can only be sent in the Connect state.

```
sendDigits    callId  digits  
DIAL         callId  digits
```

Server Response:

```
205 Digits sent "CallId=callId".
```

7.2.3 Seize Outside Line

If the client has established a computer generated call, then seize attempts to seize an outside line and connect it to the computer generated call. Activity at the user's telephone is not relevant in this case. Otherwise, in non-computer generated calls, the telephone must be off hook and receiving dialtone.

```
seize        callId  group_number  
SZLN        callId  group_number
```

Server Response:

```
207 Line seized "CallId=callId".
```

The seize group number is optional. If logged in using an extension, the inside lines seize group is used regardless of whether a group number is specified. Otherwise, if logged in using the third party call control (3PCC) then the specified seize group is used. If no seize group is given in this case, then the seize group of 0 is used.

7.2.4 Hook Flash

If the call includes a central office line configured for hook-flash, then a hook flash is sent to the central office line. The duration of the hook flash is set by the configuration file. A hook flash is a signalling technique where the line goes on-hook or interrupts the flow of current for a short duration, usually about 750 milliseconds. This command cannot be used for conference calls.

```
CoHookFlash  callId  
HKFS        callId
```

Server Response:

209 OK "CallId=*callId*".

7.2.5 Disconnect

This command is supported for computer generated calls. Computer generated calls disconnect the client as the controller of the call. The extension parameter is not used in this case. For clients controlling a telephone, if the phone is ringing, a disconnect message is also sent to the caller. This command is not allowed if the phone is on hook, off hook, ringing or in a call distribution queue. Offhook in this case means when the phone is offhook and received dialtone timeout. Care should be taken when sending disconnect requests to outside lines. If a phone call is up, the call will be taken down.

```
disconnect  callId  ext
DISC       callId  ext
```

Server Response:

208 Call disconnected "CallId=*callId*".

where:

callId - call id.

ext - current ext to which to send this command.

7.3 Hold

7.3.1 To Hold

According to JTAPI, the hold features are associated with a terminal. A terminal is a device (usually a phone) which has an associated address (phone number or extension), but is not necessarily the only device associated with that phone number. This acts like the local hold button on a phone.

With the NexPath system, hold is associated with a *line*.

The phone must be connected or on hold. This command is not supported with computer generated calls.

```
toHold    callId
THLD     callId
```

Server Response:

231 Call put on hold. New *callId* generated. "CallId=*callId*".

7.3.2 From Hold

From Hold is not supported with computer generated calls. The call id is required to be that of the call at the top of the hold stack (Hold1 - See Asynchronous Event Messages). The phone must be receiving dialtone.

```
fromHold  callId
FHLD     callId
```

Server Response:

232 Call from hold "CallId=*callId*".

7.4 Swap Hold

SwapHold will attempt to swap the current call with the *callId* of a call on hold. The *callId* in the command must be that of the call on hold to swap. The hold call must be at the top of the hold stack

(Hold1). The phone must be connected or on hold. This command is not supported with computer generated calls.

swapHold *callId*

SHLD *callId*

Server Response:

233 Call swap hold "CallId=*callId*".

where:

callId - current call id or an associated call id for call waiting or calls on hold.

7.5 Transfer Call

7.5.1 Transfer

The call must be in the connect state or a call on computer hold. For computer hold calls, an additional Address2 is required, so that **transfer** will transfer the caller extension (Address2) of the call id which the computer is connected to, to the specified Address (the extension to which to transfer the call).

transfer *callId* *Address* [*Address2*]

BXFR *callId* *Address*

Server Response:

234 Call transferred "CallId=*callId*".

7.5.2 Consultation Transfer

Not supported with computer generated calls. Call must be in the connect state.

transferWithConsult *callId*

SXFR *callId*

Server Response:

235 Call transfer setup "CallId=*callId*".

7.5.3 Complete Consultation Transfer

Not supported with computer generated calls.

transferComplete *callId*

CXFR *callId*

Server Response:

236 Call transfer complete "CallId=*callId*".

where:

callId - current call id or an associated call id for call waiting or calls on hold. The *callId* returned after a transfer will be the new current call id for that call (in dialtone state).

Address - current extension for the call to be transferred

7.6 Pickup Calls

7.6.1 Pickup Ring Group or extension

The telephone must be receiving dialtone.

```
pickup group_or_ext
PURN group_or_ext
```

Server Response:

```
237 Group pickup "CallId=callId" .
where:
```

group_or_ext - group number to pick up or extension of phone to pick up (receiving ring-back).

7.6.2 Pick up Ringing Call

This command will pickup a ringing call (or a waiting call) and connect it to the current valid computer generated call id (i.e., put the call on computer hold).

```
pickupCall callId address
PURC callId address
```

Server Response:

```
209 OKOK "CallId=xx"
where:
```

callid- call id of call which has connection in ringback.

address- address of the ringing phone (or waiting phone extension).

Pickup ringing call will pick up the call ringing or waiting at the given address (extension). This is only supported with computer generated calls, i.e., the ringing/waiting call is picked up and put on computer hold. The call must be transferred or it will be on hold until the caller disconnects. For example, if you want to pick up a ringing extension and transfer it to voice mail the sequence is (assume the ringing call Id = 21):

```
p pickupCall 21 302
p 209 OK "CallId=21"
q GETS CALL_EV 21
q 224 CALS "CallId=21 Caller=205,4082358916,NexPathCorp, ..."
x transfer 21 599 205
x 234 Call transferred "CallId=21"
```

7.6.3 Pickup Waiting Call

Pickup waiting call is not supported for computer generated calls. To pickup a waiting call to a computer generated call id, see "Pick up Ringing Call" at 7.6.2 on page 14 above. The phone must be receiving dialtone or be connected in a call for this command.

```
pickupWaitingCall waitingCallId
PKUP waitingCallId
```

Server Response:

```
241 Waiting call pickup "CallId=callId" .
where:
```

waitingCallId- call id of the waiting call.

callId- call id of the picked up call.

7.7 Conference Circuits

7.7.1 Conference

The **CONF** command takes the current call id of the computer generated call and the specified call id and merges them into one call. Both calls **MUST** be connected to outside lines. Otherwise, use the transfer command. This command **ONLY** works for computer generated calls.

```
conference    callId
CONF         callId
```

Server Response:

```
242 Conference "CallId=callId" .
```

7.7.2 Set up Conference

Not supported with computer generated calls. The phone must either be a party to a call or receiving dialtone.

```
setupConf    callId
SCNF         callId
```

Server Response:

```
243 Setup conference "CallId=callId" .
```

7.7.3 Add to conference

Not supported with computer generated calls. The client's phone must be actively connected to a call.

```
addToConf    callId
CCFR         callId
```

Server Response:

```
244 Add to conference "CallId=callId" .
```

7.7.4 Join Call

When a request to join (**BARGE**) a conversation is made, usually the conversion in progress is two-way. Therefore, when the request is made, a conference is set up and the third party is added to that conference. Otherwise, if a caller in voice mail, he will be connected in a two-way conversation with the line requesting the **BARGE**. The call identifier is that of the call to be joined to or barged into. The address is the extension of the party being barged in to. **BARGE** does not work for conference calls. **BARGE** is not supported with computer generated calls. The client must have barge-in privileges. Both connections on the barged in call must not be on secure lines. The client's phone must be receiving dialtone for this operation.

```
join         callId  Address
BARG        callId  Address
```

Server Response:

```
245 Join call "CallId=callId" .
```

7.7.5 Listen Call

When a request to listen to a conversation is made, the requestor will listen in on the conversation. The address is the extension of the party to listen to. **LIST** does not work for conference calls. **LIST** is not supported with computer generated calls. The client must have barge-in privileges. Both connections on the barged in call must not be on secure lines. The client's phone must be receiving dialtone for this operation.

```
listen      callId  Address
```

```
LIST      callId  Address
```

Server Response:

```
247 Listen call "CallId=callId" .
```

WARNING: *Listening in to a conversation without the consent of one or both parties may be a violation of local, state, and federal privacy laws. It is the responsibility of the user of the NexPath Telephony Server, when using features of the system, to assure that he or she is in compliance with all applicable laws.*

7.8 Audio Operations To/From a File

Record/Play audio through the phone is used to play an audio file to a listener or to record an audio file from a single extension (single ended call). The telephone user must take the phone off-hook and be receiving dialtone for these operations to work. The `recordThruPhone` and `playThruPhone` commands are used from the **AdminTool** to allow administration of the audio files associated with the automated attendant.

When it is desired to play a file to the “other” end or “far end” of a call, the command `PlayAudio` is used. A call must be up and the desired audio file is played to the listener who has picked up the call. Additionally, the command can play an audio file to a telephone user who has been put on hold. This command is used when a caller is put on hold prior to the system attempting to find the called party.

7.8.1 Record a Message To a File

This is used to record a message to be used in the automated attendant, for example. It only records a single party. To record a conversation (two party), see “Record Conversation” on page 17.

```
recordThruPhone  callId  audio_filename
ADMR             callId  audio_filename
```

Server Response:

```
238 Recording "CallId=callId" .
```

7.8.2 Play a message file

Play a message file to near end (originating) party. Extension must be in dial tone state to hear the message file.

```
playThruPhone  callId  audio_filename
ADMP           callId  audio_filename
```

Server Response:

```
239 Playing "CallId=callId" .
```

7.8.3 Play a message file to called party

Play a message file to other end of a call.

```
playAudio      callId  audio_filename
PLAA           callId  audio_filename
```

Server Response:

```
209 OK "CallId=callId" .
```

where:

callId - current call id or an associated call id for call waiting or calls on hold.

audio_filename - complete filename including relative path from */var/cbts*. See note at "Discrepancies and Open Issues" on page 26.

7.9 Record Conversation

A two party conversation can be recorded to a file using these commands. Recording a conversation requires several permissions be properly set, and that local laws be followed. See the **Administration Guide** for details. The recording is saved in the voice mail extension directory with the time and date stamp with an "Rcd_" prefix.

```
startRecord  callId      vm_extension
STRC        callId      vm_extension
```

Server response:

```
209"CallId=callId File=/var/cbts/vmail/nnn/Rcd_mmm_dd_hh:mm:ss_yyyya.prg"
The ".prg" suffix is changed to ".au" after the recording is completed, and the system post processes the audio file. If permissions are not correct, the following is sent:
574 Unauthorized user for this operation. ""
```

```
stopRecord   callId
SPRC        callId
```

Server Response:

```
209 OK "CallId=5"
```

Complete example of interchange to start and stop recording (the message identifier is shown as consecutive starting at 1):

```
1 GETS CONN_EV 303
1 228 CONS "Addr=303 CallId=5 ConnState=Connect   WaitingCallId=none   "
2 GETS CALL_EV 5
2 224 CALS "CallId=5 Caller=303,303,Exten%20303,08/21,14:13 Called=305
CallState=Active CallType=NormalCall Record=off"
3 STRC 5 503
3 209 OK "CallId=5 File=/var/cbts/vmail/503/Rcd_Aug_21_14:14:14_2003a.prg"
4 GETS CALL_EV 5
4 224 CALS "CallId=5 Caller=303,303,Exten%20303,08/21,14:13 Called=305
CallState=Active CallType=NormalCall Record=on"
5 SPRC 5
5 209 OK "CallId=5"
6 GETS CALL_EV 5
6 224 CALS "CallId=5 Caller=303,303,Exten%20303,08/21,14:13 Called=305
CallState=Active CallType=NormalCall Record=off"
```

WARNING: Recording a conversation without the consent of one or both parties may be a violation of local, state, and federal privacy laws. It is the responsibility of the user of the NexPath Telephony Server, when using features of the system, to assure that he or she is in compliance with all applicable laws.

7.10 Record Short Name (ShortName.au)

The `recShortName` command is designed to allow the interface to create the short name file for a voice mail box in much the same way as the telephone user interface. The system will play the 'recording' message and record the file for 3 seconds. The file will be stored in `ShortName.au` in the voice

mail extension file path. If there already exists a short name file, it will be overwritten. Upon completion of the recording, then the file will also be copied to the `/var/cbts/config/AutoAtten` directory for use with the dial by name feature. The extension must be in the dial tone state for the command to succeed.

```

recShortName callId      vm_extension
RSNM         callId      vm_extension

```

where:

callId - current call id or an associated call id for call waiting or calls on hold.

vm_extension - voice mail extension that this short name is associated with. .

7.11 Operator

This command is not supported for computer generated calls. The line may either be an inside or outside line. The phone must be receiving dialtone.

```
OPER callId
```

```
oper callId
```

Server Response:

```
240 Operator "CallId=callId".
```

where:

callId- call id of the phone receiving dialtone.

8. Monitoring and Status Commands (Events)

The NexPath Server Interface provides both asynchronous and synchronous (status) information about the objects in the telephone system. Asynchronous information is sent spontaneously by the Server to any connection that has registered for that information. Status (synchronous) information can be obtained at any time based on a request from the client.

8.1 Asynchronous Messages

The client can register for asynchronous notification of a change in the state of the provider, the address, one or more call objects, connections associated with one or more addresses, or changes in park orbits. In general, only changes in the status are reported. There are some exceptions to this as noted in the sections below.

8.1.1 Registering for Asynchronous Event Messages

Clients can register for asynchronous events from the Server with the following commands:

```
SETE [CALL_EV | ADDR_EV | PROV_EV | CONN_EV | PARK_EV]
[<ext>|<callId>]
```

```
setEvents [CALL_EV | ADDR_EV | PROV_EV | CONN_EV | PARK_EV]
[<ext>|<callId>]
```

Only one event specifier may be present per command. Only those events associated with the event type will be sent. After registering for an event, the asynchronous events will be sent to the client without any further requests, as the events occur on the Server. This will continue until the event is removed with the RMEV command described below, or the object or item has ceased to exist (i.e., the call id is no longer valid), or the socket connection is closed.

Leaving out a specific address or call id will result in registering for all events of that type. That is, the command:

SETE ADDR_EV
will return all address events.

To receive call events for a particular call id:

SETE CALL_EV callId

To receive address events for a particular address:

SETE ADDR_EV ext

To receive connection events for a particular address:

SETE CONN_EV ext

Note that setting the event for park orbits (**SETE PARK_EV**) will always monitor all park orbit extensions.

The current SETE command will **not** remove previous commands for the event types not in the command. That is, setting the ADDR_EV for a particular address will not affect reporting of CONN_EV, CALL_EV or PROV_EV. But, it will add to the current reporting of the ADDR_EV.

The event types can have their reporting removed with the command:

RMEV [CALL_EV | ADDR_EV | PROV_EV | CONN_EV | PARK_EV] [*ext* | *callId*]

where:

PROV_EV - general provider information
ADDR_EV - information specific to the address
CALL_EV - relative to a particular call
CONN_EV - relative to a connection
PARK_EV - relative to park orbits
ext - a valid normal extension
callId - call id for the call

Server Response:

210 Async Call logging is on.
211 Async Call logging is off.
212 Async Connect logging is on.
213 Async Connect logging is off.
214 Async Address logging is on.
215 Async Address logging is off.
216 Async Provider logging is on.
217 Async Provider logging is off.
236 Async Park logging is on.
237 Async Park logging is off.

8.1.2 Provider Events

The provider event message is in the following format:

225 PROE "OL-line= [Deadline | Ready] "

This event is sent when a line enters the deadline state or comes back to availability. If the line was in the by-pass state, then when the call is released, an event will be triggered and the ready status will be returned.

The following event is sent when the system toggles from the day to night and night to day mode.

225 PROE "Mode= [day | night] "

Only the data that has changed is reported in the event.

8.1.3 Address Events

Address Event messages are of the format:

229 ADDE "Addr=addr a=b c=d ..."

where a=b are mnemonic, data pairs as specified below.

The following event is sent when the extension is placed in the on-line or off-line mode.

229 ADDE "Addr=addr DND= [yes | no] "

The following event is sent when the forwarding address is set or cleared.

229 ADDE "Addr=addr Forward= [addr | none] "

The following event is sent when call waiting is set or cleared for this extension.

229 ADDE "Addr=addr CallWaitingEnabled= [yes | no] "

One of the following events is sent when a voice mail preference of voice mail box extension *vmAddr* is set or changed, as the result of a **setVmPref** command (see "Set Voicemail Preferences" on page 6).

229 ADDE "Addr=vmAddr PagingNum= [addr | none] "

229 ADDE "Addr=vmAddr Separator= [addr | none] "

229 ADDE "Addr=vmAddr Delay= [0..10] "

229 ADDE "Addr=vmAddr CIDHeader= [on | off] "

229 ADDE "Addr=vmAddr TimeDateHeader= [on | off] "

229 ADDE "Addr=vmAddr Greeting= [standard | alternate] "

229 ADDE "Addr=vmAddr emailName= email_name "

229 ADDE "Addr=vmAddr emailHost= email_host "

229 ADDE "Addr=vmAddr emailAttachAudio= [entire | header] "

229 ADDE "Addr=vmAddr emailDeleteAfterSend= [yes | no] "

229 ADDE "Addr=vmAddr emailUser= username "

229 ADDE "Addr=vmAddr emailPass= **** "

229 ADDE "Addr=vmAddr vmDistGroup= filename.au; ext1, ext2... extn "

Example (showing voice mail distribution groups are space separated for address events)

```
229 ADDE "Addr=227 vmDistGroup=group1.au;227,744 vmDistGroup=/var/cbts/config/system_vm_groups/somevm.au; "
```

The following event is sent when voice mail is left, or the messages are listened to for the first time or deleted.

229 ADDE "Addr=vmAddr NewMsgs=numMsgs OldMsgs=numMsgs"

8.1.4 Call Events

Call Event messages are in a variety of formats as described in this section. If the call state changes, the following event is sent:

223 CALE "CallId=callId CallState=[Invalid|Idle|Active]
CallType=ctype Record=[on|off]"

Whenever the extension or caller id of either the caller or the called party is changed, the event below will be sent. All fields of this message are reported in the event message. Since the switch cannot discriminate dialed digits and called addresses for outbound calls, the digits dialed will be sent with each call event or call state. When no digits have been dialed, the Digits field is left blank.

223 CALE "CallId=callId Caller=addr,cid,cname,calldate,calltime
Called=addr Digits=<digits> CallState=[Invalid|Idle|Active] CallType=ctype
Record=[off|on]"

The following message is the call event message for the conference calls. The message has been formatted for clarity; there are no embedded newlines in any Server Interface responses. Conference calls do not have a call id associated with the conference. A Conference call is merely a *grouping* of up to four call ids. Each call id in the conference has a Digits field (dialed digits if the call was made to an outside line), and an Ext field with information about what the extension and number that placed the call.

223 CALE "CallId=callId CallId1=callId Ext1=addr,cid,cname,call-
date,calltime Digits1=<digits> CallId2=callId Ext2=addr,cid,cname,call-
date,calltime Digits2=<digits> CallId3=callId
Ext3=addr,cid,cname,calldate,calltime Digits3=<digits> CallId4=callId
Ext4=addr,cid,cname,calldate,calltime Digits4=<digits> CallType=ConfCall
Record=[on|off]"

The Ext field data is as follows:

addr - extension, if available

cid - caller id, if supplied

cname - called name, if supplied, URL encoded

calldate - call date if available from the caller identification information in the form mm/dd

calltime - call time in the form hh:mm:, if available

The comma delimiter will be present regardless of whether the data is available. If the called name data is available, it will be URL encoded per 4.3. Examples of some valid call events are:

223 CALE "Callid=3 Caller=203,5551212,,, called=308 Digits= Call-
State=active CallType=NormalCall Record=[on|off]"

223 CALE "CallId=4 Caller=203,P,,, Called=315 Digits= CallState=active
CallType=NormalCall Record=[on|off]"

223 CALE "CallId=5 Caller=305,,,, Called=300 Digits=7671111 Call-
State=active CallType=NormalCall Record=[on|off]"

223 CALE "CallId=5 Caller=,5551212,Acme%2C%20Inc.,, Called=300 Digits=
CallState=active CallType=NormalCall Record=[on|off]"

```
223 CALE "CallId=5 Caller=,5551212,Acme%2C%20Inc.,, Called=300 Digits=
CallState=active CallType=NormalCall Record=[on|off]"
```

```
223 CALE "CallId=3 Caller=203,5551212,Acme%2C%20Inc.,7/21,3:25 Called=308
Digits= CallState=active CallType=NormalCall Record=[on|off]"
```

```
223 CALE "CallId=5 CallId1=4 Ext1=305,,, Digits1= CallId2=6 Ext2=301,,,
Digits2= CallId3=5 Ext3=303,,, Digits3= CallId4= Ext4=,,, Digits4=
CallType=ConfCall Record=[on|off]"
```

The caller identification information (*cid*, *cname*, *date*, *time*) is only available on incoming calls. In order to find the called number for outgoing calls, the digits from the connect state (see “Connection Events” at 8.1.5 on page 22 below) must be monitored. After a four second delay, the digits dialed on an outgoing call will be reported in the *Digits* field in the call event.

8.1.5 Connection Events

Connection events are associated with physical lines, so certain types of connection events can only be triggered on primary extensions. When the Server cannot determine whether an event occurred on the primary or secondary extensions, such as when the telephone goes off-hook and *ConnState* changes from *ConnState=Discon* to *ConnState=Dialtone*, then the Server assumes it occurred on the primary extension, and the event will only be generated on the primary extension. Other events, such as *ConnState=Ring* result from dialing a particular extension. This extension number is known by the NexPath Server, and the ringing event will be generated for the actual extension, whether it is primary or non-primary. To get all connections events for a given physical line, the client software must register for and monitor all extensions associated with a line. See the command “Extensions for Line” at 5.6 on page 5.

The connection event messages have the formats:

```
227 CONE "Addr=addr CallId=[callId |-1] ConnState=ConnState a=b c=d
..."
```

where a=b are tagged data pairs as specified below. Invalid or non-existent call ids are report as negative one (See “Discrepancies and Open Issues” at 9. on page 26)

The following event is sent whenever there is a change in the state of the connection.

```
227 CONE "Addr=addr CallId=callId ConnState=ConnState "
```

The following event is sent whenever this connection puts a call in the park orbit.

```
227 CONE "Addr=addr CallId=callId ConnState=ConnState Park=pkadr"
```

The following event is sent when the current call detects that there is a waiting call.

```
227 CONE "Addr=addr CallId=callId ConnState=ConnState WaitingCal-
lId=[none | callId]"
```

The following event is sent in the *Connect* state when a digit is pressed from the phone. Also, if digits are sent through the server interface (DTMF command), the event is sent.

```
227 CONE "Addr=addr CallId=callId ConnState=ConDigit Digit=digitDi-
aled "
```

The following event is sent when the first call is put on hold.

227 CONE "Addr=*addr* CallId=*callId* ConnState=*ConnState* Hold1=*callId*
Htype1=*holdType* Hcid1=*addr, cid, cname, calldate, calltime*"

The following event is sent when there was only one call on hold and it was dropped.

227 CONE "Addr=*addr* CallId=*callId* ConnState=*ConnState* Hold1=*none*"

When the second call is put on hold (pushed on to the hold stack), the following event along with an event describing the new top call on hold is sent. That is, the first call on hold is the call on the top of the hold stack. When the next call comes it, it is pushed to the second call.

227 CONE "Addr=*addr* CallId=*callId* ConnState=*ConnState* Hold2=*callId*
Htype2=*holdType* Hcid2=*addr, cid, cname, calldate, calltime* "

The following event is sent when the third call is put on hold. See above for details.

227 CONE "Addr=*addr* CallId=*callId* ConnState=*ConnState* Hold3=*callId*
Htype3=*holdType* Hcid3=*addr, cid, cname, calldate, calltime*"

The following message is the event sent when a conference call is put on hold. Since conference calls do not have a call id associated with them, but are merely a *grouping* of call ids, the call id for the hold call is reported as "conf".

227 CONE "Addr=*addr* CallId=*callId* ConnState=*ConnState* Hold1=*callId*
Htype1=ConfCall Hcid1=conf, , , "

8.1.6 Park Orbit Events

Park Orbits are a form of public hold. A call can be placed into a park orbit by one extension, and picked up without restriction at any other extension. The park orbit event message has the format below. This event is sent whenever a call is put into a park orbit or removed from one.

229 PARE "Park-*pkadr=callId*"

8.2 Requesting Status Messages

The client can request the status of all of the information about the current state of the provider, address, call, connection, or park orbit. The information returned will be in the same format as the asynchronous messages, but with a different mnemonic to indicate it was a requested status rather than an asynchronous event. Leaving out the *call id* or *ext* will not return the status of all items. The *call id* or *ext* is required and if it is missing it is an error. To determine all of the extensions or other resources on the system, the client must read the system phone book (See "Important File Locations" on page 5 in FileCommands.doc).

Commands:

```
GETS  CALL_EV  callId
GETS  ADDR_EV  ext
GETS  PROV_EV
GETS  CONN_EV  ext
GETS  PARK_EV
```

where:

PROV_EV - general provider information
ADDR_EV - information specific to the address

CALL_EV - relative to a particular call
 CONN_EV - relative to a particular address connection
 PARK_EV - relative to the park orbits
ext - a valid normal extension
callId - call id for the call

8.2.1 Provider Status

The provider status can be requested with the following command:

```
226 PROS "Mode= [day|night] OL-line= [ByPass|Deadline|Ready] ... "
```

The *OL-line= [ByPass|Deadline|Ready]* portion is included for each line in the system. The line is in the by-pass state when the system powers up and the line is in use (off-hook) through the by-pass circuit, indicating a call is in progress. The line is in the deadline state when a problem has been detected on the line (i.e., no central office line is detected as connected). Otherwise, it is reported as ready and further state information must be gathered from the other events.

8.2.2 Address Status

The status of an address is reported as follows:

```
230 ADDS "Addr=ext DND= [yes|no] Forward= [addr |none] CallWaitingEnabled= [yes|no] "
```

```
230 ADDS "Addr=vmExt PagingNum= [addr|none] NewMsgs=numMsgs OldMsgs=numMsgs Delay=3 [0..10] Separator=[# |*|<blank> CIDHeader=[on|off] TimeDateHeader=[on|off] Greeting=[standard|alternate] emailAllow=[allow|deny] emailName=email_name emailHost=host_name emailUser=user_name emailPass=**** emailLevel=[reg|short] emailAttachAudio=[header|entire] emailDeleteAfterSend=[no|yes] vmDistGroup=file_name;ext,...,ext vmDistGroup=/var/cbts/config/system_vm_groups/system_file_name; "
```

Example:

```
230 ADDS "Addr=227 PagingNum=18004758941 NewMsgs=3 OldMsgs=0 Delay=3 Separator= CIDHeader=off TimeDateHeader=on Greeting=standard emailAllow=allow emailName=5554822570@gte-messaging.com emailHost=192.88.116.24 emailUser=george emailPass=**** emailLevel=reg emailAttachAudio=header emailDeleteAfterSend=no vmDistGroup=group1.au;227,744 vmDistGroup=/var/cbts/config/system_vm_groups/somevm.au; vmDistGroup=/var/cbts/config/system_vm_groups/allvm.au; "
```

8.2.3 Call Status

The status of a call id is reported as follows:

```
224 CALS "CallId=callId Caller=ext,cid,cname,calldate,calltime Called=ext Digits=<digits> CallState=[Idle|Active|Invalid] CallType=ctype Record=[off|on] "
```

```
224 CALS "CallId=callId CallId1=callId Ext1=addr,cid,cname,calldate,calltime Digits1=<digits> CallId2=callId Ext2=addr,cid,cname,calldate,calltime Digits2=<digits> CallId3=callId Ext3=addr,cid,cname,calldate,calltime Digits3=<digits> CallId4=callId Ext4=addr,cid,cname,calldate,calltime Digits4=<digits> CallType=[Conf|Call|WaitingConf ] Record=[on|off] "
```

8.2.4 Connection Status

The connection status is reported in the following formats:

```
228 CONS "Addr=ext CallId=callId ConnState=ConnState WaitingCallId=[-1
|callId] [Hold1=none | callId] Hold1=callId Htype1=holdType
Hcid1=addr,cid,cname,calldate,calltime] [Hold2=none | callId] Hold2=callId
Htype2=holdType Hcid2=addr,cid,cname,calldate,calltime] [Hold3=none | cal-
lId] Hold3=callId Htype3=holdType hcid3=cidaddr,cid,cname,calldate,call-
time]"
```

When an extension is connected to a park orbit, it is reported as follows:

```
228 CONS "Addr=ext Park=pkadr"
```

8.2.5 Park Orbit Status

```
230 PARS "Park-pkadr=[callId | none]"
```

The extension sequence is repeated for each park orbit in the system. For instance, a park orbit status message may look like:

```
230 PARS "Park-100=none Park-101=4 Park-102=none Park-103=7"
```

All other information about a call can then be retrieved through the **CALL_EV**. The connect state will indicate `Park=callId` for the address which is "parked". The other party may be in other states.

8.3 Status and Event Message Abbreviations

The following is a description of the abbreviations and mnemonics used in the status and event messages described in this above sections.

<i>callId-</i>	call id for the call
<i>ext-</i>	a valid normal extension
<i>vmExt-</i>	a valid voicemail extension
<i>numMsgs-</i>	number of waiting voice mail messages
<i>address-</i>	Either a valid extension or a valid phone number
<i>cid-</i>	<number P O NA>
<i>holdType, ctype-</i>	

One of the following:

NoCall

NormalCall

ConfCall

WaitingXfer

WaitingConf

RingGroup

ConnState-

One of the following:

Discon
Onhook
Offhook
Ringing
Dialtone
Reqcon
Busysig
Ringback
Connect
Onhold
VmGreet
VmRecord
VmPlay
AutoAttn
CallDist
Admin
Maint
ConnDigit
ConRingback
ConDialtone
ConBusy
ConNone
ConFastBusy
ConSilence
VmwiCheck
DetectFax
WaitSock
CallScrn
CIDonCW
AnnounceN
vmExitOp

ConnDigit is sent when the switch detects that a digit has been pressed or it has processed a **DIAL** command. This will only happen while a line is in the **Connect** state. Call Progress states are reported as **ConRingback**, **ConDialtone**, **ConBusy**, **ConNone**, **ConFastBusy**, and **ConSilence**. These are only reported on outside lines while in the **Connect** state, and are Connection Events.

9. Discrepancies and Open Issues

1. Invalid or non-existent call ids may be reported as **-1** or **none** depending on the how the event was generated and which key-value pair it is associated with. Client software should accept any positive number as a valid call id, and anything else as an invalid call id.
2. If the **ext** field is left out of the **GETS CONN_EV**, the status of the last address requested is returned, rather than an error.

3. The **PLAA** command requires a filename with an absolute path name, not a relative path name.
4. Connection events can be registered on non-primary (virtual) extensions, but no events are generated. Attempting to register for a connection event on a non-primary extension should result in an error, since an event will never be triggered.
5. In many instances, the **ConnState** of an address is reported as **Disconn**, rather than **Onhook**. Client software should consider a **Disconn** state of a connection as the same as **Onhook**.
6. Conference calls do not issue **CALE** events to all call ids in the grouping (conference) when a call drops out of the conference. Software should monitor all of the call ids (**Ext1 - Ext4**) in the conference to determine when a call has dropped out of the conference.

10. Future Considerations

1. Consider tags for parameters in commands, similar to tags for variable data in double quotes
2. Consider time stamps on events.

11. Revision History

1. Beta Release: CVS Version 1.35 Wed Apr 1 09:17:29 1998
2. 1998 Release 3: CVS Version 1.36.
3. 1998 Release 4: CVS Version 1.38 1999/01/05 19:56:55
4. Release 5: CVS Version 1.50 September 25, 1999.
5. Release 6: CVS Version 1.54 March 11, 2001.
6. Release 7: CVS Version 1.58 August 29, 2003.

