

Abekas 6000 Ethernet API

September 2004

Accom, Inc.

Table of Contents

1	Introduction	1
1.1	Purpose	1
1.2	The API Server	1
1.3	Protocol Notes	1
1.3.1	Byte Order	1
1.3.2	Strings	1
1.3.3	Booleans	2
2	Abekas 6000 Terminology	3
3	Command Formatting	5
3.1	MESSAGE HEADER	5
3.2	BYTE COUNT	5
3.3	COMMAND HEADER	5
3.4	TRANSACTION ID	5
3.5	COMMAND TYPE	6
3.6	SUB-COMMAND TYPE	6
3.7	ARG TYPE	6
3.8	ARG	6
4	Reply Formatting	7
4.1	REPLY HEADER	7
4.2	REPLY MESSAGE BYTE COUNT	7
4.3	TRANSACTION ID	7
4.4	ERROR REPLY CODE	7
4.5	ERROR MESSAGE BYTE COUNT	8
4.6	ERROR MESSAGE	8
4.7	REPLY TYPE	8
4.8	REPLY TYPE SPECIFIC INFORMATION	9
5	Commands	11
5.1	Configuration Commands	11
5.1.1	ACQUIRE CHANNEL (channelMask)	11
5.1.2	FORCE RELEASE CHANNEL (channelMask)	11
5.1.3	RELEASE CHANNEL (channelMask)	12
5.2	Channel Commands	12
5.2.1	CUE CLIP (channel, clipID, blackOnCue, gotoField)	12
5.2.2	CUE NEXT CLIP (channel, clipID, blackOnCue, gotoField)	13

5.2.3	CUE PREVIOUS CLIP (channel, clipID, blackOnCue, gotoField)	14
5.2.4	GOTO FIELD (channel, gotoField, minLatency)	14
5.2.5	INSERT EDIT CLIP (channel, trackMask)	15
5.2.6	JOG CLIP (channel, relFields, minLatency)	15
5.2.7	LIVE (channel, on_off, minlatency)	16
5.2.8	NETVUE CLIP (channel, clipID, fieldToView)...	16
5.2.9	NETVUE NEXT CLIP (channel, clipID, fieldToView).....	16
5.2.10	NETVUE PREVIOUS CLIP (channel, clipID, fieldToView).....	16
5.2.11	PLAY (channel, speed, minLatency)	17
5.2.12	PRECUE CLIP	17
5.2.13	STOP (channel, minLatency).....	18
5.2.14	TEST PATTERN (channel, patternID).....	19
5.2.15	TRIM END (channel, outpoint)	19
5.3	Database Commands	19
5.3.1	ABORT COPY CLIP (toID)	19
5.3.2	ABORT MOVE CLIP (toID).....	19
5.3.3	ADD CLIP TRACKS (clipRecordSpec)	19
5.3.4	ALLOCATE CLIP (clipRecordSpec)	20
5.3.4.1	ClipID	21
5.3.4.2	Length	21
5.3.4.3	Allocation Type.....	21
5.3.4.4	Title	21
5.3.4.5	Creator	21
5.3.4.6	Project.....	21
5.3.4.7	Keywords	21
5.3.4.8	Purge Date.....	22
5.3.4.9	Source Timecode	22
5.3.4.10	Playback Mode	22
5.3.4.11	Interpolator Mode	22
5.3.4.12	Repeat Control Mode	23
5.3.4.13	Timecode Preference	23
5.3.4.14	Dropframe Preference	23
5.3.4.15	Number of Tracks.....	23
5.3.4.16	Track Type.....	23
5.3.4.17	Track Video Standard	24
5.3.4.18	Track Video Resolution.....	24
5.3.4.19	Track Key Type	24
5.3.4.20	Track Audio Standard.....	24
5.3.4.21	Audio Phase.....	24
5.3.4.22	Audio Subchannel Mask	25
5.3.5	CLIP EXISTS (clipID, hostID)	25
5.3.6	COPY CLIP (fromID, toID, update).....	25

5.3.7	COPY EXTRACT CLIP (fromID, InPoint, outPoint, toID)	25
5.3.8	DELETE CLIP (clipID, hostID, channelMask)...	26
5.3.9	FIND NEXT FREE CLIPID (clipID, hostID, inclFlag, wrapFlag)	26
5.3.10	FIND NEXT USED CLIPID (clipID, hostID, inclFlag, wrapFlag, findCueable)	26
5.3.11	FIND PREVIOUS FREE CLIPID (clipID, hostID, inclFlag, wrapFlag)	27
5.3.12	FIND PREVIOUS USED CLIPID (clipID, hostID, inclFlag, wrapFlag, findCueable)	28
5.3.13	GET CLIP INFO (clipID, hostID)	28
5.3.14	GET NEXT CLIP INFO (clipID, hostID)	28
5.3.15	GET PREV CLIP INFO (clipID, hostID)	29
5.3.16	LOCK CLIP (clipID, lock_unlock)	29
5.3.17	MODIFY CLIP ATTRIBUTES (clipID, countAttrs, <attrID,attrVal>)	29
5.3.17.1	Title	30
5.3.17.2	Creator	30
5.3.17.3	Project	30
5.3.17.4	Keywords	30
5.3.17.5	Purge Date	31
5.3.17.6	Browse Field	31
5.3.17.7	Repeat In Field	31
5.3.17.8	Repeat Out Field	31
5.3.17.9	Playback Mode	31
5.3.17.10	Interpolator Mode	32
5.3.17.11	Repeat Control Mode	32
5.3.17.12	Timecode Preference	32
5.3.17.13	Dropframe Preference	32
5.3.18	MOVE CLIP (fromID, toID)	32
5.3.19	MOVE EXTRACT CLIP (fromID, InPoint, outPoint, toID)	33
5.3.20	TRIM CLIP (clipID, markIn, duration)	33
5.3.21	GET COPY PROGRESS (copyid)	33
5.4	External VTR Commands	34
5.4.1	VTR ABORT (channel)	34
5.4.2	VTR BYPASS (channel, onoff)	34
5.4.3	VTR FAST FWD (channel)	34
5.4.4	VTR GOTO (channel, timecode)	35
5.4.5	VTR JOG (channel, direction)	35
5.4.6	VTR PLAY (channel)	35
5.4.7	VTR PLAY FOR (channel, duration)	35
5.4.8	VTR RECORD (channel, chbits)	35
5.4.9	VTR REWIND (channel)	35
5.4.10	VTR SHUTTLE (channel, speed)	35
5.4.11	VTR STILL (channel)	35

5.4.12	VTR STOP (channel).....	35
5.4.13	VTR TO CLIP (channel, clipin, vtrin, duration, clipid, chbits).....	35
5.4.14	VTR TO CLIP PVW (channel, clipin, vtrin, duration, clipid, chbits).....	35
5.4.15	VTR TO TAPE (channel, clipin, vtrin, duration, clipid, chbits).....	36
5.4.16	VTR TO TAPE PVW (channel, clipin, vtrin, duration, clipid, chbits).....	36
5.4.17	VTR VARPLAY (channel, speed).....	36
5.4.18	VTR VARPLAY FOR (channel, speed, duration)	36
5.5	Report Commands	36
5.5.1	GET REPORT	36
5.5.2	REGISTER FOR REPORT	37
5.5.3	UNREGISTER FOR REPORT	38
5.5.4	GET PARAM (paramID)	38
5.5.5	SET PARAM (paramID, paramValue)	40
6	Replies	43
6.1	REPLY NONE.....	43
6.1.1	REPLY OK	43
6.1.2	REPLY ERROR.....	43
6.2	REPLY CLIPID	43
6.3	REPLY CLIPID HOSTID	43
6.4	REPLY HOSTID	43
6.5	REPLY CLIPINFO	43
6.5.1	Host ID.....	44
6.5.2	Clip ID	45
6.5.3	Length.....	45
6.5.4	Allocation Type	45
6.5.5	Title	45
6.5.6	Creator	45
6.5.7	Project	45
6.5.8	Keywords	46
6.5.9	Timecode Fields	46
6.5.10	Timecode Preference.....	46
6.5.11	Creation Date.....	47
6.5.12	Last Access Date	47
6.5.13	Purge Date	47
6.5.14	Lock State	47
6.5.15	Browse Field	48
6.5.16	Repeat In Field	48
6.5.17	Repeat Out Field	48
6.5.18	Playback Mode.....	48
6.5.19	Interpolator Mode	48
6.5.20	Repeat Control Mode.....	48

6.5.21	Number of Tracks	49
6.5.22	Track Number	49
6.5.23	Track Type	49
6.5.24	Track Audio Standard	50
6.5.25	Track Audio Phase	50
6.5.26	Track Audio Subchannel Mask	50
6.5.27	Track Video Standard	51
6.5.28	Track Video Resolution	51
6.5.29	Track Key Type	51
6.6	REPLY CHANNEL ATTRIBUTES	51
6.6.1	Interpolator Mode	52
6.6.2	Output Mode	52
6.6.3	Loop Mode	52
6.6.4	Timecode Mode	52
6.6.5	Timecode Offset	52
6.6.6	Dropframe Mode	52
6.7	REPLY REPORT CHANNEL TIMECODE	52
6.8	REPLY REPORT CHANNEL STATUS	53
6.9	REPLY PARAM	53
6.10	REPLY COPY START	53
6.11	REPLY COPY PROGRESS	53
6.12	REPLY COPYID NOT FOUND	54
6.13	REPLY COPY DONE OK	54
6.14	REPLY COPY DONE ERROR	54
7	Examples	57
7.1	Configuration Commands	57
7.1.1	ACQUIRE CHANNEL	57
7.1.2	RELEASE CHANNEL	57
7.1.3	FORCE RELEASE CHANNEL	58
7.2	Channel Commands	58
7.2.1	RECORD CLIP	58
7.3	Database Commands	59
7.3.1	ABORT COPY CLIP	59
7.3.2	ABORT MOVE CLIP	60
7.3.3	CLIP EXISTS	60
7.3.4	COPY CLIP	61
7.3.5	DELETE CLIP	61
7.3.6	FIND NEXT FREE CLIPID	62
7.3.7	GET CLIP INFO	63
7.3.8	MODIFY CLIP ATTRIBUTES	64
7.4	Report Commands	65
7.4.1	GET REPORT	65
7.4.1.1	Get Channel Status	65
7.4.1.2	Get Channel Timecode	66
7.4.2	REGISTER FOR REPORT	67
7.4.2.1	Register For Channel Status	67

7.4.2.2	Register For Channel Timecode	68
7.4.2.3	Register For Clip Change Notification ..	69
8	Revision History	71
8.1	Version 3.0	71
8.2	Version 4.0	71
8.3	Version 4.0.3	71
8.4	Version 4.5	71
8.5	Version Unknown	72

1 Introduction

This manual describes the Abekas 6000 ethernet control protocol as implemented in Version 4.5BETA of the system software.

Please note that this document is only a preliminary description of a protocol that is still under development. Therefore, any or all parts of the specifications described here are subject to change.

1.1 Purpose

The Abekas 6000 contains a command server which allows the Abekas 6000 to be controlled by remote devices on a TCP/IP Network. This document provides details of the control protocol (API) which will allow you to write a client program.

1.2 The API Server

When powered up, the Abekas 6000 automatically starts a server on TCP port 8911. The Server supports 8 simultaneous client connections.

Details of how to configure the IP Address of your Abekas 6000 can be found in the Abekas 6000 Installation & Maintenance Manual, which is shipped with each system. This manual may also be downloaded as a PDF document from the Accom web site, under the Abekas 6000 product section. Please visit <http://www.accom.com> to obtain a free copy of this manual.

In order to connect and control our server, you will first need to know how to create and operate a TCP connection on your platform. How to setup such a connection varies depending on what Operating System you are using, and is therefore beyond the scope of this document.

1.3 Protocol Notes

1.3.1 Byte Order

When multibyte values, such as 32 bit integers, are sent by the protocol, the values are sent in Network Byte Order. This is sometimes known as Big Endian order. In this ordering, the bytes that make up the value are sent in order from the most significant byte to the least significant byte.

1.3.2 Strings

ASCII strings are used in a number of places in this protocol. Often, these strings of characters are preceded by a byte count. This may lead you to the mistaken impression that the strings can be any arbitrary sequence of bytes. The Abekas 6000 does not tolerate

NUL characters in strings (a character whose binary value is zero). In theory, we do tolerate all other eight-bit values, although the control panel will only display the standard ASCII characters correctly.

1.3.3 Booleans

Boolean values in the protocol are expressed in the tradition of the C Programming Language. 0 is defined to mean **false**. Non-zero values are defined to mean **true**. It is good practice to prefer 1 as **true**.

2 Abekas 6000 Terminology

For the purposes of this document it may be helpful to define some terms which have special meaning in our system.

- ‘Clip’ A set of recorded tracks of video, key and/or audio intended to be operated upon as a group.
- ‘ClipID’ The primary key into the Clip Database. It is a number between 1 and 9999999, and each number may only be used once throughout the entire network
- ‘Clip Database’
 A collection of information about recorded clips, distributed across all the nodes on the network.
- ‘Command’ A request sent from the client program to the Abekas 6000.
- ‘Field’ One half of a video image. The smallest addressable increment of a clip on Abekas 6000.
- ‘Frame’ Two fields, or a complete image.
- ‘ListID’ The primary key into the Playlist Database. It is a number between 1 and 9999999, and each number may only be used once throughout the entire network.
- ‘Node’ An individual Abekas 6000 server chassis.
- ‘Playlist’ A script for running a series of clips, including duration of transitions and channels to output the clips. Playlists are maintained in a Playlist Database.
- ‘Reply’ A reply sent by the Abekas 6000 to the client program as a result of a command sent from the client program to the Abekas 6000.
- ‘Report’ An asynchronous message sent from the Abekas 6000 to the client program, which is not necessarily associated with a command, and provides status or state information.
- ‘Still’ A clip with duration of 1 frame.

3 Command Formatting

Commands to the OECF server are sent as packets within the TCP bytestream. The format of command packet is:

```

<MESSAGE HEADER>
<BYTE COUNT>
<COMMAND HEADER>
<TRANSACTION ID>
<COMMAND TYPE>
<SUB COMMAND TYPE>
<ARG #1 TYPE><ARG #1>
<ARG #2 TYPE><ARG #2>
...
<ARG #N TYPE><ARG #N>

```

3.1 MESSAGE HEADER

The message header provides a marker for the beginning of a command packet. This header is present simply to add redundancy to the protocol in order to help verify correctness.

‘Number of Bytes’

OECP_MSG_HEADER_REPEAT (currently 3) in oecp_package.h

‘Content’ OECP_MSG_HEADER_FLAG (currently 0xe7 0xe7 0xe7) in oecp_package.h

‘Purpose’ To mark the beginning of a message

3.2 BYTE COUNT

‘Number of Bytes’

2

‘Content’ The length of the rest of the message in bytes (excluding MESSAGE HEADER and BYTE COUNT)

‘Purpose’ To let the reader (server) know how many bytes of data to expect.

‘Range’ The legal range for BYTE COUNT is 0-1024 (in decimal)

3.3 COMMAND HEADER

‘Number of Bytes’

OECP_CMD_HEADER_REPEAT (currently : 3) in oecp_package.h

‘Content’ OECP_CMD_HEADER_FLAG (currently : 0xda 0xda 0xda) in oecp_package.h

‘Purpose’ To mark the beginning of a command in a message stream.

3.4 TRANSACTION ID

‘Number of Bytes’

1 + 4 == 5

‘content’ One byte OECP_EXT_TRANSID_TYPE, followed by a four-byte client selected transaction number.

‘Purpose’ Responses to this command will be tagged with the same Transaction ID to help you identify which responses were provoked by which command. You may choose your transaction IDs in any way that suits your needs. The OECP server does not try to divine any meaning from your choice of transaction IDs.

3.5 COMMAND TYPE

‘Number of Bytes’

1 + 1 == 2

‘Content’ The Command Type defined in header file "oecp_cmd_types.h"

‘Purpose’ To specify the main command type

‘Note’ The 1 byte of COMMAND TYPE is preceded by one byte of identifier - "OECP_COMMAND_ID_TYPE"

3.6 SUB-COMMAND TYPE

‘Number of Bytes’

1 + 1 == 2

‘Content’ The Sub-Command Type defined in "oecp_XXX_cmds.h"

‘Purpose’ To specify the sub-command type

‘Note’ The 1 byte of SUB COMMAND TYPE is preceded by one byte of identifier - "OECP_SUB_COMMAND_ID_TYPE"

3.7 ARG TYPE

‘Number of Bytes’

1

‘Content’ OECP argument type defined in "oecp_arg_types.h"

‘Purpose’ To indicate the type of argument that follows

3.8 ARG

‘Number of Bytes’

Variable, depending on the type of the argument.

‘Content’ Argument for the command

4 Reply Formatting

Replies from the OECF server are sent as packets within the TCP bytestream. The format of the reply packet is:

```

    <REPLY HEADER>
    <REPLY MESSAGE BYTE COUNT>
    <TRANSACTION ID>
    <ERROR REPLY CODE>
    <ERROR MESSAGE BYTE COUNT>
    <ERROR MESSAGE>
    <REPLY TYPE>
    <REPLY TYPE SPECIFIC INFORMATION>

```

4.1 REPLY HEADER

‘Number of Bytes’

OECP_REPLY_HEADER_REPEAT (currently : 3) in oecp-package.h

‘Content’ OECP REPLY HEADER (currently : 0xd6 0xd6 0xd6) in oecp-package.h

‘Purpose’ To mark the beginning of a reply.

4.2 REPLY MESSAGE BYTE COUNT

‘Number of Bytes’

2

‘Content’ The length of the reply message excluding REPLY HEADER and REPLY MESSAGE BYTE COUNT.

4.3 TRANSACTION ID

‘Number of Bytes’

1 + 4

‘Content’ The original TRANSACTION ID that was sent with the command that yielded the reply.

‘Purpose’ To identify uniquely the command to which the reply belongs.

4.4 ERROR REPLY CODE

‘Number of Bytes’

1 + 2

‘Content’ The byte OECP_COMMAND_REPLY_ERC_TYPE followed by a two byte error code

‘Purpose’ To provide a meaningful error code that can be interpreted by software.

Currently, the following error codes are defined

‘OECF_ERROR_OK’

No error occurred.

‘OECF_ERROR_MISC’

An non specific error occurred.

In the future, we will define other error codes which may be meaningfully interpreted by the client.

4.5 ERROR MESSAGE BYTE COUNT

‘Number of Bytes’

2

‘Content’ The length of the ERROR MESSAGE in bytes.

‘Purpose’ To let the client know how many bytes of error message to expect in the reply.

4.6 ERROR MESSAGE

‘Number of Bytes’

Provided in ERROR MESSAGE BYTE COUNT field.

‘Content’ An ASCII message string which is either OK or is an error message resulting from the command operation.

‘Purpose’ To let the client know whether or not the command succeeded, and provide a human-readable message string that might explain the any failure.

‘Note’ Error messages are intended to be used by humans, not programs. With the exception of the error string “OK,” client programs must not attempt to divine meaning from error message strings. The message text issued by the Abekas 6000 may be altered in each new release of software, so do not rely on error text strings in your client.

4.7 REPLY TYPE

Two bytes which tells the receiver the format of the remainder of the reply. Possible values include

```
OECF_COMMAND_REPLY_TYPE OECF_CMD_REPLY_NONE
OECF_COMMAND_REPLY_TYPE OECF_CMD_REPLY_CLIPID
OECF_COMMAND_REPLY_TYPE OECF_CMD_REPLY_HOSTID
OECF_COMMAND_REPLY_TYPE OECF_CMD_REPLY_CLIPID_HOSTID
OECF_COMMAND_REPLY_TYPE OECF_CMD_REPLY_CLIPINFO
OECF_COMMAND_REPLY_TYPE OECF_CMD_REPLY_CHNL_ATTRS
```

```
OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_REPORT_CHNL_STATUS
OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_REPORT_CHNL_TIMECODE
OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_PARAM
OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_COPY_START
OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_COPY_PROGRESS
OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_COPYID_NOT_FOUND
OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_COPY_DONE_OK
OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_COPY_DONE_ERROR
```

4.8 REPLY TYPE SPECIFIC INFORMATION

Except `OECP_CMD_REPLY_NONE`, each `REPLY TYPE` carries some additional information as a result of a command. The exact format of this field is provided in the documentation for each of the particular reply types (see [Chapter 6 \[Replies\]](#), page 43).

5 Commands

What follows is a list of all commands supported on the Abekas 6000, and a description of the arguments to each command.

The possible replies that the Abekas 6000 may send in response to each command is given. Because the `REPLY ERROR` (see [Section 6.1.2 \[REPLY ERROR\]](#), page 43) reply may be issued in response to *any* command, we will not mention it in the description of each command.

Unless otherwise specified, scalar arguments to commands are expressed as a one byte type value followed by the 32 bit scalar value.

5.1 Configuration Commands

On the Abekas 6000, only one control interface may affect the input or output of a channel at a time. In order to ensure that only one interface sends I/O commands to the channel at a given time, the Abekas 6000 keeps track of which interface has *acquired* each channel.

A controlling device must acquire a channel before sending I/O commands to that channel. When the controlling device no longer is interested in the channel, it should release the channel.

5.1.1 ACQUIRE CHANNEL (`channelMask`)

Make this client the owner of channel.

Sending the `ACQUIRE CHANNEL` command is a request to the Abekas 6000 that I/O control rights to a given channel be assigned to the client. The client must do this prior to sending any commands that would affect input or output such as `cue`, `play`, or `stop`.

All channels in `channelMask` must be acquirable, or the whole command fails. For example, a channel cannot be acquired if another client or agent has already acquired the channel.

`'channelMask'`

The set of channels to acquire. If bit 0 is set, will acquire channel A, if bit 1 is set, will acquire channel B, and so forth.

If this command succeeds, `REPLY OK` (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.1.2 FORCE RELEASE CHANNEL (`channelMask`)

Force the current owner of channel to release ownership.

If a control panel in another building has acquired a channel, it is a pain to have to take a walk just to go make that control panel release the acquired channel. The Force Release Channel command rips that channel away from whoever currently owns the channel. This functionality is also available on the hardware control panel.

For obvious reasons, this command should be used with considerable care. If your application mucks up a channel that was being used to broadcast a segment to air, your customer will be deeply upset. This command should only be used with the explicit approval (via a dialog box, prompt, or explicit configuration) of a human operator.

'channelMask'

The set of channels to release. If bit 0 is set, will forcibly release channel A, if bit 1 is set, will forcible release channel B, and so forth.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.1.3 RELEASE CHANNEL (channelMask)

Release ownership of channel.

Under normal circumstances, a client should release a channel that it no longer needs in order to allow that channel to be acquired and used for other purposes.

All channels in `channelMask` must be releasable, or the whole command fails. For example, if you try to release a channel you do not own, the command fails.

'channelMask'

The set of channels to force release. If bit 0 is set, will release channel A, if bit 1 is set, will release channel B, and so forth.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.2 Channel Commands

5.2.1 CUE CLIP (channel, clipID, blackOnCue, gotoField)

Cue a clip on a channel.

'channel' The channel number to cue on. 1 is Channel A, 2 is channel B, and so forth.

'clipID' The clip to cue.

'blackOnCue'

Optional. Defaults to **false**.

This boolean field determines what is output when the clip is cued. If true, the output will remain black until a play command is sent. If false, the `gotoField` will be output immediately.

'repeat' *Optional.* Defaults to `OECP_CH_REPEAT_OFF`.

This field determines the repeat mode of the clip when the clip is cued.

```

‘OECF_CH_REPEAT_OFF’
‘OECF_CH_REPEAT_LOOP’
‘OECF_CH_REPEAT_LOOP_TO’
‘OECF_CH_REPEAT_PINGPONG’
‘OECF_CH_REPEAT_PINGPONG_TO’

```

‘gotoField’

Optional. Defaults to 0.

The field of the clip to display after cueing. Zero means the first field, Two means the start of the second frame. If `gotoField` is greater than the length of the clip, the clip will be cued to the end of the clip.

If the parameter `useTc` is `true`, then the value of this field is interpreted differently. This `useTc` is assumed to be a timecode to cue to. The base timecode of the clip is determined by the timecode attributes of the clip.

‘endField’

Optional. Defaults to 0.

You can cue just a portion of a clip, instead of the entire clip. To do this, use `endField` to specify the length you want the cued clip to be. Set `endField` to 0 if you want the entire clip to be cued. If you set `endField` to be a length that is greater than the length of the clip, the server will behave as if you set `endField` to 0.

‘useTc’

Optional. Defaults to `false`.

If `true`, this parameter indicates that the `gotoField` shall be interpreted as a timecode. See the documentation for the parameter `gotoField` above for more information about this.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\], page 43](#)) will be sent from the server.

5.2.2 CUE NEXT CLIP (channel, clipID, blackOnCue, gotoField)

Cue the next `clipID` on this node on `channel` at `gotoField`. Autowraps at end of range.

‘channel’ The channel to cue on. 1 is Channel A, 2 is channel B, and so forth.

‘clipID’ The clip to start the search from.

‘blackOnCue’

This boolean field determines what is output when the clip is cued. If `true`, the output will remain black until a play command is sent. If `false`, the `gotoField` will be output immediately.

‘gotoField’

The field of the clip to display after cueing. Zero means the first field, Two means the start of the second frame. If `gotoField` is greater than the length of the clip, the clip will be cued to the end of the clip.

‘endField’

You can cue just a portion of a clip, instead of the entire clip. To do this, use `endField` to specify the length you want the cued clip to be. Set `endField` to 0 if you want the entire clip to be cued. If you set `endField` to be a length that is greater than the length of the clip, the server will behave as if you set `endField` to 0.

If this command succeeds, REPLY CLIPID (see [Section 6.2 \[REPLY CLIPID\]](#), page 43) will be sent from the server to let the client know what clip was cued.

5.2.3 CUE PREVIOUS CLIP (`channel`, `clipID`, `blackOnCue`, `gotoField`)

Cue the previous `clipID` on this node on `channel` at `gotoField`. Autowraps at end of range.

‘channel’ The channel to cue on. 1 is Channel A, 2 is channel B, and so forth.

‘clipID’ The clip to start the search from.

‘blackOnCue’

This boolean field determines what is output when the clip is cued. If true, the output will remain black until a play command is sent. If false, the `gotoField` will be output immediately.

‘gotoField’

The field of the clip to display after cueing. Zero means the first field, Two means the start of the second frame. If `gotoField` is greater than the length of the clip, the clip will be cued to the end of the clip.

‘endField’

You can cue just a portion of a clip, instead of the entire clip. To do this, use `endField` to specify the length you want the cued clip to be. Set `endField` to 0 if you want the entire clip to be cued. If you set `endField` to be a length that is greater than the length of the clip, the server will behave as if you set `endField` to 0.

If this command succeeds, REPLY CLIPID (see [Section 6.2 \[REPLY CLIPID\]](#), page 43) will be sent from the server to let the client know what clip was cued.

5.2.4 GOTO FIELD (`channel`, `gotoField`, `minLatency`)

Jump to a different location in the currently cued clip. After the jump, the clip will be paused.

‘channel’ The channel to cue on. 1 is Channel A, 2 is channel B, and so forth.

‘gotoField’

The field of the clip to jump to. Zero means the first field, Two means the start of the second frame.

‘minLatency’

This boolean field specifies that the command should be executed as quickly as possible. Minimizing latency is useful in order to make the machine more responsive when under human control. This function grants the channel more bandwidth than it normally receives for a short burst of time. Continuous use of this flag may cause the machine to have insufficient bandwidth and fail to playout correctly even on channels which are not under client control. Therefore it is important to use this flag only for commands that are sent by a human operator, and never use this command when the machine is under the control of an autonomous client.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\], page 43](#)) will be sent from the server.

5.2.5 INSERT EDIT CLIP (channel, trackMask)

This command begins an insert edit into the clip cued on channel **channel**. **trackMask** is interpreted as a bitmask specifying which tracks should be edited.

‘channel’ The channel to send the edit command to. 1 is Channel A, 2 is channel B, and so forth.

‘trackMask’

A bitmask describing which channels to edit into, loosely based on the edit bits in the Sony VTR Serial Control Protocol.

‘BIT0’	Audio 1.
‘BIT1’	Audio 2.
‘BIT2’	Audio 3.
‘BIT3’	Audio 4.
‘BIT4’	Video
‘BIT5’	Key

Note: At this time, the Abekas 6000 does not support split audio editing. If any of the audio edit bits are set, then all four audio tracks will be edited.

Ending the edit is accomplished by sending a **trackMask** which is all zero.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\], page 43](#)) will be sent from the server.

5.2.6 JOG CLIP (channel, relFields, minLatency)

Jump to a different location in the currently cued clip relative to the current location.

‘channel’ The channel to cue on. 1 is Channel A, 2 is channel B, and so forth.

‘relFields’

The number of fields to move relative to the current position. A 2 means to move one frame forwards in time. -60 means to jump one second in reverse (in 525/60).

‘minLatency’

This boolean field specifies that the command should be executed as quickly as possible. Minimizing latency is useful in order to make the machine more responsive when under human control. This function grants the channel more bandwidth than it normally receives for a short burst of time. Continuous use of this flag may cause the machine to have insufficient bandwidth and fail to playout correctly even on channels which are not under client control. Therefore it is important to use this flag only for commands that are sent by a human operator, and never use this command when the machine is under the control of an autonomous client.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.2.7 LIVE (channel, on_off, minlatency)

Put input video on channel. More commonly known as “Bypass.”

`minlatency` is *not* implemented.

5.2.8 NETVUE CLIP (channel, clipID, fieldToView)

If clipID is on the local node, cue clipID on channel at fieldtoView. Else display the clip’s frame/field at fieldToView on channel. Frame/Field display depends on channel output mode.

If this command succeeds, REPLY HOSTID (see [Section 6.4 \[REPLY HOSTID\]](#), page 43) will be sent from the server.

5.2.9 NETVUE NEXT CLIP (channel, clipID, fieldToView)

Find the next used clipID on the network. Find next used autowraps at end of range. If the next clip is on the local node, cue nextClipId on channel at fieldToView. Else display the next clip’s frame/field at fieldToView on channel. Frame/Field display depends on channel output mode.

If this command succeeds, REPLY CLIPID HOSTID (see [Section 6.3 \[REPLY CLIPID HOSTID\]](#), page 43) will be sent from the server.

5.2.10 NETVUE PREVIOUS CLIP (channel, clipID, fieldToView)

Find the previous used clipID on the network. Find previous used autowraps at beginning of range. If the previous clip is on the local node, cue prevClipID on channel at fieldToView. Else display prevClipID’s frame/field at fieldToView on channel. Frame/Field display depends on channel output mode.

If this command succeeds, `REPLY CLIPID HOSTID` (see [Section 6.3 \[REPLY CLIPID HOSTID\]](#), page 43) will be sent from the server.

5.2.11 PLAY (channel, speed, minLatency)

Play the clip on channel at speed.

`'channel'` The channel whose clip should begin to play back. 1 is Channel A, 2 is channel B, and so forth.

`'speed'` The speed at which to play. Speed is expressed in 16:16 fixed point notation. For example 0x10000 is one times play speed.

`'minLatency'`
This boolean field specifies that the command should be executed as quickly as possible. Minimizing latency is useful in order to make the machine more responsive when under human control. This function grants the channel more bandwidth than it normally receives for a short burst of time. Continuous use of this flag may cause the machine to have insufficient bandwidth and fail to playout correctly even on channels which are not under client control. Therefore it is important to use this flag only for commands that are sent by a human operator, and never use this command when the machine is under the control of an autonomous client.

If this command succeeds, `REPLY OK` (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.2.12 PRECUE CLIP

It is desirable to allow clips to be played out back-to-back under the direction of the client software. In order to play a new clip out immediately following a previous clip, the Abekas 6000 will need advance notice of what clip is expected to follow in order than it can fetch the necessary frames off disk in advance.

In the Abekas 6000, the precue command provides the mechanism for this advance notification. To precue a clip means to prepare it for output. The cut between the old clip and the new clip can be activated either automatically or manually in a number of ways.

`'channel'` The channel to precue the clip on. 1 is Channel A, 2 is channel B, and so forth.

`'clipid'` The clip to precue.

`'precueAction'`
This field specifies when the cut between the old clip and the new clip shall take place.

`'OECP_PRECUE_ACTION_WAIT_FOR_EXPLICIT_SWITCH'`

This option specifies that once the new clip is precued, an explicit `PRECUE ACTIVATE` command will be sent by the client to switch to the new clip. Unlike precue, `PRECUE ACTIVATE` is pretty much a realtime command.

`'OECP_PRECUE_ACTION_SWITCH_WHEN_PREVIOUS_STOPS'`

This option specifies that when the old clip stops playing, a cut to the new clip will immediately occur.

`'OECP_PRECUE_ACTION_SWITCH_IMMEDIATELY'`

This option specifies that the cut will occur immediately after the precue completes.

`'OECP_PRECUE_ACTION_SWITCH_AT_END_OF_CLIP'`

This option specified that when the old clip reaches the end of clip, a cut to the new clip will immediately occur.

`'precueUsePreviousSpeed'`

When `true`, this boolean argument specifies that after the cut to the new clip, the new clip shall play at the same speed that the old clip was playing at.

`'speed'`

The speed at which the new clip shall immediately begin playing at once the cut to the new clip occurs. This field is irrelevant if `precueUsePreviousSpeed` is `true`. Of course, you must still *send* this command even if it has no effect.

`'gotoField'`

The first field of the new clip that shall be output after the cut to the new clip.

`'endField'`

You can precue just a portion of a clip, instead of the entire clip. To do this, use `endField` to specify the length you want the precued clip to be. Set `endField` to 0 if you want the entire clip to be precued. If you set `endField` to be a length that is greater than the length of the clip, the server will behave as if you set `endField` to 0.

`'preview'`

If `true`, this boolean field specifies that the cut to the new clip should happen immediately if the previous clip is stopped at the time the precue occurs. If this happens, after the cut, the new clip will also be stopped. (I can't think of any good reason to set this to `true` from the OECP protocol. This argument is exists only to support the Louth Serial Protocol).

5.2.13 STOP (channel, minLatency)

Stop the playout of clip on channel (speed = 0.00).

`'channel'`

The channel whose clip should stop playing back. 1 is Channel A, 2 is channel B, and so forth.

`'minLatency'`

This boolean field specifies that the command should be executed as quickly as possible. Minimizing latency is useful in order to make the machine more responsive when under human control. This function grants the channel more bandwidth than it normally receives for a short burst of time. Continuous use of this flag may cause the machine to have insufficient bandwidth and fail to playout correctly even on channels which are not under client control. Therefore it is important to use this flag only for commands that are sent by a human

operator, and never use this command when the machine is under the control of an autonomous client.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.2.14 TEST PATTERN (channel, patternID)

Cue a test pattern, identified by `patternID`, in `channel`. The Abekas 6000 provides a set of test patterns that permanently reside on the system. Pattern IDs are in `oecp_channel_cmds.h`

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.2.15 TRIM END (channel, outpoint)

Change the accessible length of the currently cued clip. If the outpoint is zero, or if the output is greater than the length of the currently cued clip, the whole clip will be accessible.

5.3 Database Commands

5.3.1 ABORT COPY CLIP (toID)

Abort a clip copy previously initiated by this client.

‘toID’ The `toID` specified as the destination clipID in the copy command.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.3.2 ABORT MOVE CLIP (toID)

Abort a clip move previously initiated by this client.

‘toID’ The `toID` specified as the destination clipID in the move command.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.3.3 ADD CLIP TRACKS (clipRecordSpec)

Add tracks to an existing clip, specified by the `clipRecordSpec` trackSpecs.

For a description of `clipRecordSpec`, see [Section 5.3.4 \[ALLOCATE CLIP\]](#), page 20.

If a track is specified in `clipRecordSpec` that already exists on the clip, that track specifier is ignored.

Only the clipid and track specification fields of `clipRecordSpec` are relevant to this command. All other fields such as title string and so forth are ignored.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.3.4 ALLOCATE CLIP (`clipRecordSpec`)

Create a new clip, specified by `clipRecordSpec`, on node.

The format of record spec follows. Unlike most structures in the protocol, the fields of this structure may appear in any order, and you are not required to send every field. Just send the ones that are mandatory, plus the fields that are interesting to your application. Reasonable defaults will be provided for all fields that you do not specify.

The field `OECP_CLIP_ID` is mandatory. If you are allocating at least one track, then `OECP_CLIP_NFIELDS` and `OECP_CLIP_TRACKS` are also mandatory. For each track you specify, `OECP_CLIP_TRACK_NUMBER` and `OECP_CLIP_TRACK_TYPE` are required.

In order to separate the start of one track specifier from the end of the previous track specifier, you must provide the field `OECP_CLIP_TRACK_NUMBER` as the first field of each track specifier.

<code>DB_RECORD_SPEC</code>	1 byte
<code>OECP_CLIP_ID clipid</code>	5 bytes
<code>OECP_CLIP_NFIELDS length</code>	5
<code>OECP_CLIP_ALLOCTYPE alloctype</code>	2
<code>OECP_CLIP_TITLE title</code>	2 + strlen
<code>OECP_CLIP_CREATOR creator</code>	2 + strlen
<code>OECP_CLIP_PROJECT project</code>	2 + strlen
<code>OECP_CLIP_KEYWORDS keywords</code>	2 + strlen
<code>OECP_CLIP_PURGE_DATE purgedate</code>	5
<code>OECP_CLIP_SRC_TC tc</code>	5
<code>OECP_CLIP_PLAYBACK_MODE mode</code>	2
<code>OECP_CLIP_INTERP_MODE mode</code>	2
<code>OECP_CLIP_REPEAT_CTRL_MODE mode</code>	2
<code>OECP_CLIP_TC_PREF mode</code>	2
<code>OECP_CLIP_SOURCE_DROPFRAME_PREFERENCE pref</code>	2
<code>OECP_CLIP_TOD_DROPFRAME_PREFERENCE pref</code>	2
<code>OECP_CLIP_VITC_DROPFRAME_PREFERENCE pref</code>	2
<code>OECP_CLIP_TRACKS numtracks</code>	2
For each track:	
<code>OECP_CLIP_TRACK_NUMBER trackNum</code>	2
<code>OECP_CLIP_TRACK_TYPE type</code>	2
If track is a video or key track:	
<code>OECP_CLIP_TRACK_VIDEO_STANDARD std</code>	2
<code>OECP_CLIP_TRACK_VIDEO_RESOLUTION res</code>	2
If track is an audio track:	
<code>OECP_CLIP_TRACK_AUDIO_STANDARD std</code>	2

OECP_CLIP_TRACK_PHASE phase	5
OECP_CLIP_TRACK_SUB_CHANNEL_MASK mask	2
If track is a key track:	
OECP_CLIP_TRACK_KEY_TYPE type	2

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.3.4.1 ClipID

The clip identifier of the new clip you want created.

5.3.4.2 Length

The length of the clip you are creating in fields.

5.3.4.3 Allocation Type

If the clip has zero tracks, set this field to ALLOCTYPE_ALLOCHDR. If you have one or more tracks in the allocation, set this field to ALLOCTYPE_ALLOCALL.

5.3.4.4 Title

This field provides a title for the clip.

This field contains a single byte OECP_CLIP_TITLE, a single byte string length which must be no more than CLIP_TITLE_LEN, and the title string text.

5.3.4.5 Creator

This field describes the person or entity that created this clip.

This field contains a single byte OECP_CLIP_CREATOR, a single byte string length which must be no more than CLIP_CREATOR_LEN, and the creator string text.

5.3.4.6 Project

This field describes the project that the clip is a part of.

This field contains a single byte OECP_CLIP_PROJECT, a single byte string length which must be no more than CLIP_PROJECT_LEN, and the project string text.

5.3.4.7 Keywords

Currently keywords are broken. Please send OECP_CLIP_KEYWORDS 6 0 0 0 0 0 0.

A clip may have up to CLIP_MAX_KEYWORDS keywords associated with it.

This field contains a single byte `OECP_CLIP_KEYWORDS`, a single byte keywords string length, and then the keywords string text.

The keywords string text contains all `CLIP_MAX_KEYWORDS` keywords, each of which is terminated by a zero byte. No single keyword may be longer than `CLIP_KEYWORD_LEN`.

If there are no keywords yet assigned to the clip, the keywords string will be `CLIP_MAX_KEYWORDS` long, and each character in the string will be a zero byte.

For example, if there are two keywords “key1” and “key2,” the keywords field would look like this.

```

OECP_CLIP_KEYWORDS
14
'k' 'e' 'y' '1' 0
'k' 'e' 'y' '2' 0
0
0
0
0

```

5.3.4.8 Purge Date

The time when the clip is destined to be eliminated. When a clip is created, the Abekas 6000 initializes this field to zero.

The Abekas 6000 currently preserves this field, but never uses it.

The `date` is a four byte unsigned integer. The `date` is the number of seconds since midnight January 1, 1970 as is traditional in the C Programming Language.

5.3.4.9 Source Timecode

5.3.4.10 Playback Mode

The playback mode of the clip. Possible values include:

‘`PLAYBACK_MODE_FIELD`’

The clip is composed of field based material. The Abekas 6000 can playback in variable speed field mode despite the fact that the underlying DV compressed material is stored and edited as whole frames.

‘`PLAYBACK_MODE_FRAME_F12`’

The clip is composed of frames where matching sets of fields are a field1 and the following field2.

‘`PLAYBACK_MODE_FRAME_F21`’

Not used at present.

‘`PLAYBACK_MODE_FRAME`’

Not used at present.

5.3.4.11 Interpolator Mode

The interpolator mode of the clip. This is only used during field mode playback. Possible values include:

`'INTERPOLATOR_MODE_ON'`

The interpolator should be used during field mode variable playback.

`'INTERPOLATOR_MODE_OFF'`

The interpolator should never be used.

5.3.4.12 Repeat Control Mode

This field controls how the clip should loop when played back.

`'REPEAT_CONTROL_OFF'`

Clip should play linearly straight through the clip exactly once.

`'REPEAT_CONTROL_LOOP'`

Clip should loop back to the start of the clip after passing past the end of the clip.

`'REPEAT_CONTROL_PINGPONG'`

Clip should play backwards from the end of the clip after passing past the end of the clip. Clip should play forwards from the beginning of the clip after passing past the beginning of the clip.

`'REPEAT_CONTROL_LOOPTO'`

Clip should loop back to the repeat-in field whenever playback passes the repeat-out field.

`'REPEAT_CONTROL_PINGPONGTO'`

Clip should play backwards anytime it passes the repeat-out point while playing forwards. Clip should play forwards anytime it passes the repeat-in point while playing backwards.

5.3.4.13 Timecode Preference

5.3.4.14 Dropframe Preference

5.3.4.15 Number of Tracks

The number of tracks you want allocated.

5.3.4.16 Track Type

The track type of this track. Possible values on the Abekas 6000 are:

`'TRACK_TYPE_DV25_KEY'`

A key track recorded in the 25Mb/s DV compression standard.

`'TRACK_TYPE_DV25_VIDEO'`

A video track recorded in the 25Mb/s DV compression standard.

`'TRACK_TYPE_DV50_VIDEO'`

A video track recorded in the 50Mb/s DV compression standard.

`'TRACK_TYPE_DV50_KEY'`

A key track recorded in the 50Mb/s DV compression standard.

`'TRACK_TYPE_MPEG25_KEY'`

A key track recorded as 25Mb/s I-Frame only MPEG2.

`'TRACK_TYPE_MPEG25_VIDEO'`

A video track recorded as 25Mb/s I-Frame only MPEG2.

`'TRACK_TYPE_MPEG50_VIDEO'`

A video track recorded as 50Mb/s I-Frame only MPEG2.

`'TRACK_TYPE_MPEG50_KEY'`

A key track recorded as 50Mb/s I-Frame only MPEG2.

`'TRACK_TYPE_DV_AUDIO'`

The audio format used on the Abekas 6000.

5.3.4.17 Track Video Standard

A clip may contain tracks that are playable only in 525/60, and also tracks that are playable only in 625/50. This field describes the recording frequency of this track

`'VIDEO_STANDARD_FPS60'`

525/60

`'VIDEO_STANDARD_FPS50'`

625/50

5.3.4.18 Track Video Resolution

Set this field to `VIDEO_BITS_PER_PIXEL_8`.

5.3.4.19 Track Key Type

Set this field to `KEY_TYPE_FULL_BANDWIDTH`.

5.3.4.20 Track Audio Standard

Set this field to `AUDIO_STANDARD_AES`.

5.3.4.21 Audio Phase

Set this field to 0.

5.3.4.22 Audio Subchannel Mask

Set this to `0x0f`.

5.3.5 CLIP EXISTS (clipID, hostID)

Determine if a clipID exists.

`'clipID'` The clipID of the clip you are trying to locate.

`'hostID'` The IP Address of the node you want check on to see if clipID exists. Setting hostID to all zeroes will cause the search to be performed netwide.

If this command succeeds, `REPLY HOSTID` (see [Section 6.4 \[REPLY HOSTID\]](#), page 43) will be sent from the server.

5.3.6 COPY CLIP (fromID, toID, update)

Copy a clip.

`'fromID'` The clipID of the clip you want to be used as the source of the copy. The source clip may be located anywhere on the network.

`'toID'` The clipID of the new clip you want to be created as a copy of fromID.

`'update'` *Optional.* Defaults to `false`.

This boolean field, when true, requests that the OECP Server send an immediate `REPLY COPY START` (see [Section 6.10 \[REPLY COPY START\]](#), page 53) reply immediately upon receiving this copy command. The reply copy start response will include information that allows the client to obtain more information about the progress of the copy.

If true, the client will immediately receive a `REPLY COPY START` (see [Section 6.10 \[REPLY COPY START\]](#), page 53). When the copy finishes, the client will *also* receive either a `REPLY OK` (see [Section 6.1.1 \[REPLY OK\]](#), page 43) or a `REPLY ERROR` (see [Section 6.1.2 \[REPLY ERROR\]](#), page 43) response.

If this command succeeds, `REPLY OK` (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.3.7 COPY EXTRACT CLIP (fromID, InPoint, outPoint, toID)

Copy clip fromID, from field/frame inPoint to field/frame outPoint, to new clip toID. Clip fromID must be located on the server node.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.3.8 DELETE CLIP (clipID, hostID, channelMask)

Delete clipID on hostID.

‘clipID’ The clip to be deleted.

‘hostID’ The host IP address of the node the clip resides on. While in general, we allow the use of 0.0.0.0 to mean “any host,” the 0.0.0.0 convention is not permitted for delete. We think that it is important to know *exactly* what clip you are trying to delete.

‘channelMask’

It is generally undesirable to delete a clip which is cued on a channel. By default, the node will refuse to delete a cued clip. You can force the node to ignore the fact that the clip is cued on specific channels. Set bit 0 of this mask to make the node ignore the problem if the clip is cued on Channel A. Use bit 1 for Channel B, and so forth. You may set this field to all-ones (0xffffffff) to indicate that you do not care at all if the clip is cued.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.3.9 FIND NEXT FREE CLIPID (clipID, hostID, inclFlag, wrapFlag)

Determine the next unused clipID.

‘clipID’ The clipID to begin searching from. The clipID must be a legal clipID in the range $1 \leq \text{clipID} \leq 9999999$.

‘hostID’ The IP Address of the Abekas 6000 you want to search on. If hostID is 0.0.0.0, all nodes on the network will be searched.

‘inclFlag’

A boolean flag which determines whether or not clipID is the first clipID to be checked for. If inclFlag is true, then clipID will be returned if clipID is free. Otherwise the search begins with the next clipID after clipID.

‘wrapFlag’

A boolean flag which determines whether or not the search should wrap around to ClipID #1 if the search extends past the end of the legal clip ID range.

If this command finds a free clipID matching the given criteria, it will return a REPLY CLIPID (see [Section 6.2 \[REPLY CLIPID\]](#), page 43). If no matching clip is found, but the search is a legal search, the REPLY CLIPID will be return with a clipID of 0.

5.3.10 FIND NEXT USED CLIPID (clipID, hostID, inclFlag, wrapFlag, findCueable)

Find the next clip which exists.

'clipID' The clipID to begin searching from. The clipID must be a legal clipID in the range $1 \leq \text{clipID} \leq 9999999$.

'hostID' The IP Address of the Abekas 6000 you want to search on. If hostID is 0.0.0.0, all nodes on the network will be searched.

'inclFlag'
A boolean flag which determines whether or not clipID is the first clipID to be checked for. If inclFlag is true, then clipID will be returned if clipID names an existing clip. Otherwise the search begins with the next legal clipID after clipID.

'wrapFlag'
A boolean flag which determines whether or not the search should wrap around to ClipID #1 if the search extends past the end of the legal clip ID range.

'findCueable'
A boolean flag which determines whether the search should return only the clipIDs of clips which could be cued on the system. For example, if findCueable is true (1), then clipIDs of 525 clips will not be returned on systems which are currently running in the 625 line standard.

If this command finds a clipID matching the given criteria, it will return a REPLY CLIPID HOSTID (see [Section 6.3 \[REPLY CLIPID HOSTID\], page 43](#)) with the clipID of the next clip and the IP Address of the Abekas 6000 containing the clip. If no matching clip is found, but the search is a legal search, the REPLY CLIPID HOSTID will be returned with a reply clipID of 0.

5.3.11 FIND PREVIOUS FREE CLIPID (clipID, hostID, inclFlag, wrapFlag)

Determine the preceding unused clipID.

'clipID' The clipID to begin searching from. The clipID must be a legal clipID in the range $1 \leq \text{clipID} \leq 9999999$.

'hostID' The IP Address of the Abekas 6000 you want to search on. If hostID is 0.0.0.0, all nodes on the network will be searched.

'inclFlag'
A boolean flag which determines whether or not clipID is the first clipID to be checked. If inclFlag is true, then clipID will be returned if clipID is free. Otherwise the search begins immediately preceding clipID.

'wrapFlag'
A boolean flag which determines whether or not the search should wrap around to ClipID #9999999 if no clipID less than clipID is determined to be free.

If this command finds a free clipID matching the given criteria, it will return a REPLY CLIPID (see [Section 6.2 \[REPLY CLIPID\]](#), page 43). If no matching clip is found, but the search is a legal search, the REPLY CLIPID will be return with a clipID of 0.

5.3.12 FIND PREVIOUS USED CLIPID (clipID, hostID, inclFlag, wrapFlag, findCueable)

Find the preceding clip which exists.

‘clipID’ The clipID to begin searching from. The clipID must be a legal clipID in the range $1 \leq \text{clipID} \leq 9999999$.

‘hostID’ The IP Address of the Abekas 6000 you want to search on. If hostID is 0.0.0.0, all nodes on the network will be searched.

‘inclFlag’ A boolean flag which determines whether or not clipID is the first clipID to be checked for. If inclFlag is true, then clipID will be returned if clipID names an existing clip. Otherwise the search begins preceding the clipID.

‘wrapFlag’ A boolean flag which determines whether or not the search should wrap around to ClipID #9999999 if no clipID less than clipID is determined to be in use.

‘findCueable’ A boolean flag which determines whether the search should return only the clipIDs of clips which could be cued on the system. For example, if findCueable is true (1), then clipIDs of 525 clips will not be returned on systems which are currently running in the 625 line standard.

If this command finds a clipID matching the given criteria, it will return a REPLY CLIPID HOSTID (see [Section 6.3 \[REPLY CLIPID HOSTID\]](#), page 43) with the clipID of the clip and the IP Address of the Abekas 6000 containing the clip. If no matching clip is found, but the search is a legal search, the REPLY CLIPID HOSTID will be returned with a reply clipID of 0.

5.3.13 GET CLIP INFO (clipID, hostID)

Return the full database description of a clip.

‘clipID’ The clipID of the clip that is being queried.

‘hostID’ The IP Address of the Abekas 6000 containing the clip. If hostID is 0.0.0.0, then the entire network will be searched for the clipID.

If the clipID exists on hostID, the server sends REPLY CLIPINFO (see [Section 6.5 \[REPLY CLIPINFO\]](#), page 44). If the clip is not found, a REPLY ERROR (see [Section 6.1.2 \[REPLY ERROR\]](#), page 43) is sent.

5.3.14 GET NEXT CLIP INFO (clipID, hostID)

Return a description of the next used clip on `hostID` after `clipID`. Wraps around at end of range.

- ‘`clipID`’ The clipID to begin searching from. `clipID` must be a legal clipID in the range $1 \leq \text{clipID} \leq 9999999$.
- ‘`hostID`’ The IP Address of the Abekas 6000 you want to search on. If `hostID` is 0.0.0.0, all nodes on the network will be searched.

If a matching clip is found, the server sends REPLY CLIPINFO (see [Section 6.5 \[REPLY CLIPINFO\]](#), page 44). If no matching clip is found, a REPLY ERROR (see [Section 6.1.2 \[REPLY ERROR\]](#), page 43) is sent.

5.3.15 GET PREV CLIP INFO (clipID, hostID)

Return a description of the next used clip on `hostID` prior to `clipID`. Wraps around at end of range.

- ‘`clipID`’ The clipID to begin searching from. `clipID` must be a legal clipID in the range $1 \leq \text{clipID} \leq 9999999$.
- ‘`hostID`’ The IP Address of the Abekas 6000 you want to search on. If `hostID` is 0.0.0.0, all nodes on the network will be searched.

If a matching clip is found, the server sends REPLY CLIPINFO (see [Section 6.5 \[REPLY CLIPINFO\]](#), page 44). If no matching clip is found, a REPLY ERROR (see [Section 6.1.2 \[REPLY ERROR\]](#), page 43) is sent.

5.3.16 LOCK CLIP (clipID, lock_unlock)

Toggle clip locking on or off. A clip may be locked by ANY external client, subject to certain constraints (e.g., a clip may not be locked while it is being edited). If a clip is locked, it may NOT be modified.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.3.17 MODIFY CLIP ATTRIBUTES (clipID, countAttrs, <attrID,attrVal>)

Modify one or more clip attributes.

The format of this command is:

OECP_MODIFY_CLIP_ATTRIBUTES	1 byte
DB_FROMID clipID	5 bytes
DB_COUNT_ATTRS count	5 bytes
count attributes selected from the following:	
OECP_CLIP_TITLE title	2 + strlen

OECP_CLIP_CREATOR	creator	2 + strlen
OECP_CLIP_PROJECT	project	2 + strlen
OECP_CLIP_KEYWORDS	keywords	2 + strlen
OECP_CLIP_PURGE_DATE	date	5
OECP_CLIP_BROWSE_FIELD	field	5
OECP_CLIP_REPEAT_IN_FIELD	field	5
OECP_CLIP_REPEAT_OUT_FIELD	field	5
OECP_CLIP_PLAYBACK_MODE	mode	2
OECP_CLIP_INTERP_MODE	mode	2
OECP_CLIP_REPEAT_CTRL_MODE	mode	2
OECP_CLIP_TC_PREF	pref	2
OECP_CLIP_SOURCE_DROPFRAME_PREFERENCE	pref	2
OECP_CLIP_VITC_DROPFRAME_PREFERENCE	pref	2
OECP_CLIP_TOD_DROPFRAME_PREFERENCE	pref	2

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

A brief description of each of these attributes follows:

5.3.17.1 Title

This field provides a title for the clip.

This field contains a single byte OECP_CLIP_TITLE, a single byte string length which must be no more than CLIP_TITLE_LEN, and the title string text.

5.3.17.2 Creator

This field describes the person or entity that created this clip.

This field contains a single byte OECP_CLIP_CREATOR, a single byte string length which must be no more than CLIP_CREATOR_LEN, and the creator string text.

5.3.17.3 Project

This field describes the project that the clip is a part of.

This field contains a single byte OECP_CLIP_PROJECT, a single byte string length which must be no more than CLIP_PROJECT_LEN, and the project string text.

5.3.17.4 Keywords

A clip may have up to CLIP_MAX_KEYWORDS keywords associated with it.

This field contains a single byte OECP_CLIP_KEYWORDS, a single byte keywords string length, and then the keywords string text.

The keywords string text contains all CLIP_MAX_KEYWORDS keywords, each of which is terminated by a zero byte. No single keyword may be longer than CLIP_KEYWORD_LEN.

If there are no keywords yet assigned to the clip, the keywords string will be `CLIP_MAX_KEYWORDS` long, and each character in the string will be a zero byte.

For example, if there are two keywords “key1” and “key2,” the keywords field would look like this.

```
OECP_CLIP_KEYWORDS
14
'k' 'e' 'y' '1' 0
'k' 'e' 'y' '2' 0
0
0
0
0
```

5.3.17.5 Purge Date

The time when the clip is destined to be eliminated. When a clip is created, the Abekas 6000 initializes this field to zero.

The Abekas 6000 currently preserves this field, but never uses it.

The `date` is a four byte unsigned integer. The `date` is the number of seconds since midnight January 1, 1970 as is traditional in the C Programming Language.

5.3.17.6 Browse Field

Some clips begin with black, and fade into video during playback. When previewing a clip, we want to know what single frame of the clip would be a good representative picture of the clip. This field provides the frame number of the representative image. When a clip is first allocated, the browse field is initialized to frame 0.

5.3.17.7 Repeat In Field

This field defaults to the start of the clip.

5.3.17.8 Repeat Out Field

This field defaults to the end of the clip.

5.3.17.9 Playback Mode

The playback mode of the clip. Possible values include:

‘PLAYBACK_MODE_FIELD’

The clip is composed of field based material. The Abekas 6000 can playback in variable speed field mode despite the fact that the underlying DV compressed material is stored and edited as whole frames.

`'PLAYBACK_MODE_FRAME_F12'`

The clip is composed of frames where matching sets of fields are a field1 and the following field2.

`'PLAYBACK_MODE_FRAME_F21'`

Not used at present.

`'PLAYBACK_MODE_FRAME'`

Not used at present.

5.3.17.10 Interpolator Mode

The interpolator mode of the clip. This is only used during field mode playback. Possible values include:

`'INTERPOLATOR_MODE_ON'`

The interpolator should be used during field mode variable playback.

`'INTERPOLATOR_MODE_OFF'`

The interpolator should never be used.

5.3.17.11 Repeat Control Mode

This field controls how the clip should loop when played back.

`'REPEAT_CONTROL_OFF'`

Clip should play linearly straight through the clip exactly once.

`'REPEAT_CONTROL_LOOP'`

Clip should loop back to the start of the clip after passing past the end of the clip.

`'REPEAT_CONTROL_PINGPONG'`

Clip should play backwards from the end of the clip after passing past the end of the clip. Clip should play forwards from the beginning of the clip after passing past the beginning of the clip.

`'REPEAT_CONTROL_LOOPTO'`

Clip should loop back to the repeat-in field whenever playback passes the repeat-out field.

`'REPEAT_CONTROL_PINGPONGTO'`

Clip should play backwards anytime it passes the repeat-out point while playing forwards. Clip should play forwards anytime it passes the repeat-in point while playing backwards.

5.3.17.12 Timecode Preference

5.3.17.13 Dropframe Preference

5.3.18 MOVE CLIP (fromID, toID)

Move clip fromID to new clip toID. If fromID is on the local node, Move is simply a clipID rename. If fromID is on a remote node, Move becomes a copy of fromID to new clip toID, followed by a deletion of fromID.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.3.19 MOVE EXTRACT CLIP (fromID, InPoint, outPoint, toID)

Extract a portion of a clip fromID and assign it to the clipID toID on the local node. Clip fromID must be located on the server node.

Move extract is a fast operation which does *NO* copying.

No copying takes place. This operation is destructive to the original clip. The clip toID will be composed of the specified material. The clip fromID will be trimmed to contain only that portion of the original clip after outPoint.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

5.3.20 TRIM CLIP (clipID, markIn, duration)

Trim a clip to a smaller size.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

'clipID' The clip to be trimmed.

'markIn' The field of the original clip which is to become the first field of the trimmed clip. A zero for this field indicates that the clip is not to be trimmed. A 60 for this field indicates that one second shall be removed from the front of the clip (assuming the clip is a 525/60 recording). markIn must be an even number.

'duration' The length in fields of the clip after it has been trimmed. duration must be an even number.

5.3.21 GET COPY PROGRESS (copyid)

Poll to obtain the progress of a copy command previously sent to the machine.

'copyid' The copy identifier returned in a previous REPLY COPY START (see [Section 6.10 \[REPLY COPY START\]](#), page 53) response. This is the copy whose progress the client wants to know more about.

If the copy is still in progress, the client will receive a `REPLY COPY PROGRESS` (see [Section 6.11 \[REPLY COPY PROGRESS\]](#), page 54) reply containing information about how much of the clip has been copied.

If the copy has completed, we will send as much information as is known about the manner in which the copy completed. Since we have a finite amount of space available to record the results of previous copy commands, new commands may force us to reuse the section of the table that was recording the results of the copy you were interested in.

The reply `REPLY COPYID NOT FOUND` (see [Section 6.12 \[REPLY COPYID NOT FOUND\]](#), page 54) is sent if the copy has completed but the server no longer remembers whether the copy succeeded or failed. The server only forgets the results if new copy commands were sent to the server that forced the server to reuse the results entry in the table.

The reply `REPLY COPY DONE OK` (see [Section 6.13 \[REPLY COPY DONE OK\]](#), page 54) is sent if the copy has completed, and the server remembers that the copy succeeded.

The reply `REPLY COPY DONE ERROR` (see [Section 6.14 \[REPLY COPY DONE ERROR\]](#), page 54) is sent if the copy failed, and the server remembers the failure error message.

Please be aware that the copyid is only valid so long as your client stays connected to the OECP Server. The 6000 system software contains a web server. Viewing the web page `/oecp/copyprogress.html` will allow you to see what copyids are currently active.

5.4 External VTR Commands

Each VTR command sends a command to a `channel`. This means that the command is sent to the VTR currently associated (attached) to the specified 6000 channel.

All VTR commands return immediately with either `REPLY OK` (see [Section 6.1.1 \[REPLY OK\]](#), page 43) or `REPLY ERROR` (see [Section 6.1.2 \[REPLY ERROR\]](#), page 43). Commands whose actions take awhile to run should be considered to be started once `REPLY OK` has been returned.

5.4.1 VTR ABORT (channel)

Abort a `VTR TO TAPE` (see [Section 5.4.15 \[VTR TO TAPE\]](#), page 36), `VTR TO CLIP` (see [Section 5.4.13 \[VTR TO CLIP\]](#), page 35), `VTR PLAY FOR` (see [Section 5.4.7 \[VTR PLAY FOR\]](#), page 35), or `VTR VARPLAY FOR` (see [Section 5.4.18 \[VTR VARPLAY FOR\]](#), page 36) command which is currently executing.

5.4.2 VTR BYPASS (channel, onoff)

If `onoff` is 1, switch VTR to EE mode. If `onoff` is 0, switch VTR to TAPE mode.

5.4.3 VTR FAST FWD (channel)

Send a fast forward command to the VTR.

5.4.4 VTR GOTO (channel, timecode)

Go to timecode on the VTR.

5.4.5 VTR JOG (channel, direction)

Jog the VTR in direction.

5.4.6 VTR PLAY (channel)

Put the VTR into play.

5.4.7 VTR PLAY FOR (channel, duration)

Play the VTR for duration.

5.4.8 VTR RECORD (channel, chbits)

Send a record command to the VTR.

5.4.9 VTR REWIND (channel)

Put the VTR into rewind.

5.4.10 VTR SHUTTLE (channel, speed)

Put the VTR into shuttle at speed.

5.4.11 VTR STILL (channel)

Put the VTR into still.

5.4.12 VTR STOP (channel)

Send a stop command to the VTR.

5.4.13 VTR TO CLIP (channel, clipin, vtrin, duration, clipid, chbits)

Record onto clipid at clipin for duration field from the VTR on controlling channel channel at vtrin as the source material. Record only tracks identified by chbits.

5.4.14 VTR TO CLIP PVW (channel, clipin, vtrin, duration, clipid, chbits)

Preview a VTR TO CLIP (see [Section 5.4.13 \[VTR TO CLIP\]](#), page 35) command without actually recording anything.

5.4.15 VTR TO TAPE (channel, clipin, vtrin, duration, clipid, chbits)

Record onto the VTR at `vtrin` for `duration` fields, using `clipid` at `clipin` as the source material. Record only tracks identified by `chbits`.

5.4.16 VTR TO TAPE PVW (channel, clipin, vtrin, duration, clipid, chbits)

Preview a VTR TO TAPE (see [Section 5.4.15 \[VTR TO TAPE\]](#), page 36) command without actually recording anything.

5.4.17 VTR VARPLAY (channel, speed)

Put VTR into varplay at `speed`.

5.4.18 VTR VARPLAY FOR (channel, speed, duration)

Put VTR into varplay at `speed` for `duration` fields of realtime have elapsed.

5.5 Report Commands

Clients will sometimes be interested in a variety of status information such as channel timecode, channel clipid, and clip modifications. A client can either make an explicit request for these reports, or can register itself to receive automatic updates of this information. The report command group is the set of commands used by the client to obtain reports.

5.5.1 GET REPORT

This command tells the server to send a piece of status information. The exact format of the command arguments depend on the kind of information the client is requesting.

The first argument to this command is a report type. What additional arguments are expected by the server depend on the value of the report type.

‘Channel Status Notification’

This report type requests the current channel status of a channel.

‘channel’ The channel to fetch the channel status of. If `channel` is set to `OECP_ALL_CHNLS`, then the channel status of every channel will be returned.

‘Channel Timecode Notification’

This report type requests the current output timecode of a channel.

‘channel’ The channel to request the timecode of. If **channel** is set to **OECP_ALL_CHNLS**, then the timecode of every channel will be returned.

5.5.2 REGISTER FOR REPORT

This command requests that the client be notified of changes in some type of status.

The first argument to this command is a report type. What additional arguments are expected by the server depend on the value of the report type.

‘Request Clip Change Notification’

Request notification when certain types of changes occur to the clip database. Two additional arguments must be sent by the client.

‘Change Type’

The first additional argument is the change type value. This value specifies the types of changes that the client is interested in.

‘RPT_CLIP_CHANGE_TYPE_ANY’

The client is interested in all of the following change types.

‘RPT_CLIP_CHANGE_TYPE_NEW’

I don’t know.

‘RPT_CLIP_CHANGE_TYPE_DELETED’

Inform the client when a clip is deleted.

‘RPT_CLIP_CHANGE_TYPE_TRACKS’

I don’t know.

‘RPT_CLIP_CHANGE_TYPE_RECORD_EXTENT’

I don’t know.

‘RPT_CLIP_CHANGE_TYPE_RENAME’

Inform the client when a clip is renamed (moved).

‘RPT_CLIP_CHANGE_TYPE_HEADER’

Inform the client when the metadata of a clip is changed.

‘clipid’ The second argument specifies what clip the client is interested in receiving notifications about. If **clipid** is set to **ALL_CLIPS**, then the client will be notified of the change type regardless of what clip is changed.

‘Channel Status Notification’

This report type requests client notifications when the channel status of a channel changes.

‘channel’ The channel to request channel status updates for. If `channel` is set to `OECP_ALL_CHNLS`, then channel status updates will be sent for changes of channel status on all channels.

‘Channel Timecode Notification’

This report type requests client notifications when the timecode value of a channel changes.

‘channel’ The channel to request timecode updates for. If `channel` is set to `OECP_ALL_CHNLS`, then timecode updates will be sent for all channels.

5.5.3 UNREGISTER FOR REPORT

This command cancels a previous registration the client for a particular report type. The arguments are the same as for REGISTER FOR REPORT (see [Section 5.5.2 \[REGISTER FOR REPORT\]](#), page 37).

5.5.4 GET PARAM (paramID)

A number items of internal status on the Abekas 6000 are available through the GET PARAM interface. A number of parameters are provided that the client can query the value of. This API call is intended to be simpler than the send report API; in particular, it is easier to extend within the 6000 to provide additional information when requested by customers.

The first and only argument for this command parameter ID, which is a 32 bit value which identifies which parameter the client wants to know the value of.

‘paramID’ The parameter ID of the parameter the client wants the value of. Can be any one of the following

‘OECP_PARAMID_FREESPACE_DV25’

Return the amount of unallocated space on the system in terms of the number of fields of DV25 video that could be recorded into the free space. Note that it may not be possible to record a single clip that is this long, rather, this is the sum of all available space.

‘OECP_PARAMID_FREESPACE_DV50’

Like `OECP_PARAMID_FREESPACE_DV25`, but for DV50 instead of DV25.

‘OECP_PARAMID_FREESPACE_MPEG25’

Like `OECP_PARAMID_FREESPACE_DV25`, but for MPEG25 instead of DV25.

‘OECP_PARAMID_FREESPACE_MPEG50’

Like `OECP_PARAMID_FREESPACE_DV25`, but for MPEG50 instead of DV25.

- ‘OECP_PARAMID_MAXCREATE_DV25’
The length in fields of the longest DV25 that you can currently allocate. This may be less than OECP_PARAMID_FREESPACE25 due to fragmentation.
- ‘OECP_PARAMID_MAXCREATE_DV50’
Like OECP_PARAMID_MAXCREATE_DV25, but for DV50 instead of DV25.
- ‘OECP_PARAMID_MAXCREATE_MPEG25’
Like OECP_PARAMID_FREESPACE_DV25, but for MPEG25 instead of DV25.
- ‘OECP_PARAMID_MAXCREATE_MPEG50’
Like OECP_PARAMID_MAXCREATE_DV25, but for MPEG50 instead of DV25.
- ‘OECP_PARAMID_CAPACITY_DV25’
Return total capacity of the system’s video storage expressed as the number of fields of DV25 video the system could record.
- ‘OECP_PARAMID_CAPACITY_DV50’
Like OECP_PARAMID_CAPACITY_DV25, but for DV50 instead of DV25.
- ‘OECP_PARAMID_CAPACITY_MPEG25’
Like OECP_PARAMID_CAPACITY_DV25, but for MPEG25 instead of DV25.
- ‘OECP_PARAMID_CAPACITY_MPEG50’
Like OECP_PARAMID_CAPACITY_DV25, but for MPEG50 instead of DV25.
- ‘OECP_PARAMID_REMOTE_CONFIG_CHA’
‘OECP_PARAMID_REMOTE_CONFIG_CHB’
‘OECP_PARAMID_REMOTE_CONFIG_CHC’
‘OECP_PARAMID_REMOTE_CONFIG_CHD’
‘OECP_PARAMID_REMOTE_CONFIG_CHE’
‘OECP_PARAMID_REMOTE_CONFIG_CHF’
‘OECP_PARAMID_REMOTE_CONFIG_CHG’
‘OECP_PARAMID_REMOTE_CONFIG_CHH’
These parameters correspond to the “Remote Config #” setting in the “REMOTE SETUP” menu of the 6000.

```

‘OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_1’
‘OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_2’
‘OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_3’
‘OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_4’
‘OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_5’
‘OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_6’
‘OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_7’
‘OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_8’
‘OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_9’
‘OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_10’
‘OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_11’

```

These parameters correspond to the “AutoEdit Config#” setting in the “AUTO EDIT SETUP” menu of the 6000.

If this command succeeds, REPLY PARAM (see [Section 6.9 \[REPLY PARAM\]](#), page 53) will be sent from the server to let the client know that value of the requested parameter.

5.5.5 SET PARAM (paramID, paramValue)

A number items of internal state and configuration on the Abekas 6000 can be set. All of these piece of state can be read with GET PARAM (see [Section 5.5.4 \[GET PARAM\]](#), page 38).

The first argument for this command parameter ID, which is a 32 bit value which identifies which parameter the client wants to know the value of.

The second argument for this command is a value to set for the parameter. This value is 32 bits.

‘paramID’ The parameter ID of the parameter the client wants to set. Can be any one of the following

```

‘OECF_PARAMID_REMOTE_CONFIG_CHA’
‘OECF_PARAMID_REMOTE_CONFIG_CHB’
‘OECF_PARAMID_REMOTE_CONFIG_CHC’
‘OECF_PARAMID_REMOTE_CONFIG_CHD’
‘OECF_PARAMID_REMOTE_CONFIG_CHE’
‘OECF_PARAMID_REMOTE_CONFIG_CHF’
‘OECF_PARAMID_REMOTE_CONFIG_CHG’
‘OECF_PARAMID_REMOTE_CONFIG_CHH’

```

These parameters correspond to the “Remote Config #” setting in the “REMOTE SETUP” menu of the 6000.

```
'OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_1'  
'OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_2'  
'OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_3'  
'OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_4'  
'OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_5'  
'OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_6'  
'OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_7'  
'OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_8'  
'OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_9'  
'OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_10'  
'OECF_PARAMID_AUTOEDIT_CONFIG_EDITPORT_11'
```

These parameters correspond to the “AutoEdit Config#” setting in the “AUTO EDIT SETUP” menu of the 6000.

If this command succeeds, REPLY OK (see [Section 6.1.1 \[REPLY OK\]](#), page 43) will be sent from the server.

6 Replies

This chapter attempts to categorize the spectrum of possible replies your client may receive.

6.1 REPLY NONE

When the client receives this reply, everything it needs to know is really carried in the error string. There is no extra information (see [Section 4.8 \[REPLY TYPE SPECIFIC INFORMATION\]](#), page 9) carried in this reply.

6.1.1 REPLY OK

Some commands return no information. If they succeed, they return a REPLY NONE with the error message is set to “OK.”

6.1.2 REPLY ERROR

A command that fails usually sends a REPLY NONE with the error message string set to something other than “OK.”

6.2 REPLY CLIPID

This reply message includes just one piece of information, a clipID. The error message string is always “OK” for this message type.

OECP_COMMAND_REPLY_TYPE	OECP_CMD_REPLY_CLIPID	2 bytes
DB_CLIPID	clipid	5 bytes

6.3 REPLY CLIPID HOSTID

This reply message includes two pieces of information, a clipID and a node IP Address.

OECP_COMMAND_REPLY_TYPE	OECP_CMD_REPLY_CLIPID_HOSTID	2 bytes
DB_CLIPID	clipid	5 bytes
DB_HOSTID	hostID	5 bytes

6.4 REPLY HOSTID

This reply message includes one piece of information, a node IP Address.

OECP_COMMAND_REPLY_TYPE	OECP_CMD_REPLY_HOSTID	2 bytes
DB_HOSTID	hostID	5

6.5 REPLY CLIPINFO

The Clip Info Response contains a complete description of all database information regarding a particular clip.

OECP_COMMAND_REPLY_TYPE	OECP_CMD_REPLY_CLIPINFO	2 bytes
DB_HOSTID	hostid	5
OECP_CLIP_ID	clipid	5
OECP_CLIP_NFIELDS	length	5
OECP_CLIP_ALLOCTYPE	alloctype	2
OECP_CLIP_TITLE	titleString	2 + strlen
OECP_CLIP_CREATOR	creatorString	2 + strlen
OECP_CLIP_PROJECT	projectString	2 + strlen
OECP_CLIP_KEYWORDS	keywords	2 + strlen
OECP_CLIP_SOURCE_DROPFRAME_PREFERENCE	df	2
OECP_CLIP_SRC_TC	tc	5
OECP_CLIP_VITC_DROPFRAME_PREFERENCE	df	2
OECP_CLIP_VITC_TC	tc	5
OECP_CLIP_TOD_DROPFRAME_PREFERENCE	df	2
OECP_CLIP_TOD_TC	tc	5
OECP_CLIP_TC_PREF	tcpref	2
OECP_CLIP_CREATION_DATE	date	5
OECP_CLIP_LAST_ACCESS_DATE	date	5
OECP_CLIP_PURGE_DATE	date	5
OECP_CLIP_LOCK_STATE	lockstate	2
OECP_CLIP_BROWSE_FIELD	fieldnum	5
OECP_CLIP_REPEAT_IN_FIELD	in	5
OECP_CLIP_REPEAT_OUT_FIELD	out	5
OECP_CLIP_PLAYBACK_MODE	mode	2
OECP_CLIP_INTERP_MODE	mode	2
OECP_CLIP_REPEAT_CTRL_MODE	mode	2
OECP_CLIP_TRACKS	numtracks	2
repeat	numtracks times:	
OECP_CLIP_TRACK_NUMBER	tracknum	2
OECP_CLIP_TRACK_TYPE	trackType	2
OECP_CLIP_TRACK_VIDEO_STANDARD	std	2
if track is audio:		
OECP_CLIP_TRACK_AUDIO_STANDARD	std	2
OECP_CLIP_TRACK_PHASE	phase	5
OECP_CLIP_TRACK_SUB_CHANNEL_MASK	mask	2
if track is video or key		
OECP_CLIP_TRACK_VIDEO_RESOLUTION	res	2
if track is key		
OECP_CLIP_TRACK_KEY_TYPE	type	2

Brief descriptions of each of these fields follows...

6.5.1 Host ID

The IP Address of the node on which the clip is recorded.

6.5.2 Clip ID

The clipID of the clip being described.

6.5.3 Length

The length of the clip, expressed as a multiple of fields. For example, a one second clip in 525/60 is 60 fields long.

6.5.4 Allocaction Type

A clip can be as simple as a pre-reserved identifier, or as complex as a clip having been fully recorded, or somewhere in between. The `alloctype` describes how far along this process the clip is.

Possible values for `alloctype` include:

`'ALLOCTYPE_UNKNOWN'`

I don't know.

`'ALLOCTYPE_FREE'`

The clipID is not in use on the network.

`'ALLOCTYPE_ALLOCID'`

The clipID is in use. It has been *reserved* on behalf of a client.

`'ALLOCTYPE_ALLOCHDR'`

Clip attributes have been defined, but no tracks exist for the clip.

`'ALLOCTYPE_ALLOCALL'`

The clip has been fully allocated, but has not yet been recorded to.

`'ALLOCTYPE_RECORD'`

The clip has been recorded.

6.5.5 Title

This field provides a title for the clip.

This field contains a single byte `OECP_CLIP_TITLE`, a single byte string length which must be no more than `CLIP_TITLE_LEN`, and the title string text.

6.5.6 Creator

This field describes the person or entity that created this clip.

This field contains a single byte `OECP_CLIP_CREATOR`, a single byte string length which must be no more than `CLIP_CREATOR_LEN`, and the creator string text.

6.5.7 Project

This field describes the project that the clip is a part of.

This field contains a single byte `OECP_CLIP_PROJECT`, a single byte string length which must be no more than `CLIP_PROJECT_LEN`, and the project string text.

6.5.8 Keywords

A clip may have up to `CLIP_MAX_KEYWORDS` keywords associated with it.

This field contains a single byte `OECP_CLIP_KEYWORDS`, a single byte keywords string length, and then the keywords string text.

The keywords string text contains all `CLIP_MAX_KEYWORDS` keywords, each of which is terminated by a zero byte. No single keyword may be longer than `CLIP_KEYWORD_LEN`.

If there are no keywords yet assigned to the clip, the keywords string will be `CLIP_MAX_KEYWORDS` long, and each character in the string will be a zero byte.

For example, if there are two keywords “key1” and “key2,” the keywords field would look like this.

```
OECP_CLIP_KEYWORDS
14
'k' 'e' 'y' '1' 0
'k' 'e' 'y' '2' 0
0
0
0
0
```

6.5.9 Timecode Fields

The database keeps track of three timecodes associated with each clip. Each timecode field specifies a timecode for the first field of the clip. Timecodes are specified as a field number where field 0 is midnight (00:00:00:00). All timecodes have a dropframe attribute which specifies whether or not the timecode should be manipulated as a dropframe timecode or as a nondropframe timecode. This dropframe attribute exists even in 625 where it is meaningless.

The first timecode is a user specifiable “source timecode.” This user-editable field allows the user to specify an arbitrary timecode for the start of the clip. The database field `OECP_CLIP_SRC_TC` specifies this timecode, and the field `OECP_CLIP_SOURCE_DROPFRAME_PREFERENCE` determines whether or not the source timecode is dropframe or nondropframe.

The second timecode is called the time of day timecode. This field records the time when the clip was created.

The third timecode is called the vitc timecode. This database field remembers the VITC timecode of the recorded video when the video was first recorded. Since this is a single timecode, it cannot represent broken VITC within the recorded video. Just the VITC of the video at record time.

6.5.10 Timecode Preference

When a clip is cued, the timecode reported by the Abekas 6000 is based on one of four methods. This field determines which method is used.

`'PLAYBACK_TC_PREF_CLIP'`

Timecode is zero based. The first field of the clip has timecode 00:00:00:00.

`'PLAYBACK_TC_PREF_SOURCE_TC'`

Timecode is reported based on the client specified source timecode of the clip.

`'PLAYBACK_TC_PREF_VITC'`

Timecode is reported based on the input VITC when the clip was first recorded.

`'PLAYBACK_TC_PREF_TOD'`

Timecode is reported based upon the initial record time of the clip.

`'PLAYBACK_TC_PREF_REC_VITC'`

Timecode is reported based upon the recorded-in VITC timecode, which may be discontinuous.

6.5.11 Creation Date

This field describes when the clip was first created.

The `date` is a four byte unsigned integer. The `date` is the number of seconds since midnight January 1, 1970 as is traditional in the C Programming Language.

6.5.12 Last Access Date

This field describes the last time this clip was cued.

The `date` is a four byte unsigned integer. The `date` is the number of seconds since midnight January 1, 1970 as is traditional in the C Programming Language.

6.5.13 Purge Date

The time when the clip is destined to be eliminated. When a clip is created, the Abekas 6000 initializes this field to zero.

The Abekas 6000 currently preserves this field, but never uses it.

The `date` is a four byte unsigned integer. The `date` is the number of seconds since midnight January 1, 1970 as is traditional in the C Programming Language.

6.5.14 Lock State

This database field indicates whether or not the clip should be protected. When locked, the clip cannot be deleted, moved, trimmed, or recorded to; but it can be copied.

There are two possible values for the lockstate byte:

‘LOCK_CLIP_MODE_OFF’

Not protected. This is the default when a clip is created.

‘LOCK_CLIP_MODE_ON’

Protected.

6.5.15 Browse Field

Some clips begin with black, and fade into video during playback. When previewing a clip, we want to know what single frame of the clip would be a good representative picture of the clip. This field provides the frame number of the representative image. When a clip is first allocated, the browse field is initialized to frame 0.

6.5.16 Repeat In Field

This field defaults to the start of the clip.

6.5.17 Repeat Out Field

This field defaults to the end of the clip.

6.5.18 Playback Mode

The playback mode of the clip. Possible values include:

‘PLAYBACK_MODE_FIELD’

The clip is composed of field based material. The Abekas 6000 can playback in variable speed field mode despite the fact that the underlying DV compressed material is stored and edited as whole frames.

‘PLAYBACK_MODE_FRAME_F12’

The clip is composed of frames where matching sets of fields are a field1 and the following field2.

‘PLAYBACK_MODE_FRAME_F21’

Not used at present.

‘PLAYBACK_MODE_FRAME’

Not used at present.

6.5.19 Interpolator Mode

The interpolator mode of the clip. This is only used during field mode playback. Possible values include:

‘INTERPOLATOR_MODE_ON’

The interpolator should be used during field mode variable playback.

‘INTERPOLATOR_MODE_OFF’

The interpolator should never be used.

6.5.20 Repeat Control Mode

This field controls how the clip should loop when played back.

`'REPEAT_CONTROL_OFF'`

Clip should play linearly straight through the clip exactly once.

`'REPEAT_CONTROL_LOOP'`

Clip should loop back to the start of the clip after passing past the end of the clip.

`'REPEAT_CONTROL_PINGPONG'`

Clip should play backwards from the end of the clip after passing past the end of the clip. Clip should play forwards from the beginning of the clip after passing past the beginning of the clip.

`'REPEAT_CONTROL_LOOPTO'`

Clip should look back to the repeat-in field whenever playback passes the repeat-out field.

`'REPEAT_CONTROL_PINGPONGTO'`

Clip should play backwards anytime it passes the repeat-out point while playing forwards. Clip should play forwards anytime it passes the repeat-in point while playing backwards.

6.5.21 Number of Tracks

Number of tracks in the clip. A clip is composed of some combination of a video track, a key track, and an audio track. For example, a clip might have no tracks; just a video track; or a video, audio, and key track.

A track record is returned for each track.

6.5.22 Track Number

The track number within the clip. These numbers are not necessarily contiguous or in order. This value is a user specified handle for the track provided in `clipRecordSpec`.

6.5.23 Track Type

The track type lets you know what kind of track is being described. Possible values include:

`'TRACK_TYPE_AES_AUDIO'`

Unused audio type in the Abekas 6000.

`'TRACK_TYPE_CCIR601_VIDEO'`

Unused video type in the Abekas 6000.

`'TRACK_TYPE_CCIR601_KEY'`

Unused key type in the Abekas 6000.

‘TRACK_TYPE_UNKNOWN’

This should never happen.

‘TRACK_TYPE_DV25_KEY’

A key track recorded in the 25Mb/s DV compression standard.

‘TRACK_TYPE_DV25_VIDEO’

A video track recorded in the 25Mb/s DV compression standard.

‘TRACK_TYPE_DV50_VIDEO’

A video track recorded in the 50Mb/s DV compression standard.

‘TRACK_TYPE_DV50_KEY’

A key track recorded in the 50Mb/s DV compression standard.

‘TRACK_TYPE_MPEG25_KEY’

A key track recorded as 25Mb/s I-Frame only MPEG2.

‘TRACK_TYPE_MPEG25_VIDEO’

A video track recorded as 25Mb/s I-Frame only MPEG2.

‘TRACK_TYPE_MPEG50_VIDEO’

A video track recorded as 50Mb/s I-Frame only MPEG2.

‘TRACK_TYPE_MPEG50_KEY’

A key track recorded as 50Mb/s I-Frame only MPEG2.

‘TRACK_TYPE_DV_AUDIO’

The audio format used on the Abekas 6000.

6.5.24 Track Audio Standard

There is only one value for this field. Informative, no?

‘AUDIO_STANDARD_AES’

6.5.25 Track Audio Phase

525/60 is not really a 60Hz frequency. It is just shy of this frequency. Unfortunately, AES audio is exactly 48kHz. So if you want to record audio into field or frame sized “clumps” to simplify editing on a video system, the audio “frames” do not all contain the same number of samples. The software subsystem must keep track of the phase relationship between the video and audio to ensure that the right number of samples are played back from each recorded audio frame.

When we first allocate an audio track, we mark it’s phase as 0. Anytime we trim a frame from the front of the clip, we add 2 to the phase. This way, we can always reconstruct the original phase relationship of the recorded material.

All professional VTR tape formats include an audio versus video phase relationship in 525.

In 625/50, the frequency is 50Hz, which is an exact multiple of the Professional AES frequency of 48kHz. Therefore, this brain damage is uninteresting to 625/50 recordings.

6.5.26 Track Audio Subchannel Mask

I don't know what this does.

6.5.27 Track Video Standard

A clip may contain tracks that are playable only in 525/60, and also tracks that are playable only in 625/50. This field describes the recording frequency of this track

'VIDEO_STANDARD_FPS60'

525/60

'VIDEO_STANDARD_FPS50'

625/50

'VIDEO_STANDARD_FPS48'

625/48. Popular for editing film since film is recorded at 24 frames per second. This value is not currently used on the Abekas 6000.

6.5.28 Track Video Resolution

Serial Digital Video can be recorded in either eight or ten bits per pixel. Because the Abekas 6000 is a compressed video recorder, this field is always set to VIDEO_BITS_PER_PIXEL_8.

'VIDEO_BITS_PER_PIXEL_8'

Recording stores eight bits per pixel.

'VIDEO_BITS_PER_PIXEL_10'

Recording stores ten bits per pixel.

6.5.29 Track Key Type

Determines whether or not the key track has been stored in half the bandwidth of a video track.

On the Abekas 6000, this field is *always* set to KEY_TYPE_FULL_BANDWIDTH.

'KEY_TYPE_FULL_BANDWIDTH'

Key is recorded in the same amount of storage space as video.

'KEY_TYPE_HALF_BANDWIDTH'

Key is recorded half the same amount of storage space as video.

6.6 REPLY CHANNEL ATTRIBUTES

The Channel Attributes response returns a description of the state of a channel.

OECP_COMMAND_REPLY_TYPE	OECP_CMD_REPLY_CHNL_ATTRS	2 bytes
OECP_CHNL_INTERP_MODE	mode	2
OECP_CHNL_OUTPUT_MODE	mode	2

OECP_CHNL_LOOP_MODE mode	2
OECP_CHNL_DV_COMPRESSION compression	2
OECP_CHNL_TIMECODE_MODE mode	2
OECP_CHNL_TIMECODE_OFFSET tcoffset	5
OECP_CHNL_TIMECODE_DROPFRAME_TYPE dftype	2

Brief descriptions of each of these fields follows...

6.6.1 Interpolator Mode

'OECP_CHNL_INTERP_MODE_OFF'
 'OECP_CHNL_INTERP_MODE_ON'
 'OECP_CHNL_INTERP_MODE_AUTO'

6.6.2 Output Mode

'OECP_CHNL_OUTPUT_MODE_FIELD'
 'OECP_CHNL_OUTPUT_MODE_FRAME12'
 'OECP_CHNL_OUTPUT_MODE_FRAME21'

6.6.3 Loop Mode

'OECP_CHNL_LOOP_MODE_OFF'
 'OECP_CHNL_LOOP_MODE_LOOP'
 'OECP_CHNL_LOOP_MODE_PINGPONG'
 'OECP_CHNL_LOOP_MODE_LOOPTO'
 'OECP_CHNL_LOOP_MODE_PINGPONGTO'

6.6.4 Timecode Mode

'OECP_CHNL_TIMECODE_MODE_SOURCE'
 'OECP_CHNL_TIMECODE_MODE_TOD'
 'OECP_CHNL_TIMECODE_MODE_VITC'

6.6.5 Timecode Offset

6.6.6 Dropframe Mode

Specifies how the timecode should be displayed for this clip.

'OECP_CHNL_TIMECODE_DROPFRAME_TYPE_NDF'
 Timecode for this channel should be displayed as non-drop frame.

'OECP_CHNL_TIMECODE_DROPFRAME_TYPE_DF'
 Timecode for this channel should be displayed as drop frame.

6.7 REPLY REPORT CHANNEL TIMECODE

The Channel Timecode response reports the timecode for one or more channels.

OECP_COMMAND_REPLY_TYPE	OECP_CMD_REPLY_REPORT_CHNL_TIMECODE	2 bytes
RPT_NUM_BATCH_REPORTS	count	2
repeat the following count times:		
RPT_CHANNEL	channel	5
RPT_CHANNEL_TIMECODE	timecode	5

6.8 REPLY REPORT CHANNEL STATUS

The Channel Status response reports the status of one or more channels.

OECP_COMMAND_REPLY_TYPE	OECP_CMD_REPLY_REPORT_CHNL_STATUS	2 bytes
RPT_NUM_BATCH_REPORTS	count	2
repeat the following count times:		
RPT_CHANNEL	channel	5
RPT_CHANNEL_STATUS	status	5

6.9 REPLY PARAM

The parameter reply reports the value of a system parameter back to the client.

Currently, all parameter values reported are 32 bit values. In the future, this may be extended to be report other types depending upon the value of the parameter id.

OECP_COMMAND_REPLY_TYPE	OECP_CMD_REPLY_PARAM	2 bytes
RPT_PARAMID	paramid	5
RPT_PARAMVALUE	parametervalue	5

6.10 REPLY COPY START

This reply informs the client that a copy requested by the client has begun. This reply is only sent if the client explicitly requests this information by setting the `update` argument to the COPY CLIP (see [Section 5.3.6 \[COPY CLIP\], page 25](#)) command.

The reply includes a `copyid` identifier which can be used to obtain additional information regarding the progress of the copy command using the command GET COPY PROGRESS (see [Section 5.3.21 \[GET COPY PROGRESS\], page 33](#)).

Please be aware that the `copyid` is only valid so long as your client stays connected to the OECP Server. The 6000 system software contains a web server. Viewing the web page `/oecp/copyprogress.html` will allow you to see what `copyids` are currently active.

OECP_COMMAND_REPLY_TYPE	OECP_CMD_REPLY_COPY_START	2 bytes
DB_COPY_ID	copyid	5

6.11 REPLY COPY PROGRESS

This response to the command GET COPY PROGRESS (see [Section 5.3.21 \[GET COPY PROGRESS\]](#), page 33) informs the client that the requested copy is still in progress, and contains information regarding how much of the clip has been copied so far.

This reply contains a `copyprogress` field which is a 16:16 fixed point value which represents the percentage of the clip which has been copied so far. Thus the value 0 will indicate that no fields of the clip have yet been copied; the value `0x320000` (50%) indicates that the copy is half done; and the value `0x640000` (100%) indicates the entire content has been copied.

OECP_COMMAND_REPLY_TYPE	OECP_CMD_REPLY_COPY_PROGRESS	2 bytes
DB_COPY_ID	copyid	5
DB_COPY_PROGRESS	copyprogress	5

6.12 REPLY COPYID NOT FOUND

This response to the command GET COPY PROGRESS (see [Section 5.3.21 \[GET COPY PROGRESS\]](#), page 33) informs the client that the copy has completed but that the server no longer recalls whether the copy succeeded or failed.

This response can also be sent if the client sends GET COPY PROGRESS with an incorrect `copyid`; that is, one that did not come from the server.

OECP_COMMAND_REPLY_TYPE	OECP_CMD_REPLY_COPYID_NOT_FOUND	2 bytes
DB_COPY_ID	copyid	5

6.13 REPLY COPY DONE OK

This response to the command GET COPY PROGRESS (see [Section 5.3.21 \[GET COPY PROGRESS\]](#), page 33) informs the client that the copy has completed, and that the server remembers that the copy was completed correctly.

OECP_COMMAND_REPLY_TYPE	OECP_CMD_REPLY_COPY_DONE_OK	2 bytes
DB_COPY_ID	copyid	5

6.14 REPLY COPY DONE ERROR

This response to the command GET COPY PROGRESS (see [Section 5.3.21 \[GET COPY PROGRESS\]](#), page 33) informs the client that the copy failed. The server includes a copy of the error message originally sent as a REPLY ERROR (see [Section 6.1.2 \[REPLY ERROR\]](#), page 43) in response to the original COPY CLIP (see [Section 5.3.6 \[COPY CLIP\]](#), page 25) command.

OECP_COMMAND_REPLY_TYPE	OECP_CMD_REPLY_COPY_DONE_OK	2 bytes
DB_COPY_ID	copyid	5
OECP_COMMAND_REPLY_ERC_TYPE	orig_error_code	3
original error message	byte count	2

original error message ASCII text

variable

7 Examples

7.1 Configuration Commands

7.1.1 ACQUIRE CHANNEL

The following command requests acquisition of Channel C and Channel E.

```
E7 E7 E7      Message header
00 11         bytcount = 17
DA DA DA     command header
01 00 01 D4 C1 OECP_EXT_TRANSID_TYPE 120001
02 03         OECP_COMMAND_ID_TYPE OECP_CMD_CONFIG_TYPE
03 00         OECP_SUB_COMMAND_ID_TYPE OECP_ACQUIRE_CHANNEL
60 00 00 00 14 CONFIG_CHANNEL_MASK 0x14
```

If the command succeeds, the following response is sent:

```
D6 D6 D6      reply header
00 0E         byte count = 14
01 00 01 D4 C1 OECP_EXT_TRANSID_TYPE 120001
05 00 00      OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02         Error string length = 2
4F 4B        "OK"
04 01         OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_NONE
```

7.1.2 RELEASE CHANNEL

The following command releases Channel C and Channel E..

```
E7 E7 E7      Message header
00 11         bytcount = 17
DA DA DA     command header
01 00 01 D4 C2 OECP_EXT_TRANSID_TYPE 120002
02 03         OECP_COMMAND_ID_TYPE OECP_CMD_CONFIG_TYPE
03 01         OECP_SUB_COMMAND_ID_TYPE OECP_RELEASE_CHANNEL
60 00 00 00 14 CONFIG_CHANNEL_MASK 0x14
```

If the command succeeds, the following response is sent:

```

D6 D6 D6      reply header
00 0E         byte count = 14
01 00 01 D4 C2 OECP_EXT_TRANSID_TYPE 120002
05 00 00      OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02         Error string length = 2
4F 4B        "OK"
04 01         OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_NONE

```

7.1.3 FORCE RELEASE CHANNEL

The following command releases Channel C and Channel E, no matter who owns it.

```

E7 E7 E7      Message header
00 11         bytecount = 17
DA DA DA      command header
01 00 01 D4 C3 OECP_EXT_TRANSID_TYPE 120003
02 03         OECP_COMMAND_ID_TYPE OECP_CMD_CONFIG_TYPE
03 02         OECP_SUB_COMMAND_ID_TYPE OECP_FORCE_RELEASE_CHANNEL
60 00 00 00 14 CONFIG_CHANNEL_MASK 0x14

```

If the command succeeds, the following response is sent:

```

D6 D6 D6      reply header
00 0E         byte count = 14
01 00 01 D4 C3 OECP_EXT_TRANSID_TYPE 120003
05 00 00      OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02         Error string length = 2
4F 4B        "OK"
04 01         OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_NONE

```

7.2 Channel Commands

7.2.1 RECORD CLIP

The following command records a 10 second VA clip on Channel A using DV50 compression.

```

E7 E7 E7          Message header
00 65            bytcount = 101
DA DA DA        command header
01 00 01 D4 C4  OECP_EXT_TRANSID_TYPE 120004
02 01           OECP_COMMAND_ID_TYPE OECP_CMD_CHANNEL_TYPE
03 0A           OECP_SUB_COMMAND_ID_TYPE OECP_RECORD_CLIP
10 00 00 00 01  CHANNEL_CHANNEL 1
1B             CHANNEL_RECORD_SPEC
20 00 02 49 F0  OECP_CLIP_ID 150000
21 00 00 02 58  OECP_CLIP_NFIELDS 600
22 05           OECP_CLIP_ALLOCTYPE ALLOCTYPE_RECORD
23 07 4D 79 20 43 6C 69 OECP_CLIP_TITLE "My Clip"
70
24 02 63 6A     OECP_CLIP_CREATOR "cj"
25 06 73 65 63 72 65 74 OECP_CLIP_PROJECT "secret"
26 00           OECP_CLIP_KEYWORDS ""
2C 00 00 00 00  OECP_CLIP_PURGE_DATE 0
27 00 00 00 00  OECP_CLIP_SRC_TC 0
11 00 00 00 00  OECP_CLIP_EDIT_IN_FIELD 0
31 01           OECP_CLIP_PLAYBACK_MODE PLAYBACK_MODE_FRAME_F12
32 01           OECP_CLIP_INTERP_MODE INTERPOLATOR_MODE_OFF
33 00           OECP_CLIP_REPEAT_CTRL_MODE REPEAT_CONTROL_OFF
34 00           OECP_CLIP_TC_PREF PLAYBACK_TC_PREF_CLIP
40 02           OECP_CLIP_TRACKS 2

41 00           OECP_CLIP_TRACK_NUMBER 0
42 06           OECP_CLIP_TRACK_TYPE TRACK_TYPE_DV50_VIDEO
43 00           OECP_CLIP_TRACK_VIDEO_STANDARD VIDEO_STANDARD_FPS60
45 00           OECP_CLIP_TRACK_VIDEO_RESOLUTION VIDEO_BITS_PER_PIXEL_8

41 01           OECP_CLIP_TRACK_NUMBER 1
42 08           OECP_CLIP_TRACK_TYPE TRACK_TYPE_DV_AUDIO
43 00           OECP_CLIP_TRACK_VIDEO_STANDARD VIDEO_STANDARD_FPS60
44 00           OECP_CLIP_TRACK_AUDIO_STANDARD AUDIO_STANDARD_AES
48 00 00 00 00  OECP_CLIP_TRACK_PHASE 0
49 0F           OECP_CLIP_TRACK_SUB_CHANNEL_MASK 0xf

```

When the record completes, the following response is sent.

```

D6 D6 D6        reply header
00 0E           byte count = 14
01 00 01 D4 C4 OECP_EXT_TRANSID_TYPE 120004
05 00 00        OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02           Error string length = 2
4F 4B          "OK"
04 01           OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_NONE

```

7.3 Database Commands

7.3.1 ABORT COPY CLIP

Abort a previous copy whose destination ClipID is 760100.

```

E7 E7 E7      Message header
00 11         bytecount = 17
DA DA DA     command header
01 00 01 D4 C5 OECP_EXT_TRANSID_TYPE 120005
02 04         OECP_COMMAND_ID_TYPE OECP_CMD_DB_TYPE
03 11         OECP_SUB_COMMAND_ID_TYPE OECP_ABORT_COPY_CLIP
7B 00 0B 99 24 DB_TOID 760100

```

If the command succeeds, the following response is sent:

```

D6 D6 D6      reply header
00 0E         byte count = 14
01 00 01 D4 C5 OECP_EXT_TRANSID_TYPE 120005
05 00 00      OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02         Error string length = 2
4F 4B        "OK"
04 01         OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_NONE

```

7.3.2 ABORT MOVE CLIP

Abort a previous move whose destination ClipID is 760100.

```

E7 E7 E7      Message header
00 11         bytecount = 17
DA DA DA     command header
01 00 01 D4 C6 OECP_EXT_TRANSID_TYPE 120006
02 04         OECP_COMMAND_ID_TYPE OECP_CMD_DB_TYPE
03 14         OECP_SUB_COMMAND_ID_TYPE OECP_ABORT_MOVE_CLIP
7B 00 0B 99 24 DB_TOID 760100

```

If the command succeeds, the following response is sent:

```

D6 D6 D6      reply header
00 0E         byte count = 14
01 00 01 D4 C6 OECP_EXT_TRANSID_TYPE 120006
05 00 00      OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02         Error string length = 2
4F 4B        "OK"
04 01         OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_NONE

```

7.3.3 CLIP EXISTS

Determine if the clip 760100 exists anywhere on the network.

```

E7 E7 E7      Message header
00 16         bytecount = 22
DA DA DA     command header
01 00 01 D4 C7 OECP_EXT_TRANSID_TYPE 120007
02 04         OECP_COMMAND_ID_TYPE OECP_CMD_DB_TYPE
03 00         OECP_SUB_COMMAND_ID_TYPE OECP_CLIP_EXISTS
70 00 0B 99 24 DB_CLIPID 760100
71 00 00 00 00 DB_HOSTID  netwide

```

If the clip is found on 192.20.202.251, the following response is sent back to the client:

```

D6 D6 D6      reply header
00 13         byte count = 19
01 00 01 D4 C7 OECP_EXT_TRANSID_TYPE 120007
05 00 00      OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02         Error string length = 2
4F 4B        "OK"
04 03         OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_HOSTID
71 C0 14 CA FB DB_HOSTID 192.20.202.251

```

7.3.4 COPY CLIP

Copy clip 17 to 760100.

```

E7 E7 E7      Message header
00 16         bytecount = 22
DA DA DA     command header
01 00 01 D4 C8 OECP_EXT_TRANSID_TYPE 120008
02 04         OECP_COMMAND_ID_TYPE OECP_CMD_DB_TYPE
03 0F         OECP_SUB_COMMAND_ID_TYPE OECP_COPY_CLIP
7A 00 00 00 11 DB_FROMID 17
7B 00 0B 99 24 DB_TOID 760100

```

When the clip has been copied, the server sends back

```

D6 D6 D6      reply header
00 0E         byte count = 14
01 00 01 D4 C8 OECP_EXT_TRANSID_TYPE 120008
05 00 00      OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02         Error string length = 2
4F 4B        "OK"
04 01         OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_NONE

```

7.3.5 DELETE CLIP

Delete clip 19 on host 192.20.202.251. But do not delete the clip if it is cued on a channel.

```
E7 E7 E7      Message header
00 1B         bytcount = 27
DA DA DA     command header
01 00 01 D4 C9 OECP_EXT_TRANSID_TYPE 120009
02 04         OECP_COMMAND_ID_TYPE OECP_CMD_DB_TYPE
03 0E         OECP_SUB_COMMAND_ID_TYPE OECP_DELETE_CLIP
70 00 00 00 13 DB_CLIPID 19
71 C0 14 CA FB DB_HOSTID 192.20.202.251
79 00 00 00 00 DB_CHANNEL_MASK 0
```

When the clip has been deleted, the server sends back

```
D6 D6 D6      reply header
00 0E         byte count = 14
01 00 01 D4 C9 OECP_EXT_TRANSID_TYPE 120009
05 00 00      OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02         Error string length = 2
4F 4B         "OK"
04 01         OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_NONE
```

7.3.6 FIND NEXT FREE CLIPID

Find the next clipID greater than or equal to 600 that is unused on the entire network.

```
E7 E7 E7      Message header
00 20         bytcount = 32
DA DA DA     command header
01 00 01 D4 CA OECP_EXT_TRANSID_TYPE 120010
02 04         OECP_COMMAND_ID_TYPE OECP_CMD_DB_TYPE
03 01         OECP_SUB_COMMAND_ID_TYPE OECP_FIND_NEXT_FREE_CLIPID
70 00 00 02 58 DB_CLIPID 600
71 00 00 00 00 DB_HOSTID netwide
72 00 00 00 01 DB_INCL_FLAG true
73 00 00 00 00 DB_WRAP_FLAG false
```

If 600 is free across the entire network, the reply is

```

D6 D6 D6      reply header
00 13         byte count = 19
01 00 01 D4 CA OECP_EXT_TRANSID_TYPE 120010
05 00 00      OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02         Error string length = 2
4F 4B         "OK"
04 02         OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_CLIPID
20 00 00 02 58 OECP_CLIP_ID 600

```

If no free IDs exist anywhere on the network (which is impossible...), the reply is

```

D6 D6 D6      reply header
00 13         byte count = 19
01 00 01 D4 CA OECP_EXT_TRANSID_TYPE 120010
05 00 00      OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02         Error string length = 2
4F 4B         "OK"
04 02         OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_CLIPID
20 00 00 00 00 OECP_CLIP_ID 0

```

7.3.7 GET CLIP INFO

Lookup the clip information of clip 6565 anywhere on the network.

```

E7 E7 E7      Message header
00 16         bytcount = 22
DA DA DA      command header
01 00 01 D4 CB OECP_EXT_TRANSID_TYPE 120011
02 04         OECP_COMMAND_ID_TYPE OECP_CMD_DB_TYPE
03 05         OECP_SUB_COMMAND_ID_TYPE OECP_GET_CLIP_INFO
70 00 00 19 A5 DB_CLIPID 6565
71 00 00 00 00 DB_HOSTID netwide

```

A reply might look something like this:

```

D6 D6 D6          reply header
00 A0            byte count = 160
01 00 01 D4 CB   OECP_EXT_TRANSID_TYPE 120011
05 00 00         OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02           Error string length = 2
4F 4B           "OK"
04 05           OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_CLIPINFO
71 C0 14 CA FB   DB_HOSTID 192.20.202.251
20 00 00 19 A5   OECP_CLIP_ID 6565
21 00 00 02 58   OECP_CLIP_NFIELDS 600
22 05           OECP_CLIP_ALLOCTYPE ALLOCTYPE_RECORD
23 0F 4D 79 20 54 69 74 OECP_CLIP_TITLE "My Title String"
6C 65 20 53 74 72 69 6E
67
24 09 4D 72 2E 20 41 63 OECP_CLIP_CREATOR "Mr. Accom"
63 6F 6D
25 05 4F 72 69 6F 6E   OECP_CLIP_PROJECT "Orion"
26 06 00 00 00 00 00 00 OECP_CLIP_KEYWORDS "\\0\0\0\0\0\0"

36 01           OECP_CLIP_SOURCE_DROPFRAME_PREFERENCE DROPFRAME_ON
27 00 00 00 00   OECP_CLIP_SRC_TC 0

37 01           OECP_CLIP_VITC_DROPFRAME_PREFERENCE DROPFRAME_ON
29 00 00 00 00   OECP_CLIP_VITC_TC 0

38 01           OECP_CLIP_TOD_DROPFRAME_PREFERENCE DROPFRAME_ON
28 00 00 00 00   OECP_CLIP_TOD_TC 0

34 01           OECP_CLIP_TC_PREF PLAYBACK_TC_PREF_SOURCE_TC

2A 00 00 00 00   OECP_CLIP_CREATION_DATE 0
2B 00 00 00 00   OECP_CLIP_LAST_ACCESS_DATE 0
2C 00 00 00 00   OECP_CLIP_PURGE_DATE 0
2D 00           OECP_CLIP_LOCK_STATE LOCK_CLIP_MODE_OFF
2E 00 00 00 00   OECP_CLIP_BROWSE_FIELD 0
2F 00 00 00 00   OECP_CLIP_REPEAT_IN_FIELD 0
30 00 00 02 58   OECP_CLIP_REPEAT_OUT_FIELD 600
31 00           OECP_CLIP_PLAYBACK_MODE PLAYBACK_MODE_FIELD
32 00           OECP_CLIP_INTERP_MODE INTERPOLATOR_MODE_ON
33 01           OECP_CLIP_REPEAT_CTRL_MODE REPEAT_CONTROL_LOOP
40 02           OECP_CLIP_TRACKS 2

41 00           OECP_CLIP_TRACK_NUMBER 0
42 05           OECP_CLIP_TRACK_TYPE TRACK_TYPE_DV25_VIDEO
43 00           OECP_CLIP_TRACK_VIDEO_STANDARD VIDEO_STANDARD_FPS60
45 00           OECP_CLIP_TRACK_VIDEO_RESOLUTION VIDEO_BITS_PER_PIXEL_8

41 01           OECP_CLIP_TRACK_NUMBER 1
42 08           OECP_CLIP_TRACK_TYPE TRACK_TYPE_DV_AUDIO
43 00           OECP_CLIP_TRACK_VIDEO_STANDARD VIDEO_STANDARD_FPS60
44 00           OECP_CLIP_TRACK_AUDIO_STANDARD AUDIO_STANDARD_AES
48 00 00 00 00   OECP_CLIP_TRACK_PHASE 0
49 FF           OECP_CLIP_TRACK_SUB_CHANNEL_MASK 0xff

```

7.3.8 MODIFY CLIP ATTRIBUTES

Modify clip 656565 to set the title to “Abekas 6000 Title,” the creator to “CJ,” and the project to “Orion.”

```

E7 E7 E7          Message header
00 34            bytecount = 52
DA DA DA        command header
01 00 01 D4 CC   OECF_EXT_TRANSID_TYPE 120012
02 04            OECF_COMMAND_ID_TYPE OECF_CMD_DB_TYPE
03 16            OECF_SUB_COMMAND_ID_TYPE OECF_MODIFY_CLIP_ATTRIBUTES
7A 00 0A 04 B5   DB_FROMID 656565
81 00 00 00 03   DB_COUNT_ATTRS 3
23 11 41 62 65 6B 61 73 OECF_CLIP_TITLE "Abekas 6000 Title"
20 36 30 30 30 20 54 69
74 6C 65
24 02 43 4A      OECF_CLIP_CREATOR "CJ"
25 05 4F 72 69 6F 6E OECF_CLIP_PROJECT "Orion"

```

If the command succeeds, the server will send back:

```

D6 D6 D6        reply header
00 0E           byte count = 14
01 00 01 D4 CC OECF_EXT_TRANSID_TYPE 120012
05 00 00        OECF_COMMAND_REPLY_ERC_TYPE OECF_ERROR_OK
00 02           Error string length = 2
4F 4B          "OK"
04 01           OECF_COMMAND_REPLY_TYPE OECF_CMD_REPLY_NONE

```

7.4 Report Commands

7.4.1 GET REPORT

7.4.1.1 Get Channel Status

Request the current channel status of Channel C.

```

E7 E7 E7      Message header
00 16         bytecount = 22
DA DA DA     command header
01 00 01 D4 CD OECP_EXT_TRANSID_TYPE 120013
02 07         OECP_COMMAND_ID_TYPE OECP_CMD_REPORT_TYPE
03 00         OECP_SUB_COMMAND_ID_TYPE OECP_GET_REPORT
B0 00 00 00 02 RPT_TYPE OECP_RPT_TYPE_CHANNEL_STATUS
B1 00 00 00 03 RPT_CHANNEL 3

```

If Channel C is currently in playback, the reply might be

```

D6 D6 D6      reply header
00 1A         byte count = 26
01 00 01 D4 CD OECP_EXT_TRANSID_TYPE 120013
05 00 00      OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02         Error string length = 2
4F 4B        "OK"
04 07         OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_REPORT_CHNL_STATUS
B5 01         RPT_NUM_BATCH_REPORTS 1
B1 00 00 00 03 RPT_CHANNEL 3
B6 00 00 00 04 RPT_CHANNEL_STATUS OECP_RPT_CHANNEL_STATUS_PLAY

```

7.4.1.2 Get Channel Timecode

Request the current timecode of Channel C.

```

E7 E7 E7      Message header
00 16         bytecount = 22
DA DA DA     command header
01 00 01 D4 CE OECP_EXT_TRANSID_TYPE 120014
02 07         OECP_COMMAND_ID_TYPE OECP_CMD_REPORT_TYPE
03 00         OECP_SUB_COMMAND_ID_TYPE OECP_GET_REPORT
B0 00 00 00 03 RPT_TYPE OECP_RPT_TYPE_CHANNEL_TIMECODE
B1 00 00 00 03 RPT_CHANNEL 3

```

If the timecode is 30

```

D6 D6 D6      reply header
00 1A         byte count = 26
01 00 01 D4 CE OECP_EXT_TRANSID_TYPE 120014
05 00 00      OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02         Error string length = 2
4F 4B        "OK"
04 08         OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_REPORT_CHNL_TIMECODE
B5 01         RPT_NUM_BATCH_REPORTS 1
B1 00 00 00 03 RPT_CHANNEL 3
B7 00 00 00 1E RPT_CHANNEL_TIMECODE 30

```

Request the current timecode of every channel in an eight channel system.

```

E7 E7 E7      Message header
00 16         bytecount = 22
DA DA DA      command header
01 00 01 D4 CF OECP_EXT_TRANSID_TYPE 120015
02 07         OECP_COMMAND_ID_TYPE OECP_CMD_REPORT_TYPE
03 00         OECP_SUB_COMMAND_ID_TYPE OECP_GET_REPORT
B0 00 00 00 03 RPT_TYPE OECP_RPT_TYPE_CHANNEL_TIMECODE
B1 00 00 00 00 RPT_CHANNEL OECP_ALL_CHNLS

```

If every channel has timecode 0, except channel C which has timecode 30...

```

D6 D6 D6      reply header
00 60         byte count = 96
01 00 01 D4 CF OECP_EXT_TRANSID_TYPE 120015
05 00 00      OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02         Error string length = 2
4F 4B        "OK"
04 08         OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_REPORT_CHNL_TIMECODE
B5 08         RPT_NUM_BATCH_REPORTS 8
B1 00 00 00 01 RPT_CHANNEL 1
B7 00 00 00 00 RPT_CHANNEL_TIMECODE 0
B1 00 00 00 02 RPT_CHANNEL 2
B7 00 00 00 00 RPT_CHANNEL_TIMECODE 0
B1 00 00 00 03 RPT_CHANNEL 3
B7 00 00 00 1E RPT_CHANNEL_TIMECODE 30
B1 00 00 00 04 RPT_CHANNEL 4
B7 00 00 00 00 RPT_CHANNEL_TIMECODE 0
B1 00 00 00 05 RPT_CHANNEL 5
B7 00 00 00 00 RPT_CHANNEL_TIMECODE 0
B1 00 00 00 06 RPT_CHANNEL 6
B7 00 00 00 00 RPT_CHANNEL_TIMECODE 0
B1 00 00 00 07 RPT_CHANNEL 7
B7 00 00 00 00 RPT_CHANNEL_TIMECODE 0
B1 00 00 00 08 RPT_CHANNEL 8
B7 00 00 00 00 RPT_CHANNEL_TIMECODE 0

```

7.4.2 REGISTER FOR REPORT

7.4.2.1 Register For Channel Status

Register to receive a notification when the status of Channel C changes.

```

E7 E7 E7      Message header
00 16         bytcount = 22
DA DA DA     command header
01 00 01 D4 D0 OECP_EXT_TRANSID_TYPE 120016
02 07         OECP_COMMAND_ID_TYPE OECP_CMD_REPORT_TYPE
03 01         OECP_SUB_COMMAND_ID_TYPE OECP_REGISTER_FOR_REPORT
B0 00 00 00 02 RPT_TYPE OECP_RPT_TYPE_CHANNEL_STATUS
B1 00 00 00 03 RPT_CHANNEL 3

```

If the registration succeeds, the node will send back

```

D6 D6 D6      reply header
00 0E         byte count = 14
01 00 01 D4 D0 OECP_EXT_TRANSID_TYPE 120016
05 00 00      OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02         Error string length = 2
4F 4B        "OK"
04 01         OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_NONE

```

If Channel C's status changes to playback, then a report will be issued.

```

DC DC DC      Report header
00 0C         bytcount = 12
06 01         OECP_ASYNC_REPORT_TYPE OECP_REPORT_CHNL_STATUS
B1 00 00 00 03 RPT_CHANNEL 3
B6 00 00 00 04 RPT_CHANNEL_STATUS OECP_RPT_CHANNEL_STATUS_PLAY

```

7.4.2.2 Register For Channel Timecode

Register to receive a notification when the timecode of Channel C changes.

```

E7 E7 E7      Message header
00 16         bytcount = 22
DA DA DA     command header
01 00 01 D4 D1 OECP_EXT_TRANSID_TYPE 120017
02 07         OECP_COMMAND_ID_TYPE OECP_CMD_REPORT_TYPE
03 01         OECP_SUB_COMMAND_ID_TYPE OECP_REGISTER_FOR_REPORT
B0 00 00 00 03 RPT_TYPE OECP_RPT_TYPE_CHANNEL_TIMECODE
B1 00 00 00 03 RPT_CHANNEL 3

```

If the registration succeeds, the node will send back

```

D6 D6 D6      reply header
00 0E         byte count = 14
01 00 01 D4 D1 OECP_EXT_TRANSID_TYPE 120017
05 00 00      OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02         Error string length = 2
4F 4B         "OK"
04 01         OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_NONE

```

If Channel C's timecode changes to 60, the following report is issued.

```

DC DC DC      Report header
00 0C         bytecount = 12
06 02         OECP_ASYNC_REPORT_TYPE OECP_REPORT_CHNL_TIMECODE
B1 00 00 00 03 RPT_CHANNEL 3
B7 00 00 00 3C RPT_CHANNEL_TIMECODE 60

```

7.4.2.3 Register For Clip Change Notification

Register to receive all clip change notifications.

```

E7 E7 E7      Message header
00 1B         bytecount = 27
DA DA DA      command header
01 00 01 D4 D2 OECP_EXT_TRANSID_TYPE 120018
02 07         OECP_COMMAND_ID_TYPE OECP_CMD_REPORT_TYPE
03 01         OECP_SUB_COMMAND_ID_TYPE OECP_REGISTER_FOR_REPORT
B0 00 00 00 01 RPT_TYPE OECP_RPT_TYPE_CLIP_CHANGE
B2 00 00 00 01 RPT_CHANGE_TYPE OECP_RPT_CLIP_CHANGE_TYPE_ANY
B3 00 00 00 00 RPT_CLIPID OECP_ALL_CLIPS

```

If the registration succeeds, the node will send back

```

D6 D6 D6      reply header
00 0E         byte count = 14
01 00 01 D4 D2 OECP_EXT_TRANSID_TYPE 120018
05 00 00      OECP_COMMAND_REPLY_ERC_TYPE OECP_ERROR_OK
00 02         Error string length = 2
4F 4B         "OK"
04 01         OECP_COMMAND_REPLY_TYPE OECP_CMD_REPLY_NONE

```

If clip 4 is deleted on node 192.20.202.251, the following notification will be sent.

```
DC DC DC      Report header
00 11         bytcount = 17
06 03         OECP_ASYNC_REPORT_TYPE OECP_REPORT_CLIP_CHANGE
B2 00 00 00 03 RPT_CHANGE_TYPE OECP_RPT_CLIP_CHANGE_TYPE_DELETED
B3 00 00 00 04 RPT_CLIPID 4
B4 C0 14 CA FB RPT_HOSTID 192.20.202.251
```

8 Revision History

8.1 Version 3.0

This document was provided as part of Version 3.0 System Software for the Abekas 6000. However, the API was significantly disfunctional as implemented in Version 3.0.

A few developers received beta copies of Version 3.5, which was later renamed Version 4.0 which had a fully functional API server.

8.2 Version 4.0

The `useTc` parameter of the `CUE CLIP` (see [Section 5.2.1 \[CUE CLIP\]](#), page 12) command is new. Since this parameter is optional, old code will still work.

Some beta versions of Version 3.5 and 4.0 contained a redundant copy of the lock attribute in the `REPLY CLIPINFO` (see [Section 6.5 \[REPLY CLIPINFO\]](#), page 44) response from the API server. This redundant field has been removed.

All tracks in `REPLY CLIPINFO` (see [Section 6.5 \[REPLY CLIPINFO\]](#), page 44) now contain a `OECF_CLIP_TRACK_VIDEO_STANDARD` attribute. Previously, audio tracks did not contain this attribute. This made it impossible to determine if an audio-only clip was formatted for 525 or 625 playback.

A new report subtype, `RPT_CLIP_CHANGE_TYPE_HEADER` has been added to the list of possible clip change types (see [Section 5.5.2 \[REGISTER FOR REPORT\]](#), page 37).

8.3 Version 4.0.3

The command `GET PARAM` (see [Section 5.5.4 \[GET PARAM\]](#), page 38) and its reply `REPLY PARAM` (see [Section 6.9 \[REPLY PARAM\]](#), page 53) have been added.

8.4 Version 4.5

The command `SET PARAM` (see [Section 5.5.5 \[SET PARAM\]](#), page 40) has been added. The command `GET PARAM` (see [Section 5.5.4 \[GET PARAM\]](#), page 38) has been added.

A scheme for determining the progress of a copy command has been provided. First, the command `COPY CLIP` (see [Section 5.3.6 \[COPY CLIP\]](#), page 25) now has a new optional argument `update` which causes the server to generate a `REPLY COPY START` (see [Section 6.10 \[REPLY COPY START\]](#), page 53) response in addition to the normal response to a copy command. The `REPLY COPY START` (see [Section 6.10 \[REPLY COPY START\]](#), page 53) response contains a `copyid` which is an identifier that can be used to poll the progress of the copy command. This identifier is used when sending a `GET COPY PROGRESS` (see [Section 5.3.21 \[GET COPY PROGRESS\]](#), page 33) command to obtain information regarding the current progress of the copy. In response to the `GET COPY PROGRESS` (see [Section 5.3.21 \[GET COPY PROGRESS\]](#), page 33) command,

the server will reply with either REPLY COPY PROGRESS (see [Section 6.11 \[REPLY COPY PROGRESS\]](#), page 54), REPLY COPYID NOT FOUND (see [Section 6.12 \[REPLY COPYID NOT FOUND\]](#), page 54), REPLY COPY DONE OK (see [Section 6.13 \[REPLY COPY DONE OK\]](#), page 54), or REPLY COPY DONE ERROR (see [Section 6.14 \[REPLY COPY DONE ERROR\]](#), page 54).

8.5 Version Unknown

Three commands have been removed: RECORD CLIP, RECORD NEXT CLIP, and RECORD PREVIOUS CLIP. These commands didn't work correctly. A better way of achieving the same results is to allocate a clip and then edit into the clip yourself.

The NetVideo Commands section of the manual has been removed since it was not known if the commands worked. FTP may be a more effective way of transferring material on and off the system anyway.

The VTR Commands section (see [Section 5.4 \[External VTR Commands\]](#), page 34) was added. All commands in this section are new.