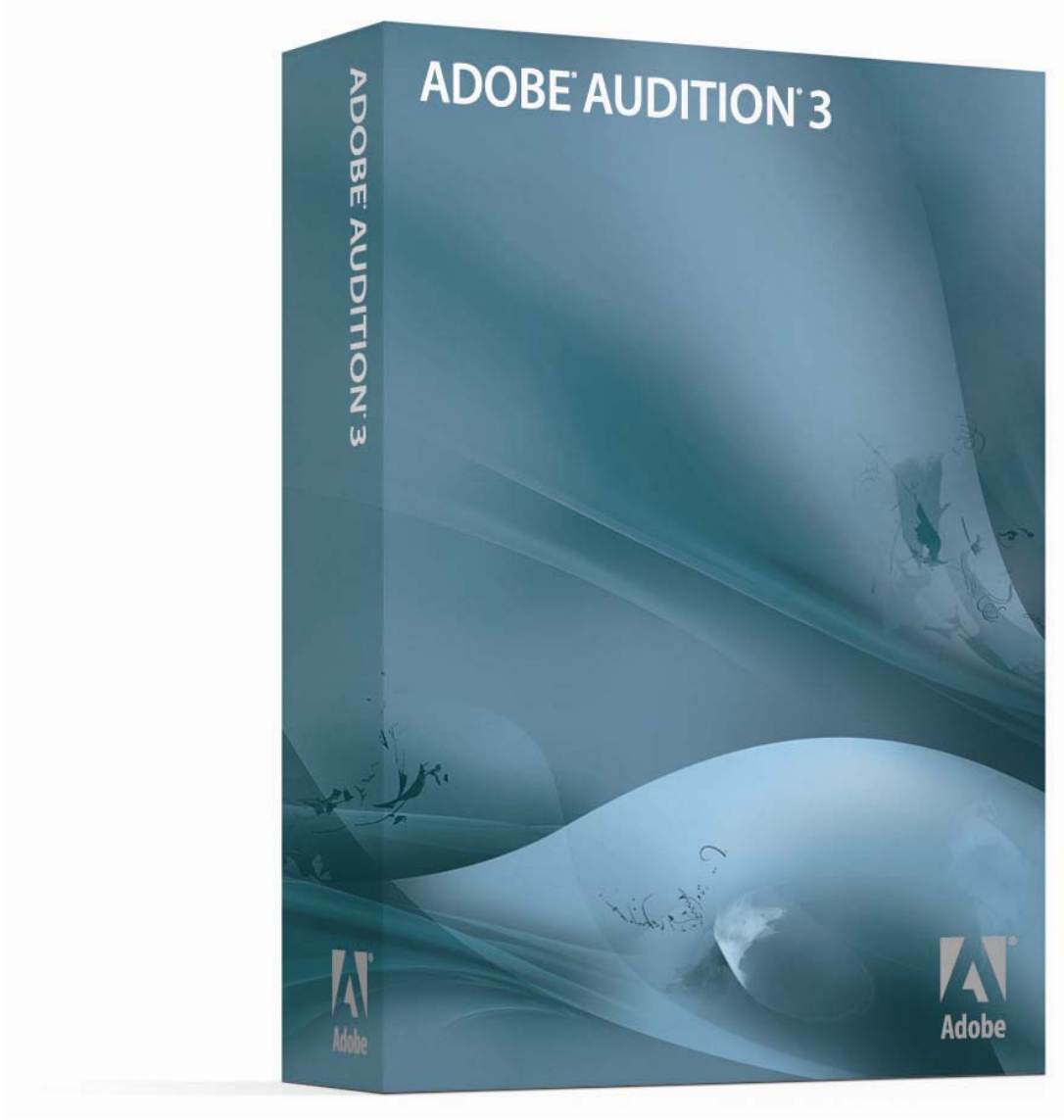

The External Controller (CV2) API

Version 3.0

The Adobe® Audition® Engineering team supplies the material in this document.

Notes on this release of Adobe® Audition® SDK

There have been relatively few content changes to the SDK sample code and documentation. References to “Cool” will persist, for the time being; after all, just because it’s an Adobe product doesn’t mean it’s not cool any more!

As we progress, we’ll rework the SDK, prioritizing our efforts based on developer requests. Please feel free to contact us with any API-related technical questions.

mailto:auditionsdk@adobe.com?subject=Audition_API_Question

The Adobe® Audition® Controller SDK is a set of APIs designed to allow controller device manufacturers to write controller DLLs (extension *.cv2) for use with Adobe® Audition®. Examples of such devices include external hardware devices with transport controls (Play, Stop, Rewind, etc.), USB audio devices with integrated faders and/or other controls, and custom hardware for recording studio or broadcast use of Adobe® Audition®.

The SDK consists of a small sample project and a single header file (ctrlmsg.h), which defines all the needed structures and types for you to communicate with Adobe® Audition®. The header is intended for use with C or C++ projects running on any Microsoft Windows™ operating system that is supported by Adobe® Audition®.

It is assumed that the reader understands how to create a DLL and export functions from that DLL. After writing your DLL as described below simply rename it from *.dll to *.cv2 and copy that file into your Adobe® Audition® working folder. Adobe® Audition® will automatically detect the presence of *.cv2 files on program startup.

Overview of functions called by Adobe® Audition®

QueryController Returns information about the external controller
OpenController Open this controller for active use
UpdateController Sends message in response to user actions in Adobe® Audition®
CloseController Close this controller

Making it all work

An Adobe® Audition® Controller DLL must implement and expose the following API functions:

BOOL QueryController(pDevInfo)	Required
DevInfo *pDevInfo	[OUT] Status information you must fill out for Adobe® Audition®

At startup, Adobe® Audition® will load your DLL and call **QueryController()** to get configuration details about your controller. This function may be called at any time (even when the controller has already been activated by **OpenController()**), so be sure not to use any resources that would interfere with standard use of your controller. It is necessary to release any resources you may have allocated in this function before returning.

After filling in the **DevInfo** structure with your information return TRUE if your device is connected and able to be started or FALSE, if not.

BOOL OpenController(pControllerFunctions)	Required
ControllerFunctions * pControllerFunctions	The complete set of available callbacks for the DLL

When Adobe® Audition® is ready to use your controller it will load your DLL and call **OpenController()** passing it a **ControllerFunctions** structure which contains a set of pointers to the accessor functions provided by Adobe® Audition®. You can use these function pointers (*pControlProc* & *pGetStatus*) to communicate with Adobe® Audition® (See Communicating with Adobe® Audition®, below). Your implementation of **OpenController()** should do any internal initialization that is needed. Any resources allocated in this function should be release in **CloseController()**. If your device is connected and all internal initialization has succeeded return TRUE. If not, return FALSE.

void UpdateController(dwMsg, pValue1, pValue2, pValue3)	Required
COOLEDIT_CTRLMSG dwMsg	A notification message (see Appendix B)
void *pValue1	Data dependant on the specific dwMsg sent
void *pValue2	Data dependant on the specific dwMsg sent
void *pValue3	Data dependant on the specific dwMsg sent

Adobe® Audition® will notify you of any changes made in the software by calling **UpdateController()**. For example, when the user changes the master volume control in Adobe® Audition®, **UpdateController()** will be

called to notify the device of this change. For a list of all **COOLEDIT_CTRLMSG**'s and their usage please see Appendix A.

void CloseController(bDisconnected)		Required
BOOL bDisconnected	TRUE if the device is already physically removed from the system	

Adobe® Audition® will call the function close the device either at program termination or when the user selects a different controller. You should do any internal cleanup that is needed before returning from this function.

Communicating with Adobe® Audition®

When your device receives any user action such as a button down or dial rotation, it will need to report messages to Adobe® Audition® by calling the *pControlProc*, which is passed to **OpenController()** at startup. Both *pGetStatus* and *pControlProc* can be used within in **OpenController()** and any other time until **CloseController()** has been called.

When the device needs to retrieve the status of certain controls or settings from Adobe® Audition® (such as the current active track, track volume, etc.) it can use the *pGetStatus*, which is passed to **OpenController()** at startup.

When Adobe® Audition® needs to update the controller of a user action it will call **UpdateController()** as documented above.

typedef void (*CONTROLLERMSGPROC)(message, iValue1, iValue2, iValue3)		CEP
COOLEDIT_CTRLMSG message	A notification message (see Appendix A)	
short iValue1	Data dependant on the specific message sent	
short iValue2	Data dependant on the specific message sent	
short iValue3	Data dependant on the specific message sent	

typedef void (*GETCTRLSTATUS)(message, piValue1, piValue2, piValue3)		CEP
COOLEDIT_CTRLMSG message	A notification (query) message (see Appendix A)	
void *piValue1	[OUT] Data dependant on the specific message sent	
void *piValue2	[OUT] Data dependant on the specific message sent	
void *piValue3	[OUT] Data dependant on the specific message sent	

Structures

```
typedef struct
{
        DWORD dwDevID;
                A unique number to identify the controller.  The number 0 is reserved for testing purposes and
                will work as long as only one test DLL is in use on your machine at a time.
        DWORD dwVersion;
                The version of the SDK that your DLL is designed to support.  Currently the only supported
                version is CTRL_SDK_VER.
        DWORD dwReserved;
                Reserved for future use and should not be modified by your DLL.
        char  szDevName[20];
                The short (null-terminated) description of your device, which Adobe® Audition® will show to
                the user in the “preferences\external controller” configuration dialog.
        DWORD dwNumTracks;
                How many tracks your device wants to “listen” to.  If you pass in 8, and the current session has
                17 tracks, then your controller will receive notifications (volume changes, etc.) for tracks 1
                thru 8, but not 9 thru 17.  If you pass in 20, and the current session has 17 tracks, then you will
                receive notification messages for all 17 tracks, and if the user creates more tracks, you will
                receive notifications for the first 3.
} DevInfo;

typedef struct
{
        // Core API
        CONTROLLERMSGPROC pControlProc;
        GETCTRLSTATUS pGetStatus;

        // Programmable API Extension
        GET_COMMAND GetCommand;
        LOOKUP_COMMAND_INDEX LookupCommandIndex;
        EXECUTE_COMMAND ExecuteCommand;
        int iCommandCount;
} ControllerFunctions;
```

Appendix A – Status & Control Messages

This documents the messages that are sent to Adobe® Audition's® CONTROLLERMSGPROC function. The messages can be passed directly through the API; however, we suggest you use the macros that are provided in the SDK and this documentation will cover those macros only.

For every macro you will need to pass the required function pointer as the first argument.

Status Messages

void GET_TRACK_SELECT(GETCTRLSTATUS fn, short *pTrack, short *pNumBlocks

Retrieves the currently selected track and the number of blocks in that track.

pTrack	A pointer to receive the currently selected track
pNumTrack	A pointer to receive the number of blocks in the active track or NULL if this value is not needed

void GET_TRACK_FADER(GETCTRLSTATUS fn, short track, double *pVolume)

Retrieves the volume setting of a track.

track	The track which this message should affect
pVolume	A pointer to receive the volume level. To get the decibel rating devide the returned value by 100.

void GET_TRACK_MUTE(GETCTRLSTATUS fn, short track, BOOL *pBool)

Retrieves the mute setting of a track.

track	The track which this message should affect
pBool	A pointer to receive the mute status. TRUE if mute is set.

void GET_TRACK_SOLO(GETCTRLSTATUS fn, short track, BOOL *pBool)

Retrieves the solo setting of a track.

track	The track which this message should affect
pBool	A pointer to receive the solo status. TRUE if solo is set.

void GET_TRACK_RECORDARM(GETCTRLSTATUS fn, short track, BOOL *pBool)

Retrieves the record setting of a track

track	The track which this message should affect
pBool	A pointer to receive the record status. TRUE if record is set.

void GET_TRACK_PAN(GETCTRLSTATUS fn, short track, short *pLeftRight)

Retrieves the pan setting of a track.

track	The track which this message should affect
pLeftRight	A pointer to receive the pan setting. PAN_LEFT is full left and PAN_RIGHT is full right.

void GET_TRACK_TITLE(GETCTRLSTATUS fn, short track, char *name)

Retrieves the text title of a track.

track	The track which this message should affect
name	A pointer to caller-allocated memory, which must be at least MAX_TRACK_TITLE characters.

void GET_TRACK_EQHIGHGAIN(GETCTRLSTATUS fn, short track, double *pVal)
void GET_TRACK_EQMIDGAIN(GETCTRLSTATUS fn, short track, double *pVal)
void GET_TRACK_EQLOWGAIN(GETCTRLSTATUS fn, short track, double *pVal)
void GET_TRACK_EQHIGHFREQ(GETCTRLSTATUS fn, short track, double *pVal)
void GET_TRACK_EQMIDFREQ(GETCTRLSTATUS fn, short track, double *pVal)
void GET_TRACK_EQLOWFREQ(GETCTRLSTATUS fn, short track, double *pVal)
void GET_TRACK_EQHIGHBAND(GETCTRLSTATUS fn, short track, double *pVal)
void GET_TRACK_EQMIDBAND(GETCTRLSTATUS fn, short track, double *pVal)
void GET_TRACK_EQLOWBAND(GETCTRLSTATUS fn, short track, double *pVal)

Retrieves the equalizer setting of a track.

track	The track which this message should affect
pVal	The current setting.

void GET_TRACK_DEVIN(GETCTRLSTATUS fn, short track, char *name)

Retrieves the input device of a track.

track	The track which this message should affect
name	A pointer to caller-allocated memory, which must be at least MAX_DEVICE_NAME characters.

void GET_TRACK_DEVOUT(GETCTRLSTATUS fn, short track, char *name)

Retrieves the output device of a track.

track	The track which this message should affect
name	A pointer to caller-allocated memory, which must be at least MAX_DEVICE_NAME characters.

void GET_DISPLAYTYPE(GETCTRLSTATUS fn, BOOL *pBoolMultitrack)

Retrieves the display mode for Adobe® Audition®

pBoolMultitrack	TRUE if multi-track display, FALSE if edit display

Controller Messages

void SET_DECK_REW(CONTROLLERMSGPROC fn, BOOL Pressed)

Rewinds the deck from the current position.

Pressed	TRUE to start rewinding, FALSE to stop.

void SET_DECK_FFW(CONTROLLERMSGPROC fn, BOOL Pressed)

Fast-Forwards the deck from the current position.

Pressed	TRUE to start fast-forwading, FALSE to stop.

void SET_DECK_PLAY(CONTROLLERMSGPROC fn)

Starts playback from the current position.

void SET_DECK_PLAYTOEND(CONTROLLERMSGPROC fn)

Starts playback from the current position and plays to the last block, then stops.

void SET_DECK_PLAYLOOPED(CONTROLLERMSGPROC fn)

Starts playback from the current position and plays continuously in a loop.

void SET_DECK_RECORD(CONTROLLERMSGPROC fn)

Starts recording from the current position.

void SET_DECK_TOGGLE_PAUSE(CONTROLLERMSGPROC fn)
Toggles playback pause on/off.

void SET_DECK_STOP(CONTROLLERMSGPROC fn)
Stops playback/recording.

void SET_DECK_TOGGLE_MONITOR(CONTROLLERMSGPROC fn)
Toggles the record level meters on/off.

void SET_DECK_SET_CUE(CONTROLLERMSGPROC fn)
Sets a cue mark at the current position.

void SET_DECK_LOCATE_RIGHT(CONTROLLERMSGPROC fn)
Sets the current position to the next cue mark to the right.

void SET_DECK_LOCATE_LEFT(CONTROLLERMSGPROC fn)
Sets the current position to the next cue mark to the left.

void SET_DECK_DIAL_UP(CONTROLLERMSGPROC fn)
Moves the current position forward in time.

void SET_DECK_DIAL_DOWN(CONTROLLERMSGPROC fn)
Moves the current position backward in time.

void SET_DECK_TOGGLE_METRONOME(CONTROLLERMSGPROC fn)
Toggles the playback metronome on/off.

Track Messages

All track messages require a (zero-based) track parameter to indicate which track should be affected.

void SET_TRACK_MUTE(CONTROLLERMSGPROC fn, short track)
Toggles the mute status on/off.

void SET_TRACK_FADER(CONTROLLERMSGPROC fn, short track, short volume)
Sets the volume.
volume The decibel value to set the volume multiplied by 100.

void SET_TRACK_FADER_F(CONTROLLERMSGPROC fn, short track, double volume)
Sets the volume.
volume The decibel value to set the volume.

void SET_TRACK_SOLO(CONTROLLERMSGPROC fn, short track)
Toggles the solo status on/off.

void SET_TRACK_SELECT(CONTROLLERMSGPROC fn, short track)
Sets the active track in the Adobe® Audition® multi-track display.
track A zero-based track to set active.

void SET_TRACK_FADER_TOUCH (CONTROLLERMSGPROC fn, short track, short touch)

For recordable track automation; signals a track that a fader is being touched (if touch is set to 1) or being released (if touch is set to 0)

void SET_TRACK_TOGGLE_RECORDARM(CONTROLLERMSGPROC fn, short track)

Toggles the record status on/off.

void SET_TRACK_PAN_SPIN(CONTROLLERMSGPROC fn, short track, short LeftRight)

Adjusts the pan settings relative to their current settings.

LeftRight <0 pan left, >0 pan right, 0 no change. The full range is -100 (full left) to 100 (full right)

void SET_TRACK_PAN_LEFT(CONTROLLERMSGPROC fn, short track)

Adjusts the pan settings to the left.

void SET_TRACK_PAN_RIGHT(CONTROLLERMSGPROC fn, short track)

Adjusts the pan settings to the right.

void SET_TRACK_EQMIDGAIN(GETCTRLSTATUS fn, short track, short delta)
void SET_TRACK_EQLOWGAIN(GETCTRLSTATUS fn, short track, short delta)
void SET_TRACK_EQHIGHFREQ(GETCTRLSTATUS fn, short track, short delta)
void SET_TRACK_EQMIDFREQ(GETCTRLSTATUS fn, short track, short delta)
void SET_TRACK_EQLOWFREQ(GETCTRLSTATUS fn, short track, short delta)
void SET_TRACK_EQHIGHBAND(GETCTRLSTATUS fn, short track, short delta)
void SET_TRACK_EQMIDBAND(GETCTRLSTATUS fn, short track, short delta)
void SET_TRACK_EQLOWBAND(GETCTRLSTATUS fn, short track, short delta)

Adjusts the equalizer settings of a track relative to their current settings.

delta The change to apply multiplied by 5.

Appendix B – Notification Messages

Notification message are those messages that are sent to the controller through the **UpdateController()** function.

CTRL_DECK_CLOCK – The current clock value. This value is exactly the same as Adobe® Audition® shows in the Time Window. It will therefore change format along with the display.

pValue1 – (char *) Pointer to a zero terminated string with the current song position.
pValue2 – Not used.
pValue3 – Not used.

CTRL_DECK_MASTERFADER – The current master fader value.

pValue1 – (double *) Pointer to the decibel value of this fader.
pValue2 – Not used.
pValue3 – Not used.

CTRL_DECK_MASTERFADER_TOUCH – Whether the master fader is touched.

pValue1 – (int) Non-zero if touching, zero if releasing.
pValue2 – Not used.
pValue3 – Not used.

CTRL_DECK_STOP – Sent when playback or recording has stopped.
CTRL_DECK_PLAYTOEND – Sent when the play to end button is pressed.
CTRL_DECK_PLAYLOOPED – Sent when the play looped button is pressed.

pValue1 – Not used.
pValue2 – Not used.
pValue3 – Not used.

CTRL_DECK_FFW – Set when the fast-forward button is pressed. Reset when FFW stops.
CTRL_DECK_REW – Set when the rewind button is pressed. Reset when REW stops.
CTRL_DECK_MONITOR – Set when the level meters are turned on. Reset when turned off.
CTRL_DECK_RECORD – Set when recording starts. Reset when recording is complete.
CTRL_DECK_PLAY – Set when the play button is pressed. Reset when playback is complete.
CTRL_DECK_PAUSE – Set when pause is pressed. Reset when playback/recording resumes.

pValue1 – (int) Zero if off or non-zero if on.

CTRL_REGISTRY_CHANGE – Sent after the user makes changes to the controllers settings dialog. The controller should refresh it's settings after receiving this message.

pValue1 – Not used.
pValue2 – Not used.
pValue3 – Not used.

CTRL_TRACK_PAN – The current pan value for this track.

pValue1 – (int) Track ID
pValue2 – (int) Pan value from PAN_LEFT to PAN_RIGHT

CTRL_TRACK_FADER – The current fader value for this track.

pValue1 – (int) Track ID
pValue2 – (double *) Pointer to the decibel value of this fader.
pValue3 – Not used.

CTRL_TRACK_FADER_TOUCH – Whether a fader is touched for automation.
pValue1 – (int) Track ID
pValue2 – (int) Non-zero if touching, zero if releasing.
pValue3 – Not used.

CTRL_TRACK_DEVIN – The current input device for this track.
CTRL_TRACK_DEVOUT – The current output device for this track.
pValue1 – (int) Track ID
pValue2 – (char *) Pointer to a zero terminated string with the current device name.
pValue3 – Not used.

CTRL_TRACK_SELECT – Sent when a new track is active.
pValue1 – (int) Track ID
pValue2 – (int) The number of blocks in this track.
pValue3 – Not used.

CTRL_TRACK_MUTE – The current mute status for this track.
CTRL_TRACK_SOLO – The current solo status for this track.
CTRL_TRACK_RECORD_ARM – Set if this track is set to record.
pValue1 – (int) Track ID
pValue2 – (int) Zero if off or non-zero if on.

CTRL_TRACK_RECORDLEVEL – The current recording level for this track. (Updates live while recording.)
pValue1 – (double *) Left channel level.
pValue2 – (double *) Right channel level.
pValue3 – (int)non-zero if clip indicator should be reset or zero otherwise.

CTRL_TRACK_TITLE – The new title for this track
Pvalue1 – (int) Track ID
pValue2 – (char *) Pointer to a zero terminated string with the current track title

CTRL_TRACK_INPUTGAIN – The input gain for this track
pValue1 – (int) Track ID
pValue2 – (double *) Pointer to the decibel input gain value for this track.
pValue3 – Not used.

CTRL_BANK_LEFTENABLE – Whether or not banking left is currently available
pValue1 – (int) non-zero if enabled, zero if not enabled
pValue2 – Not used.
pValue3 – Not used.

CTRL_BANK_RIGHTENABLE – Whether or not banking right is currently available
pValue1 – (int) non-zero if enabled, zero if not enabled
pValue2 – Not used.
pValue3 – Not used.

Programmable API Extensions Last updated January 24, 2003

We have extended the standard API to address the needs of controllers that allow for programmable functionality. The Programmable API Extensions can be used to trigger any of Adobe® Audition®'s built-in programmable events. The extensions are used only for configuring and trigger events. It does not replace the mechanisms that Adobe® Audition® uses to notify the controller of changes. To access the extended functionality you will need to understand the extra data that is in the **ControllerFunctions** structure (in bold below) that is passed to **OpenController()**.

```
typedef struct
{
        // Core API
        CONTROLLERMSGPROC pControlProc;
        GETCTRLSTATUS pGetStatus;

        // Programmable API Extension
        GET_COMMAND GetCommand;
        LOOKUP_COMMAND_INDEX LookupCommandIndex;
        EXECUTE_COMMAND ExecuteCommand;
        int iCommandCount;
} ControllerFunctions;
```

typedef UINT (*GET_COMMAND)(dwIndex, szCommandName, cbName, pdwCommandFlags)		CEP
DWORD dwIndex	Zero-based index into the Adobe® Audition® command list	
LPSTR szCommandName	[OUT] The text description of the requested command	
UINT cbName	The maximum bytes to store in szCommandName	
int *piUID	[OUT] The ID of the command.	
DWORD * pdwCommandFlags	[OUT] Switches applicable to the requested command	

Use this function to iterate through all of the available commands that Adobe® Audition® provides. These commands may change from version to version, so for complete compatibility you must not rely on any specific command being available.

Notes:
dwIndex is any value from zero (0) to *iCommandCount* - 1 (*iCommnadCount* is a member of the **ControllerFunctions** structure).

If *szCommandName* is zero the function returns the number of bytes needed to retrieve the entire string. If *szCommandName* is non-zero the functions returns the actual number of bytes copied to *szCommandName*.

cbName is the size in bytes of the buffer pointed to by *szCommandName*. If the buffer is too small the string will be truncated and zero-terminated.

piUID is the ID of the command. Note: For a partial list of command IDs, see the bottom of ctrlmsg.h.

pdwCommandFlags is used to indicate to usage of the indicated command. If Adobe® Audition® sets this to zero then this command is not intended for use by controllers.

typedef BOOL (*LOOKUP_COMMAND_INDEX)(szCommandName, pdwIndex)		CEP
int iUID	The ID of the command (See GET_COMMAND)	
DWORD * pdwIndex	[OUT] The actual index value to pass to EXECUTE_COMMAND	

Use this function to lookup the command index for a given command string. *iUID* is the integer-based ID of the command as returned from **GetCommand()**. *pdwIndex* is a pointer to a DWORD to receive the index from Adobe® Audition®. The value returned in *pdwIndex* can then be sent to **ExecuteCommand()** to actually take specific action. It is typical to store the IDs for the commands the user has assigned to any programmable buttons and look up the command indexes each time your DLL is started. Since the command indexes can

change from version to version you should not store the command indexes themselves in your configuration files. If the command is supported by Adobe® Audition®, this function will return TRUE; otherwise it will return FALSE.

typedef BOOL (*EXECUTE_COMMAND)(dwIndex, wKeyState)		CEP
DWORD dwIndex	A value retrieved from **LookupCommandIndex()**	
WORD wKeyState	CONTROLLER_KEYDOWN if the button is pressed, else CONTROLLER_KEYUP	

Call this to actually take a specific action.
This function will return TRUE if the command is handled successfully or FALSE otherwise.

void ConfigureController(hInstance, hwndParent, cCommands, pfnGetCommand)		OPTIONAL
HINSTANCE hInstance	Application instance handle	
HWND hwndParent,	Handle to the parent window in Adobe® Audition®	
int cCommands	Number of commands supported by Adobe® Audition®	
GET_COMMAND pfnGetCommand	Pointer used to retrieve command information	

If your DLL exports the **ConfigureController()** function it will enable the "Configure…" button in the "Device Properties\Ext. Controller" dialog. When the user pushes this button Adobe® Audition® will call your **ConfigController()** function to allow you to display any configuration window that suits your specific requirements. If you do not need this functionality you do not need to implement **ConfigureController()** in your DLL. A typical implementation will create a modal dialog box and use the *pfnGetCommand* function pointer to retrieve a list of available commands. The user can then assign commands to buttons on the controller and the controller can save the commands to the Windows™ registry for later use.