Engineering Acoustics Inc.

TDK API REFERENCE

Version 1.0.5.0

EAI
11/20/2015

CONTENTS

Windows Specific Notes 3

Required Files 3

Header Files 3

Interfacing with the TDK 3

Tactor Interface (Tactorinterface.h) 3

int InitializeTI(); 3

int ShutdownTI(); 3

const char* GetVersionNumber(); 4

int Connect(const char* name, int type, void* _callback); 4

int Discover(int type); 5

int DiscoverLimited(int type, int amount); 5

const char* GetDiscoveredDeviceName(int index); 6

int GetDiscoveredDeviceType(int index); 6

int Close(int deviceID); 6

int CloseAll(); 7

int Pulse(int deviceID, int tacNum, int msDuration, int delay); 7

int SendActionWait(int deviceID, int msDuration, int delay); 7

int ChangeGain(int deviceID, int tacNum, int gainval, int delay); 8

int ChangeFreq(int deviceID, int tacNum, int freqVal, int delay); 8

int RampGain(int deviceID, int tacNum, int gainStart, int gainEnd, int duration, int func, int delay); 9

int RampFreq(int deviceID, int tacNum, int freqStart, int freqEnd, int duration, int func, int delay); 9

int ChangeSigSource(int deviceID, int tacNum, int type, int delay) 10

int ReadFW(int deviceID); 10

int TactorSelfTest(int deviceID, int delay); 11

int ReadSegmentList(int deviceID, int delay); 11

int Stop(int deviceID, int delay) 12

int SetTactors(int deviceID, int delay, unsigned char* states) 12

int SetTactorType(int deviceID, int delay, int tactor, int type) 13

int UpdateTI (); 13

int GetLastEAIError(); 13

void SetTimeFactor(int value); 14

int BeginStoreTAction(int deviceID, int tacID); 14

int FinishStoreTAction(int deviceID); 14

int PlayStoredTAction(int deviceID, int delay, int tacID); 15

TAction Interface (TActionInterface.h) 16

int LoadTActionDatabase(char* tactionFile); 16

int IsDatabaseLoaded(); 16

int GetLoadedTActionSize(); 16

int GetTActionDuration(int tacID); 16

int UnloadTActions(); 17

char* GetTActionName(int tacID); 17

int PlayTAction(int boardID, int tacID, int tactorID, float gainScale, float freq1Scale, float freq2scale, float timeScale) 18

int PlayTActionToSegment(int boardID, int tacID, int tactorID, int segmentID, float gainScale, float freq1Scale, float freq2scale, float timeScale) 18

WINDOWS SPECIFIC NOTES

Required Files

The files required for TDK operation on Windows are as follows:

- eai_common.dll
- eai_serial.dll
- eai_winbluetooth.dll
- eai_winusb.dll
- SiUSBXp.dll
- TActionManager.dll
- TactorInterface.dll

Header Files

All of the functions included are declared in the TactorInterface.h and TActionInterface headers. Those functions are described below. Useful macros and definitions can be found in EAI_Defines.h.

Interfacing with the TDK

The easiest way to interface with these libraries is through the use of the header files included: TactorInterface.h and TActionInterface.h. If Microsoft Visual Studio is being used to build, those headers will automatically dllimport the functions. TactorInterface.lib and TActionInterface.lib are required to be statically linked to use this method. In order to link dynamically, use Window's LoadLibrary/GetProcAddress.

TACTOR INTERFACE (TACTORINTERFACE.H)

int InitializeTI();

Initializes the TDK.

PARAMETERS

N/A

RETURNS

On success: 0
On failure: -1, and sets GetLastEAIError();

int ShutdownTI();

Destroys all dynamic memory used by the Tactor Interface and TAction manager. It should be called when quitting the application.

PARAMETERS

N/A

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

const char* GetVersionNumber();

Gets a null-terminated string corresponding to the current version of the TDK.

PARAMETERS

N/A

RETURNS

A string representing the version of the TDK library.

int Connect(const char* name, int type, void* _callback);

Connects to a controller with the specified name and type. Sets a callback for incoming data.

PARAMETERS

IN: const char*	name	String representing the device to connect.
IN: int	type	Integer representing the type. See below.
IN: void*	callback	Function pointer to call on data received.

NOTES

The device type can be specified as follows. Note that all types are not supported on all platforms. See EAI_Defines.h for macros.

- 0: Serial
- 1: Windows USB
- 2: Android Bluetooth
- 3: Android USB

The callback function pointer should match the following signature:

```
void CallbackFunction(int board_id, unsigned char* data_buffer, int size);
```

RETURNS

On Success: A valid device ID handle, 0 or greater.
On failure: -1, and sets GetLastEAIError();

int Discover(int type);

Scans available ports for tactor controllers. See Notes for platform specifics.

PARAMETERS

IN: int	type	The type of device to scan for.

NOTES

On Windows and Linux, Discover will properly scan for tactor controllers and only return devices it's sure it can communicate with. On Android, Discover returns all attached devices, regardless if the TDK can communicate with it. For more information about the type parameter, see the Notes section of Connect();

RETURNS

On Success: The number of devices discovered.
On failure: -1, and sets GetLastEAIError();

int DiscoverLimited(int type, int amount);

Scans available ports for tactor controllers. If the amount given is reached then the discover stops. See Notes for platform specifics.

PARAMETERS

IN: int	type	The type of device to scan for.
IN: int	amount	The max amount of controllers to find.

NOTES

On Windows and Linux, Discover will properly scan for tactor controllers and only return devices it's sure it can communicate with. On Android, Discover returns all attached devices, regardless if the TDK can communicate with it. For more information about the type parameter, see the Notes section of Connect();

RETURNS

On Success: The number of devices discovered.
On failure: -1, and sets GetLastEAIError();

const char* GetDiscoveredDeviceName(int index);

Accesses the name of the discovered device at the given array index.

PARAMETERS

IN: int	index	The index of discovered device.

NOTES

When Discover returns non-zero, it can be used as an upper bound to get device names. The array is 0-based. For example, if the Discover returns 2, valid indexes for names are 0 and 1.

RETURNS

On Success: String representing the device's name.
On failure: NULL, and sets GetLastEAIError();

int GetDiscoveredDeviceType(int index);

Accesses the name of the discovered device at the given array index.

PARAMETERS

IN: int	index	The index of discovered device.

NOTES

When Discover Returns non-zero, it can be used as an upper bound to get devices names. The array is 0-based. For example, if the Discover Returns 2, valid indexes for names are 0 and 1.

RETURNS

On Success: int representing the device's type.
On failure: NULL, and sets GetLastEAIError();

int Close(int deviceID);

Closes connection to the given device.

PARAMETERS

```
IN: int                  deviceID     The device handle to close.
```

RETURNS

```
On Success: 0
On failure: -1, and sets GetLastEAIError();
```

int CloseAll();

Closes all open device connections.

PARAMETERS

N/A

RETURNS

```
On Success: 0
On failure: -1, and sets GetLastEAIError();
```

int Pulse(int deviceID, int tacNum, int msDuration, int delay);

Sends a pulse command to the tactor controller.

PARAMETERS

```
IN: int                  deviceID     The device receiving the command.
IN: int                  tacNum       The tactor to pulse.
IN: int                  msDuration   The duration of the pulse.
IN: int                  delay        How long to delay the pulse command.
```

RETURNS

```
On Success: 0
On failure: -1, and sets GetLastEAIError();
```

int SendActionWait(int deviceID, int msDuration, int delay);

Sends an action wait command to the tactor controller.

PARAMETERS

```
IN: int                  deviceID     The device receiving the command.
IN: int                  msDuration   The duration of the wait.
IN: int                  delay        How long to delay the wait command.
```

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

int ChangeGain(int deviceID, int tacNum, int gainval, int delay);

Sends a gain command to the tactor controller.

PARAMETERS

IN: int	deviceID	The device receiving the command.
IN: int	tacNum	The tactor to be gain adjusted.
IN: int	gainval	The new gain level.
IN: int	delay	How long to delay the gain command.

NOTES

A tacNum of 0 will modify the gain for all the tactors. Gain must be a number from 1 (Very low gain) to 255 (Maximum gain.)

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

int ChangeFreq(int deviceID, int tacNum, int freqVal, int delay);

Sends a frequency command to the tactor controller.

PARAMETERS

IN: int	deviceID	The device receiving the command.
IN: int	tacNum	The tactor to be frequency adjusted
IN: int	freqVal	The new frequency.
IN: int	delay	How long to delay the frequency command.

NOTES

A tacNum of 0 will modify the frequency for all the tactors. A tacNum of 255 (0xFF) will modify the frequency for the modulation value. Frequency must be a number from 300-3550.

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

```
int RampGain(int deviceID, int tacNum, int gainStart, int gainEnd,
                int duration, int func, int delay);
```

Sends a ramp command to the tactor controller, ramping the gain from start to end over duration. The Notes from ChangeGain above apply here.

PARAMETERS

IN: int	deviceID	The device receiving the command.
IN: int	tacNum	The tactor to be gain adjusted.
IN: int	gainStart	The start gain.
IN: int	gainEnd	The end gain.
IN: int	func	The function to use. (See Notes.)
IN: int	delay	How long to delay the ramp command.

NOTES

All Notes from ChangeGain apply here.

The function parameter chooses which function to interpolate along from start to end. Currently, the only function supported is Linear. Pass a value of TDK_LINEAR_RAMP. (See EAI_Defines.h)

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

```
int RampFreq(int deviceID, int tacNum, int freqStart, int freqEnd,
                int duration, int func, int delay);
```

Sends a ramp command to the tactor controller, ramping the frequency from start to end over duration. The Notes from ChangeFreq above apply here.

PARAMETERS

IN: int	deviceID	The device receiving the command.
IN: int	tacNum	The tactor to be frequency adjusted.
IN: int	freqStart	The start frequency.
IN: int	freqEnd	The end frequency.
IN: int	func	The function to use. (See Notes.)
IN: int	delay	How long to delay the ramp command.

NOTES

All Notes from ChangeFreq apply here.

The function parameter chooses which function to interpolate along from start to end. Currently, the only function supported is Linear. Pass a value of TDK_LINEAR_RAMP. (See EAI_Defines.h)

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

int ChangeSigSource(int deviceID, int tacNum, int type, int delay)

Sends a change signal source command to the tactor controller. The signal source is where the tactor gets its frequency information. See Notes for valid values.

PARAMETERS

IN: int	deviceID	The device to send the command to.
IN: int	tacNum	The tactor to change the signal source for.
IN: int	type	The new signal source.
IN: int	delay	How long to delay the command for.

NOTES

Type is a bitfield that takes any combination of the following values:

- TDK_SIG_SRC_PRIMARY (1) – Primary Tactor Frequency
- TDK_SIG_SRC_MODULATION (2) – Modulation Tactor Frequency
- TDK_SIG_SRC_NOISE (4) – Noise Frequency Generator.

See EAI_Defines.h for custom combination macros.

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

int ReadFW(int deviceID);

Sends a read firmware command to the tactor controller.

PARAMETERS

IN: int	deviceID	The device receiving the command.

NOTES

To get the value returned by this, you'll need to parse the packet returned by the device. The data values represent the version number in hexadecimal format.

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

int TactorSelfTest(int deviceID, int delay);

Sends a self test command to the tactor controller.

PARAMETERS

IN: int	deviceID	The device receiving the command.
IN: int	delay	How long to delay the command.

NOTES

The device will perform a self-test, and should not be interrupted until it's completed. When it's completed, a return packet will be sent by the device.

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

int ReadSegmentList(int deviceID, int delay);

Sends a read segment list command to the tactor controller.

PARAMETERS

IN: int	deviceID	The device receiving the command.
IN: int	delay	How long to delay the command.

NOTES

To get the value returned by this, the corresponding return packet from the device will need to be parsed. The device's response will have numbers representing the segments of tactors. The data bytes represent the end number for each segment of tactors. For example, a controller with three 8-tactor segments would have data bytes of: 8, 16, 24.

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

int Stop(int deviceID, int delay)

Sends a stop processing command to the device. This will halt any current commands, and remove any pending commands on the device.

PARAMETERS

IN: int	deviceID	The device receiving the command.
IN: int	delay	How long to delay the command.

NOTES

This will cause the device to immediately stop processing any commands it has received. Currently processing commands will be interrupted, and future commands will be aborted.

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

int SetTactors(int deviceID, int delay, unsigned char* states)

Sends a set tactors command to the device. This device will use the information in `states` to toggle tactors on or off.

PARAMETERS

IN: int	deviceID	The device receiving the command.
IN: int	delay	How long to delay the command.
IN: unsigned char*	states	The states of tactors 1-64. See notes.

NOTES

`states` is expected to be 8 consecutive bytes describing the desired states of tactors 1 through 64. Each bit in the bytes will be treated as a Boolean value whether or not to turn that given tactor on. Tactor 1 corresponds to the least significant bit of byte 1, whereas tactor 64 corresponds to the most significant bit of byte 8.

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

int SetTactorType(int deviceID, int delay, int tactor, int type)

Sends a command to the tactor controller to set a given tactor to a particular type. This command is notably for Universal controllers that have tactors that can be changed manually.

PARAMETERS

IN: int	deviceID	The device receiving the command.
IN: int	delay	How long to delay the command.
IN: int	tactor	The tactor to change the type of.
IN: int	type	The type to change the tactor to. (See Notes)

NOTES

The valid types for the type are defined in EAI_Defines.h. Those values are:

Tactor Type	Symbol	Value
C3	TDK_TACTOR_TYPE_C3	0x11
C2	TDK_TACTOR_TYPE_C2	0x12
EMS	TDK_TACTOR_TYPE_EMS	0x21
EMR	TDK_TACTOR_TYPE_EMR	0x22

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

int UpdateTI ();

Updates the internals of the tactor interface. This should be called once per frame, or at any regular interval.

PARAMETERS

N/A

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

int GetLastEAIError();

Gets the last set error value in the tactor interface.

PARAMETERS

N/A

NOTES

See EAI_Defines.h for a complete list of error return values.

RETURNS

The last set error value.

void SetTimeFactor(int value);

Sets the time multiplier passed with each command. This value is multiplied by the delays passed with commands to compute the full amount of time a command is delayed before it is processed. The default value is 10.

PARAMETERS

IN: int	value	The time factor (1-255).

RETURNS

N/A

int BeginStoreTAction(int deviceID, int tacID);

Sets the tactor controller in the ‘recording TAction’ mode. BeginStoreTAction should be coupled with FinishStoredTAction.

PARAMETERS

IN: int	deviceID	The device receiving the command.
IN: int	tacID	The address of the TAction to store (1-10)

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

int FinishStoreTAction(int deviceID);

Removes the tactor controller from ‘recording TAction’ mode. FinishStoredTAction should be coupled with BeginStoreTAction.

PARAMETERS

IN: int	deviceID	The device receiving the command.

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

```
int PlayStoredTAction(int deviceID, int delay, int tacID);
```

Plays a TAction stored on the tactor device.

PARAMETERS

IN: int	deviceID	The device receiving the command.
IN: int	delay	How long to delay the command.
IN: int	tacID	The address of the TAction to play (1-10)

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

TACTION INTERFACE (TACTIONINTERFACE.H)

int LoadTActionDatabase(char* tactionFile);

Opens and reads the given TAction database into memory.

PARAMETERS

IN: char* tactionFile The TAction database file to load.

RETURNS

On Success: The number of TActions successfully loaded.
On failure: -1, and sets GetLastEAIError();

int IsDatabaseLoaded();

Returns whether or not there is currently a database loaded.

PARAMETERS

N/A

RETURNS

On Success: 1 if there is a database loaded, otherwise 0.
On failure: -1, and sets GetLastEAIError();

int GetLoadedTActionSize();

Returns the number of currently loaded TActions.

PARAMETERS

N/A

RETURNS

On Success: The number of currently loaded TActions.
On failure: -1, and sets GetLastEAIError();

int GetTActionDuration(int tacID);

Gets the total time in milliseconds for the TAction to complete.

PARAMETERS

IN: int	tacID	The TAction to get the duration of.

RETURNS

On Success: The duration of the TAction, in milliseconds.
On failure: -1, and sets GetLastEAIError();

int UnloadTActions();

Unloads all currently loaded TAction databases.

PARAMETERS

N/A

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

char* GetTActionName(int tacID);

Gets a string representing the name of the given TAction.

PARAMETERS

IN: int	tacID	The TAction to get the name of.

RETURNS

On Success: The name of the TAction.
On failure: NULL, and sets GetLastEAIError();

int CanPlayTAction(int boardID, int tacID, int tactorID);

Returns 0 if the provided tacID can play on the provided boardID at the given tactorID.

PARAMETERS

IN: int	boardID	The device to attempt to map to.
IN: int	tacID	The TAction to attempt to map.
IN: int	tactorID	The Tactor on the controller to map to.

RETURNS

On Success: 0.
On failure: -1, and sets GetLastEAIError();

int PlayTAction(int boardID, int tacID, int tactorID, float gainScale, float freq1Scale, float freq2scale, float timeScale)

Plays a TAction.

PARAMETERS

IN: int	boardID	The device receiving the command.
IN: int	tacID	The TAction to play.
IN: int	tactorID	The Tactor to map from.
IN: float	gainScale	The value for scaling the gain commands.
IN: float	freq1Scale	The value for scaling the frequency commands.
IN: float	freq2Scale	Deprecated, unused.
IN: float	timescale	The value for scaling the start times.

NOTES

Scaling for Ramp commands is not yet implemented. If the TAction contains a ramp command, those values will not be scaled. For more information on the tactorID please review the TAction Mapping Specification document. Providing 0xDF TACTION_LOCATION_DEFAULT for the tactorID, will map the TAction how it is saved in the database.

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();

int PlayTActionToSegment(int boardID, int tacID, int tactorID, int segmentID, float gainScale, float freq1Scale, float freq2scale, float timeScale)

Plays a TAction to the given segment ID.

PARAMETERS

IN: int	boardID	The device receiving the command.
IN: int	tacID	The TAction to play.
IN: int	tactorID	The Tactor to map from.
IN: int	segmentID	The segment to map from.
IN: float	gainScale	The value for scaling the gain commands.
IN: float	freq1Scale	The value for scaling the frequency commands.
IN: float	freq2Scale	Deprecated, unused.
IN: float	timescale	The value for scaling the start times.

NOTES

command, those values will not be scaled. For more information on the tactorID and segmentID please review the TAction Mapping Specification document. Providing 0xDF

TACTION_LOCATION_DEFAULT for the tactorID or segmentID, will map the TAction how it is saved in the database.

RETURNS

On Success: 0
On failure: -1, and sets GetLastEAIError();