2017

MazeSuite API

Application Programming Interface (API) Manual

Adrian Curtin
Hasan Ayaz
DREXEL UNVIERSITY

Contents

Introduction

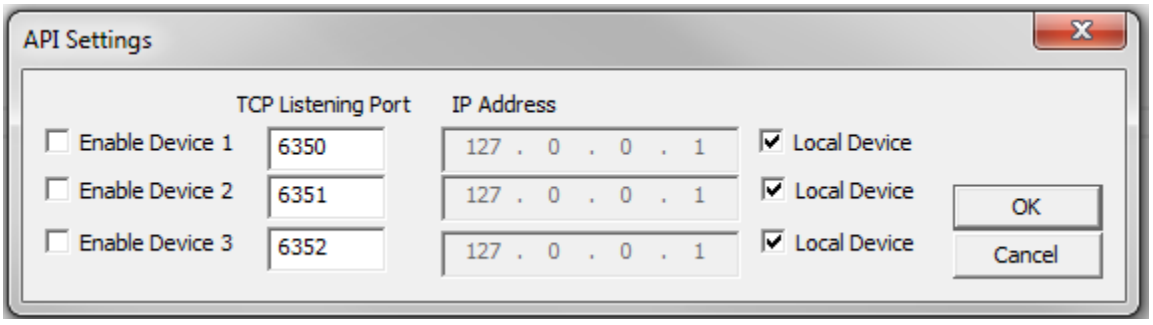
This document is intended to describe MazeSuite API (application programming interface) functions. For more information about MazeSuite, please see MazeSuite user manual at www.mazesuite.com

MazeSuite API allows any application to talk to and control MazeWalker scene rendering and actions. You can link the MazeSuite API DLL (Dynamic Link Library) File in your custom applications (your app) and the dll will communicate to MazeWalker over TCP/IP network to send and receive commands. Since the communication is over the network, your app can run on the same or different computer without changing anything.

The following section describes how to enable MazeSuite API functionality on MazeWalker and also describe the existing function calls in the MazeCOMM dll file that you can utilize from your app. Finally, the last chapter describes the TCP/IP network based communication protocol (byte stream) for cross platform projects.

MazeSuite API: MazeCOMM

All functions in the MazeCOMM API are contained within a mCOMMstr struct. The communications protocols are coded using WinSock2 libraries and use the TCP ports specified.



Software using the API DLL acts as a server to which MazeWalker Connects. To link MazeWalker to the desired application, the API must first be asked to Listen() to the appropriate port and the IP address and port number of the application must be provided to MazeWalker under the API Settings menu item.

The API DLL is written in C++, however functionality can also be used through C# through the definitions provided.

Connectivity Functions

int Connected();

Polls device and returns connection status

int Listen(int port);

Listens to TCP port for incoming packets and binds TCP for sending

int ListenFromParams();

Loads listening port from params.txt

Stored only as port number.

int ListenFromParams(char* filename);

Loads parameters from specified filename instead of params.txt

int Disconnect();

Disconnects and stops server

Movement Functions

Metered Movement (Discrete Steps)

int BoundedMovement(bool enable);

Enable or Disable bounded movement. Bounded movement prevents discrete steps (via API) from occurring that would not complete due to collision. If a particular action is blocked by a wall or other object and bounded movement is enabled, no action will occur. If bounded movement is disabled, the user will attempt to move but eventually collide with the obstacle. Bound movement is disabled by default.

int MoveForward(double dist);

Move forward *dist* maze units

int MoveBackward(double dist);

move Backward *dist* maze units

int LookLeft(double angle);

Look Left *angle* degrees

int LookRight(double angle);

Look Right *angle* degrees

int StrafeLeft(double dist);

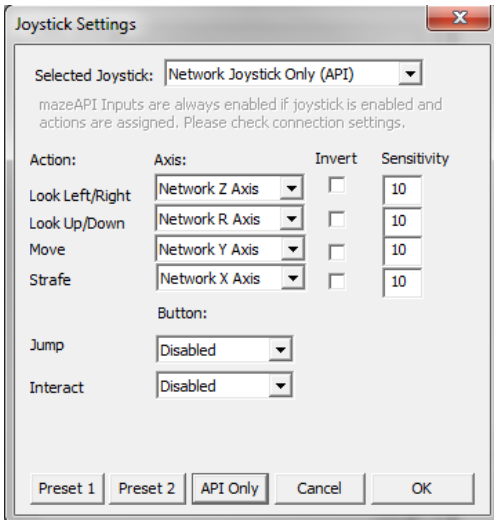
Strafe Left *dist* maze units

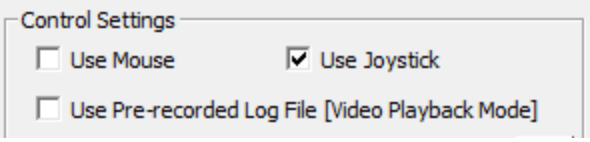
int StrafeRight(double dist);

Strafe Right *dist* maze units

Joystick Emulation (Analog Motion)

Joystick emulation over the API operates as a set of four virtual axes which can be set similar to physical joysticks in Windows. To operate joystick emulation over the API, a connected API must be enabled, “Use Joystick” must be selected from the main menu, and the joystick settings must be appropriately defined in Joystick Settings. Axes values follow joystick axes enumerations with 0 meaning Full Throttle, and 65536 Meaning Full reverse. Typically at idle joystick axes are valued at 32768 depending on calibration.





int SetNetJoystickX(int val);

Set the Network X axis to *val*. Axes values follow joystick axes enumerations with 0 meaning Full Throttle, and 65536 Meaning Full reverse. Typically at idle joystick axes are valued at 32768 depending on calibration.

int SetNetJoystickY(int val);

Set the Network Y axis to *val*. Axes values follow joystick axes enumerations with 0 meaning Full Throttle, and 65536 Meaning Full reverse. Typically at idle joystick axes are valued at 32768 depending on calibration.

int SetNetJoystickZ(int val);

Set the Network Z axis to *val*. Axes values follow joystick axes enumerations with 0 meaning Full Throttle, and 65536 Meaning Full reverse. Typically, at idle joystick axes are valued at 32768 depending on calibration.

int SetNetJoystickR(int val);

Set the Network R axis to *val*. Axes values follow joystick axes enumerations with 0 meaning Full Throttle, and 65536 Meaning Full reverse. Typically at idle joystick axes are valued at 32768 depending on calibration.

int SetNetJoystick(int netX,int netY,int netZ,int netR);

Simultaneously set the Network X axis to *netX*, the Network Y axis to *netY*, the Network Z axis to *netZ*, and the Network R axis to *netR*. Axes values follow joystick axes enumerations with 0 meaning Full Throttle, and 65536 Meaning Full reverse. Typically at idle joystick axes are valued at 32768 depending on calibration.

int ClearNetJoystick();

Clears and disables all current input to NetJoystick values.

Other Movement

int Jump();

MazeWalker avatar will jump.

Interaction Functions

int SetScore(int iArg);

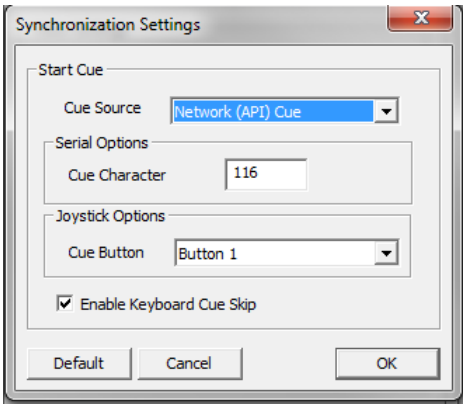
Set score in MazeWalker for activity bar with animation to slowly progress bar

int InstantSetScore(int iArg);

Set score in MazeWalker for activity bar with no animation

int PlayerInteract();

Temporarily sets score high enough to activate all dynamic objects with score/device activation requirements whose other requirements are already met. Identical to function of Interact button in MazeWalker.



Other Functions

int SendCue();

Sends cure to start mazelist or maze. Enable under Sync Settings Menu

int LogVal(int iArg);

Logs value iArg in MazeWalker log file at received time.

int LogString(char* input);

Logs string in MazeWalker log file at received time.

int SendAlert(char* input,int dispTime);'

Displays text message box to user in MazeWalker for specified period of time

int Pause();

Pause MazeWalker in interactive playback mode

int NextLevel();

Forces MazeWalker to load next level in list

int SetPosition(double xPos,double yPos, double zPos,double angle)

Sets current player position to xPos,yPos,zPos at an absolute angle

int GetPosition(double vec[])

Returns double[4] with xPosition, yPosition, zPosition, and current MazeTime

int SetPosX(double xPos)

Sets current player X position to x

int SetPosY(double yPos)

Sets current player Y position to y

int SetPosZ(double zPos)

Sets current player Z position to z

int SetRotation(double angle)

Sets current player view angle to absolute angle (degrees)

int SetMoveSpeed (bool setToDefault)

If true, then resets move speed to default value

int SetMoveSpeed (bool setToDefault, double moveSpeed)

If first argument is false, then set player move speed (while walking) to given argument in Maze Units per second (Mz/s)

int GetNextMarker(int *code, int* val)

Waits for and then returns code and value for event markers. These code and value pairs match pairs listed in the log files for events.

Low Level Functions

int Send(tcpMessage m);

Serializes tcpMessage into bytecode and sends packets to client. Returns the number of bytes sent.

int Send(int Command);

Send unmodified MazeWalker command

int Send(int Command, int iArg);

Send unmodified MazeWalker command and argument

int Send(int Command, int iArg, double dArgs[]);

Send MazeWalker command, integer argument and array of double arguments

int mCOMMstr::Send(int Command, int iArg, double dArg1,double dArg2, double dArg3,double dArg4);

Send MazeWalker command, integer argument and each individual double argument

tcpMessage Receive()

Receives tcpMessage struct from MazeWalker

Technical API

MazeWalker communicates through the tcpMessage struct

Messages are packaged as 44 byte low-endian format and are sent through TCP.

[0] - 7
[1] - 40
[2] - 40
[3-6] - Command (int)
[7-10] - iArg (int)
[11-18] - dArg[1] (double)
[19-26] - dArg[1] (double)
[27-34] - dArg[2] (double)
[35-42] - dArg[3] (double)
[43] - Checksum bit

MazeSuite expects TCP message in low endian format for communications. The message will contain a command number stored as an int, an argument stored as an int. and 4 secondary arguments stored as doubles. Each packet will be 40 bytes and are arranged serially.

Currently only the two int profiles are used, however the other components may be useful to send more detailed commands.

All of the character profile states (running, movingForwards etc) are now a new struct that also keeps track of the TCPIP commands. Currently movements are implemented in frame movements. Ex: 18, 10 will look left for 10 consecutive inputs.
These commands do not take priority over keyboard inputs. Holding forwards when signaling backwards will result in no net movement.

The TCP/IP cue has been implemented, but must be called after hitting start. or else the level simply counts the cue.

Messages returned form mazewalker are returned in the 44byte message struct using the command and iArg values listed in the log files.

Current Outbound MazeWalker Codes

Cue	20
Pause	10
Set Score	1
Instant Set Score	2
Forward	24
Backward	25
Jump	13
Strafe Left	26
Strafe Right	27
Look Left	28
Look Right	29
Log Value	-10
Log String	-100
Send Alert	-101
Next Level	-500
Set Position (x,y,z,angle)	-90
Set xPos	-91
Set yPos	-92
Set zPos	-93
Set Rotation	-94
Get Position	97
BoundedMovement	98
Player Interact	3
Set Net Joystick	400