289 lines (275 sloc) | 8.15 KB

```
'use strict';

const exec  = require('child_process').exec;
const stat = require('fs').stat;

const ErrorHandler = require('./error');

const util = {
	// Finds the correct command to start the IPC socket for mpv. It looks at the
	// output of 'mpv --version' and uses Regular Expressions to determine the mpv
	// version.
	// With mpv version 0.17.0 the command changed from '--input-unix-socket' to
	// '--input-ipc-server'
	//
	// @param options
	// options object
	//
	// @ return {promise}
	// Resolves to the command
	//
	findIPCCommand: function(options) {
		return new Promise((resolve, reject) => {

			// if the ipc Command was set by the user, use that
			if(options.ipc_command){
				// check if the command is correct
				if(!['--input-ipc-server', '--input-unix-socket'].includes(options.ipc_command)){
					reject(new ErrorHandler().errorMessage(1, 'start()', [options.ipc_command],
							// error message
							`"${options.ipc_command}" is not a valid ipc command`,
							// argument options
							{
								'--input-unix-socket': 'mpv 0.16.0 and below',
								'--input-ipc-server':  'mpv 0.17.0 and above'
							}
					));
				}
				else{
					resolve(options.ipc_command)
				}
			}
			// determine the ipc command according to the version number
			else{
				// the name of the ipc command was changed in mpv version 0.17.0 to '--input-ipc-server'
				// that's why we have to check which mpv version is running
				// asks for the mpv version
				exec((options.binary ? '"' + options.binary + '"' + ' --version' : 'mpv --version'), {encoding: 'utf8'}, (err, stdout, stderr) => {

					// if any error occurs reject it
					if(err){
						return reject(err);
					}

					// Version Number found
					if(stdout.match(/UNKNOWN/) == null){
						// get the version part of the output
						// looking for mpv 0.XX.Y
						const regex_match = (stdout.match(/(mpv) \d+.\d+.\d+/));

						if(regex_match){
							const match = regex_match[0]
							// split at the whitespace to get the numbers
							// split at the dot and look at the middle one to check for the
							// critical version number
							const versionNumber = parseInt(match.split(" ")[1].split(".")[1]);
							// Verison 0.17.0 and higher
							if(versionNumber >= 17){
								resolve('--input-ipc-server');
							}
							// Version 0.16.0 and below
							else{
								resolve('--input-unix-socket');
							}
						}
						// when MPV is built from source it sometimes has a git hash as
						// the version number
						// In this case assume it's a newer version and use the new command
						else{
```

```
                                                resolve("--input-ipc-server");
                                        }
                                }
                                // when compiling mpv from source the displayed version number is 'UNKNOWN'
                                // I assume that version that is compiled from source is the latest version
                                // and use the new command
                                else{
                                        resolve('--input-ipc-server');
                                }
                        })
                }
        });
    },
    // Chcks if the  binary passed in by the user actually exists
    // If nothing is passsed in the function is successfully resolved because
    // 'mpv' will be used
    //
    // @param binary {string}
    // Path to the mpv binary
    //
    // @return {pormise}
    //
    checkMpvBinary: function(binary) {
        return new Promise((resolve, reject) => {
                if(binary){
                        // check if the binary is actually working
                        stat(binary, (err, stats) => {
                                // check for the error
                                if(err && err.errno == -2){
                                        reject(new ErrorHandler().errorMessage(2, 'start()', [binary]));
                                }
                                else{
                                        resolve();
                                }
                        });
                }
                // if no binary is passed 'mpv' is used
                else{
                        resolve();
                }
        });
    },
    // Merges the options input by the user with the default options, giving
    // the user input options priority
    //
    // @param options
    // node-mpv options object input by the user
    //
    // @ return
    // Merged options object (UserInput with DefaultOptions)
    //
    mergeDefaultOptions: function(userInputOptions) {
        // the default options to start the socket with
        let defaultOptions = {
                debug: false,
                verbose: false,
                // Windows and UNIX defaults
                socket: process.platform === 'win32' ? '\\\\.\\pipe\\mpvserver' : '/tmp/node-mpv.sock',
                audio_only: false,
                auto_restart: true,
                time_update: 1,
                binary: null
        }

        // merge the default options with the one specified by the user
        return Object.assign(defaultOptions, userInputOptions);
    },
    // Determies the properties observed by default
    // If the player is NOT set to audio only, video properties are observed
    // as well
    //
    // @param adioOnlyOption
    // Flag if mpv should be started in audio only mode
    //
    // @return
    // Observed properties object
    //
    observedProperties: function(audioOnlyOption) {
        // basic observed properties
        let basicObserved = [
                'mute',
                'pause',
                'duration',
                'volume',
                'filename',
                'path',
                'media-title',
                'playlist-pos',
                'playlist-count',
                'loop',
        ];

        // video related properties (not required in audio-only mode)
        const observedVideo = [
                'fullscreen',
                'sub-visibility'
        ]

```

```
                return audioOnlyOption ? basicObserved : basicObserved.concat(observedVideo);
        },
        // Determines the arguments to start mpv with
        // These consist of some default arguments and user input arguments
        // @param options
        // node-mpv options object
        // @param userInputArguments
        // mpv arguments input by the user
        //
        // @return
        // list of arguments for mpv
        mpvArguments: function(options, userInputArguments) {
                // determine the IPC argument

                // default Arguments
                // --idle always run in the background
                // --msg-level=all=no,ipc=v  sets IPC socket related messages to verbose and
                // silence all other messages to avoid buffer overflow
                let defaultArgs = ['--idle', '--msg-level=all=no,ipc=v'];
                //  audio_only option aditional arguments
                // --no-video  no video will be displayed
                // --audio-display  prevents album covers embedded in audio files from being displayed
                if(options.audio_only){
                        defaultArgs = [...defaultArgs, ...['--no-video', '--no-audio-display']];
                }

                // add the user specified arguments if specified
                if(userInputArguments){
                        // concats the arrays removing duplicates
                        defaultArgs = [...new Set([...defaultArgs, ...userInputArguments])]
                }

                return defaultArgs;
        },
        // takes an options list consisting of strings of the following pattern
        //      option=value
        //   => ["option1=value1", "option2=value2"]
        // and formats into a JSON object such that the mpv JSON api accepts it
        //   => {"option1": "value1", "option2": "value2"}
        // @param options
        // list of options
        //
        // @return
        // correctly formatted JSON object with the options
        formatOptions: function(options) {
                // JSON Options object
                let optionJSON = {}
                // each options is of the form options=value and has to be splited
                let splitted = []
                // iterate through every options
                for(let i = 0; i < options.length; i++){
                        // Splits only on the first = character
                        splitted = options[i].split(/=(.+)/)
                        optionJSON[splitted[0]] = splitted[1]
                }
                return optionJSON;
        },
        // searches the function stack for the topmost mpv function that was called
        // and returns it
        //
        // @return
        // name of the topmost mpv function on the function stack with added ()
        // example: mute(), load() ...
        getCaller: function() {
                // get the top most caller of the function stack for error message purposes
                const stackMatch  = new Error().stack.match(/at\s\w*[^getCaller]\.\w*\s/g);
                const caller = stackMatch[stackMatch.length-1].split('.')[1].trim() + '()'
                return caller;
        },
        // extracts the protocol from a source string,e.g. http://someurl.com returns http
        // returns null if no protocol was found
        // @param source
        //              source string
        //
        // @return
        //              protocol string
        extractProtocolFromSource: function ( source ) {
                return !source.includes('://') ? null : source.split('://')[0];
        },
        // checks if a given protocol is supported
        // @param protocol
        //              protocol string, e.g. "http"
        //
        // @returns
        //              boolean if the protocol is supported by mpv
        validateProtocol: function( protocol ) {
                return [
                        "appending",
                        "av",
                        "bd",
                        "cdda",
                        "dvb",
                        "dvd",
                        "edl",
                        "fd",
                        "fdclose",
                        "file",
                        "hex",
```

```
                        "http",
                        "https",
                        "lavf",
                        "memory",
                        "mf",
                        "null",
                        "slice",
                        "smb",
                        "udp",
                        "ytdl"
                ].includes(protocol);
        }
}

module.exports = util;
```