SoundMex Documentation
Version 2.33.8

Sound-Toolkit for MATLAB®

http://www.soundmex.de

User Manual

HörTech
Kompetenzzentrum für
Hörgeräte-Systemtechnik

License agreement

IMPORTANT- PLEASE READ CAREFULLY:

BY INSTALLING THE SOFTWARE (AS DEFINED BELOW), COPYING THE SOFTWARE AND/OR CLICKING ON THE "ACCEPT" BUTTON BELOW, YOU (EITHER ON BEHALF OF YOURSELF AS AN INDIVIDUAL OR ON BEHALF OF AN ENTITY AS ITS AUTHORIZED REPRESENTATIVE) AGREE TO ALL OF THE TERMS OF THIS END USER LICENSE AGREEMENT ("AGREEMENT") REGARDING YOUR USE OF THE SOFTWARE. IF YOU DO NOT AGREE WITH ALL OF THE TERMS OF THIS AGREEMENT, DO NOT INSTALL AND/OR USE THE SOFTWARE.

DEFINITIONS

The term "Software" includes all software distributed with this License including all documentation. The "Software" is licensed to you under the terms specified in the License Grant below.

HIGH RISK ACTIVITIES

The Software is not fault-tolerant and is not designed, manufactured or intended for use as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or other medical devices, or weapons systems, in which the failure of the Software could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). HörTech gGmbH and its suppliers specifically disclaim any express or implied warranty of fitness for High Risk Activities.

OWNERSHIP AND COPYRIGHT

This Software is owned by HörTech gGmbH or its suppliers and is protected by copyright law and international copyright treaty. Therefore you must treat this Software like any other copyrighted material. You acknowledge that no title to the intellectual property in the Software is transferred to you. Title, ownership, rights, and intellectual property rights in and to the Software shall remain in HörTech gGmbH.

LICENSE GRANT

Subject to the license terms, HörTech gGmbH hereby grants you a non-exclusive, non-transferable (except under the terms below) license to install and to use the Software under the terms of this license. Except as provided in this license agreement, you may not transfer, rent, lease, lend, copy, modify, translate, sublicense, time-share or electronically transmit the Software. You may only either make one copy of the Software solely for backup or archival purposes or transfer the Software to a single hard disk provided you keep the original solely for backup or archival purposes. you agree not to modify the Software or attempt to decipher, de-compile, disassemble or reverse engineer the Software, except to the extent applicable laws specifically prohibit such restriction.

LICENSE TRANSFER

You may transfer your license and the rights granted in the license to a third party only if a) the third party agrees to this license agreement, b) you completely uninstall and delete all copies of this Software, c) all parts of the Software and its distribution are transferred to the third party and d) the transfer includes the current version and all prior versions of the Software.

DISCLAIMER OF WARRANTY

1 Contents

2 Introduction

SoundMex is a powerful tool for 16 and 32bit sound applications in MATLAB®. SoundMex is especially designed for acoustic measurement tasks (Psychoacoustics, Physical Acoustics, Neuro Physiology).

The software SoundMex is not a medical device or an accessory of a medical device, respectively, and not intended to drive a medical device.

The main features of SoundMex are:

- synchronous and asynchronous playback and recording
- real-time mixing to asynchronous playback
- sample-accurate synchronisation of multiple playback and recording devices
- realtime visualization of levels, time signals and spectra
- ringbuffer mode: generate sound data online (while device is already playing) and 'feed' new data to device
- record audio while playing and get recorded data directly to MATLAB®
- record from multiple devices synchronously to disk (hard disk recording)
- Realtime DSP-Plugin-Pipe with various DSP-plugins for real-time signal processing
- DSP-Plugin-SDK (Software Development Kit) for user defined realtime plugins for block by block wave and spectrum manipulation
    - script based plugins: user defined MATLAB® commands are called from the DSP-pipe. Manipulated data are written back to the pipe
    - Simple API for development of fast binary user plugins in any programming language
- direct I/O: recorded data can be directly (or after processing the data with DSP-Plugins) mapped to a wave out device for playback
- special command for 'highlighting' buttons on a MATLAB® window: sample accurate highlighting at desired playback 'positions' (for signal-synchronous user feedback)

A free version of SoundMex is available with the following restrictions:

- only a small subset of commands is available (see feature matrix below)
- only one device can be used at the same time. The first call to a command with a device parameter selects the device to use. A device with another ID can be used after the next call of soundmex2('init')

A demo version of SoundMex is available for free. The demo version has the following restrictions:

- a demo message is shown on 'init' and after every five minutes
- sound input and output is stopped every five minutes
- additional sound (the spoken words 'SoundMex demo') is added to all playbacks and recordings at random times

3 System Requirements

SoundMex runs on MATLAB® 5.3 (R11.1) and above on Microsoft Windows® 9x, Windows® ME, Windows® 2000 and Windows® XP. The computer should have at least 128 MB of RAM and a processor speed of 500MHz. The more power your computer has, the more tracks, devices and DSP-plugins can be handled simultaneously.

SoundMex runs with every sound card that supports the Microsoft Windows® standard multimedia interface MMSYSTEM (aka MME). However, HörTech cannot guarantee the support of all features with all sound cards. Especially for synchronizing multiple devices, the sound card itself must support this feature (see chapter 12.2 'Synchronizing devices').

For recommendations of sound cards please contact HörTech by email, fax or phone.

4 Installing SoundMex

Run SOUNDMEX.EXE and follow the installation instructions. The following files will be installed on your computer:

Directory	File(s)	Description

`BIN`	`SOUNDMEX2.DLL,`
	`SOUNDDLL2.DLL,`
	`SOUNDMEX.M`
	`SOUNDDLL2.INI,`
	`(SOUNDDLL2.LIC)`
		Main program files + INI-File (see below) + license file. Copy these files to any directory in your MATLAB® search path (or add the current directory to it)

`Windows-System-Directory`	`MMTOOLS25_C5.BPL`
	`VCL50.BPL`
	`VCLX50.BPL`
	`VCL50.DE`
		Runtime libraries. These files are copied to your Windows-system directory. Attention: copying these files to the MATLAB® search path or to your current working directory will not be sufficient to run SoundMex!

`EXAMPLES`	various	MATLAB® examples and wave files
`PLUGINS`	various	Plugins shipped with SoundMex
`DSP-SDK` (on request)	various	Documentation, examples and sample C-Code for the DSP-Software Development Kit

`MANUAL`	`SOUNDMEX2.PDF`
	`SOUNDMEX2 PLUGINS.PDF`
		PDF Help

`MANUAL/HTML`	`SOUNDMEX2.HTML` + related files	HTML help for access from MATLAB®. Copy HTML and subfolder 'SoundMex2-Files' to your MATLAB® ref path (see below)

After installing you should add the BIN directory of your SoundMex installation (e.g. C:\SoundMex2\bin) to your MATLAB® path.

If you want to access the HTML-help through the MATLAB® command 'doc SoundMex2' please copy the content of the subfolder 'Manual\HTML' from your SoundMex-Installation to the directory

```
<MATLAB® path>\help\techdoc\ref   (e.g C:\MATLAB®6p5\help\techdoc\ref)
```

You can set the temporary path of SoundMex (path were recorded data will be stored) by modifying the path variable in the file SOUNDDLL2.INI (or in your user defined INI-File, see command 'init').

SOUNDMEX.M is a helper file for translating SoundMex version 1.x syntax to SoundMex version 2.x syntax. Use this script to use your old MATLAB® scripts using SoundMex version 1.x. Please refer to chapter 6 'Compatibility with SoundMex v 1.x'

It is not recommended to use this script if not necessary. Use the new syntax wherever it is possible.

Caution: Never run SoundMex from network drives or set its temporary path to a network drive!

NOTE: if you are running MATLAB 7.1 or above and you want to avoid the warning 'soundmex2.mexw32 now shadows soundmex2.dll' you have to rename soundmex2.dll in the BIN directory of your SoundMex installation!

5 Uninstalling SoundMex

An uninstaller is shipped with SoundMex. Select 'Uninstall SoundMex' from the start menu or run the uninstaller from the control panel.

The copy protection driver will not be removed by this uninstaller to avoid problems with other dongles.

If you want to remove the dongle driver manually, please perform the following steps

If a version of SoundMex v1.18 or below or SoundMex 2.08 or below was installed on your system please call the command

hinstall -r

from the command prompt in the HASP-subdirectory of your SoundMex installation (e.g. c:\soundmex\hasp), or if this directory does not exist form your system directory (e.g. c:\winnt\system32). After the command has completed the file HINSTALL.EXE can be deleted.

For removing the dongle driver for SoundMex v1.19 or above or SoundMex v2.09 or above please call the command

haspdinst -r

from the command prompt in the HASP-subdirectory of your SoundMex installation (e.g. c:\soundmex2\hasp). After the command has completed the file HASPDINST.EXE can be deleted.

6 Compatibility with SoundMex v 1.x

The command interface of SoundMex version 2.x was completely rewritten to get a more flexible and 'MATLAB®'-like syntax. Thus it is completely incompatible with the syntax of SoundMex version 1.x. There exist two methods to use old scripts:

1. Usage of the syntax translation script shipped with SoundMex 2.x: in the BIN directory of SoundMex 2.x you can find the script SOUNDMEX_1.M that tries to translate calls from the old syntax into the new syntax and calls SoundMex 2.x. If you want to use this translation script with your old applications, please rename it to SOUNDMEX.M. The script will show a warning on the first call to indicate that the script is used rather than the old binary version SOUNDMEX.DLL at the moment.
   It is highly recommended to use the new syntax. The translation script was tested very

well but might not work correct under all circumstances. Use the script only for running old existing scripts that cannot be translated to new syntax.

2. Parallel use of SoundMex 1.x and SoundMex 2.x: if you want to use both binary versions of SoundMex on one computer, you have to be sure that calls to SOUNDMEX will not call the MATLAB® syntax translation script SOUNDMEX_1.M shipped with SoundMex 2.x (see above). Otherwise MATLAB® might use the script rather than the old binary version SOUNDMEX.DLL. It is recommended to use SoundMex 2.x for new scripts since SoundMex 1.x is not developed or maintained any longer!

Please note that many commands are obsolete with the new syntax, because some commands were amalgamated and the 'differences' are passed as parameters, e.g.:

Old syntax:
```
     soundmex('getrecbuffer')
     soundmex('getplaybuffer')
```

New Syntax:
```
     soundmex2('getbuffer')                   % waveout is default!
     soundmex2('getbuffer', 'type', 'wavein')
```

7 Command Description

All SoundMex 2.x commands have a similar syntax:

```
[errocode, outarg1,…] = soundmex2('command', par1, val1, par2, val2,…);
```

All command arguments (except for 'help' command) have to be specified in pairs of 'parameter name' and 'value', e.g.

```
soundmex2('playfile',                          ...  % command name
          'filename', 'noise_16bit.wav',       ...  % name of wavefile
          'loopcount', 1,                      ...  % play it 1 time
          'device', 2                          ...  % DeviceId.
);
```

Most of the parameters have default values, some are mandatory (see help on commands below).

All commands return one output argument at least. This return value error code indicates success or failure of the command itself, where 1 is returned on success and 0 on any error. Evaluate this first return value to react on any error before calling further SoundMex 2.x commands, e.g.:

```
success = soundmex2('init');
if (~success)
   error('Cannot initialize SoundMex2! ');
end
```

Commands with additional return values write these to outarg[1] … outarg[n], e.g.:

```
[success, clipcount] = soundmex2('hasclipped');
if (~success)
   error('Error in command hasclipped');
end
disp(['playback has clipped ' num2str(clipcount) ' times'];
```

```
[Success, NumBuf, BufSiz] = soundmex2('getbuffer');
if (~Success)
   error('error in command getbuffer');
end
disp(['device has ' num2str(NumBuf) ' buffers of ' num2str(BufSiz) ' bytes size']);
```

The tables below show a list of all available commands sorted by functionality. The following abbreviations are used in the column 'Description':

Name>	Name of the command
Help>	Help text
Par.>	Parameter list
Def.>	Default values of Par.>
Ret.>	Return values. The return values are semicolon separated, where each value is returned as separated outarg!

In the column 'Examples' you can find a list of example scripts that are related to the specified command.

NOTE: if any error occurs within a command, all outargs are filled with the number zero and only the error code is valid!!

Note: The standard error code return argument is omitted in the table! All commands return the error code as first outarg!

7.1.1 General Commands

Command	Description	Examples
help	Name> help Help> prints help on command or command list Par.> command name. Without argument a command list is printed.	
version	Name> version Help> returns version string Ret.> version string	
init	Name> init Help> initializes SoundMex 2.x Par.> inifile: name of inifile to use force: 0 or 1. 1 forces exit prior to init mode: 'free' to switch to free mode Def.> inifile: MEXBINPATH\SOUNDDLL2.INI force: 0 mode: none Ret.> LicenceType;Version;LicenceOwner	all
isinitialized	Name> isinitialized Help> determines if module is initialized Ret.> '1' if initialized, '0' else	
exit	Name> exit Help> de-initializes SoundMex 2.x	
show	Name> show Help> shows visualization and info on devices Par.> wavein: string with comma delimited wavein devices to show waveout: string with comma delimited waveout devices to show Def.> shows all devices	device_info.m mark_buttons.m sync_play.m
hide	Name> hide Help> hides visualization and info	
getlasterror	Name> getlasterror Help> retrievs last errors occurred in SOUNDMEX2.DLL and SOUNDDLL2.DLL Ret.> error string	error_handling.m
showerror	Name> showerror Help> sets error printing behaviour of SoundMex 2.x Par.> mode: 'on': all errors are printed to MATLAB® workspace 'off': no errors are printed to MATLAB® workspace Def.> mode: 'on'	error_handling.m

7.1.2 Playback Commands

Command	Description	Examples
playfile	Name> playfile	calculate_while_playing.m mark_buttons.m modulate.m record.m
	Help> plays file on a waveout device	
	Par.> device: ID of device to use for playback	
	filename: file to play	
	bitlength: bitlength for playback (16 or 32 only. Use 32 for 24bit-devices)	
	loopcount: loopcount for playback. 0 plays in endless loop	
	mode: two modes:	
	'sync': functions returns after playback is complete	
	'async': functions returns immediately	
	prepare: 0 or 1. If 1, the device is _not_ started but opened. Use prepare flag to prepare multiple devices for synchronization and start all prepared devices with 'startprepared'	
	random: 0 or 1: If set to 1 playback is started at a random position within the file. NOTE: the file is played from this random position until the end for the actual loop! If e.g. the loopcount is set to 3, the file is played two times in full length and one time from the random position to the end!	
	startpos: playback is started at this sample position within the file. If startpos is specified, the random parameter is ignored! The startpos must be within the length of the file. NOTE: the file is played from this position until the end for the actual loop! If e.g. the loopcount is set to 3, the file is played two times in full length and one time from that position to the end!	
	fade: length of linear fade in ramp in samples.	
	mute: mutes first n samples. If mute is specified the fade command is ignored	
	Def.> device: 0	
	filename: mandatory	
	bitlength: bitlength of file itself	
	loopcount: 1	
	mode: 'async'	
	prepare: 0	
	random: 0	
	fade: 0	
	mute: 0	
filetofile	Name> filetofile	
	Help> pumps file through DSP-pipe to file.	
	Par.> infile: name of input file	
	outfile: name of output file	
	bitlength: bitlength for output (16 or 32 only). Use 32 for 24bit-devices)	
	Def.> infile: mandatory	
	outfile: mandatory	
	bitlength: bitlength of file itself	
playmem	Name> playmem	sync_play.m
	Help> plays vector on a waveout device	rawmode.m
	Par.> device: ID of device to use for playback	
	data: normalized vector with one (mono) or two (stereo) columns	
	bitlength: bitlength for playback (16 or 32 only). Use	

	32 for 24bit-devices) sampfreq: sampling frequency for playback loopcount: loopcount for playback. 0 plays in endless loop mode: two modes: 'sync': functions returns after playback is complete 'async': functions returns immediately prepare: 0 or 1. If 1, the device is _not_ started but opened. Use prepare flag to prepare multiple devices for synchronization and start all prepared devices with 'startprepared' fade: length of linear fade in ramp in samples. mute: mutes first n samples. If mute is specified the fade command is ignored Def.> device: 0 data: mandatory bitlength: 16 sampfreq: 44100 loopcount: 1 mode: 'async' prepare: 0 fade: 0 mute: 0	
memtofile	Name> memtofile Help> pumps vector through DSP-pipe to a file Par.> data: normalized vector with one (mono) or two (stereo) columns bitlength: bitlength for output file (16 or 32 only) sampfreq: sampling frequency for playback outfile: name of output file Def.> data: mandatory bitlength: 16 sampfreq: 44100 outfile: mandatory	
startprepared	Name> startprepared Help> starts all devices previous prepared using 'playfile', 'playmem', 'record', 'startio' or 'playringbuffer'. The type 'all' is the same as 'io': it starts both (wavein and waveout) device types. Par.> type: 'waveout', 'wavein', 'io' or 'all' Def.> type: 'waveout'	sync_play.m sync_record.m sync_check.m
isplaying	Name> isplaying Help> determines if a device is playing Par.> device: ID of device to query Def.> device: 0 Ret.> 1 if device is playing, 0 else	calculate_while_playing.m ringbuffer.m
playposition	Name> playposition Help> determines playback position of a device Par.> device: ID of device to query Def.> device: 0 Ret.> actual playing position of device in samples	
mixfile	Name> mixfile Help> mix file to a device that is already playing sound Par.> device: ID of device to mix to filename: file to mix mode: two modes: 'add': adds files samples to playing samples 'multiply': multiplies playing samples with files samples Def.> device: 0	mark_buttons.m

	filename: mandatory mode: 'add'	
mixmem	Name> mixmem Help> mixes vector to a device that is already playing sound Par.> device: ID of device to use for playback data: normalized vector with one (mono) or two (stereo) columns mode: two modes: 'add': adds file's samples to playing samples 'multiply': multiplies playing samples with file's samples Def.> device: 0 channels: 2 mode: 'add'	modulate.m record.m
ismixing	Name> ismixing Help> determines if a device is mixing Par.> device: ID of device to query Def.> device: 0 Ret.> 1 if device is mixing, 0 else	mark_buttons.m modulate.m record.m
mixposition	Name> mixposition Help> determines mixing position of a device ATTENTION: ismixing will return 1 if a mix is 'pending' (data are mixed actually, but the mixed buffers have not arrived at the device yet, and so the mixed data are not audible yet). Before the mixed output is audible 'mixposition' will return negative values! Par.> device: ID of device to query Def.> device: 0 Ret.> actual mixing position in samples	record.m
stopplay	Name> stopplay Help> stops playback of a device Par.> device: ID of device to stop fade: length of fade out ramp in samples. NOTE: when calling stopplay with a ramp, there are still some non-ramped filled buffers present that were already sent to the driver. Therefore the ramp will not start immediately (delay depends on your buffer settings, see command setbuffer). The command will return after the fadeout is complete, so multiple devices cannot be ramped down simultaneously. Def.> device: 0 fade: 0	calculate_while_playing.m
stopplayall	Name> stopplayall Help> stops playback of all devices	mark_buttons.m modulate.m
stopmix	Name> stopmix Help> stops mixing of a device Par.> device: ID of device to stop Def.> device: 0	
stopall	Name> stopall Help> stops playback and recording of all devices	plugin.m record.m start_io.m
hasclipped	Name> hasclipped Help> determines if a device has clipped Par.> device: ID of device to query Def.> device: 0 Ret.> number of clipped samples since the last start of the device	calculate_while_playing.m mark_buttons.m
setplaymode	Name> setplaymode Help> sets playback mode for all waveout devices Par.> mode: 'safe': data will be copied in memory. 'raw': fast, but not safe: vectors to DLL are played directly using original data	rawmode.m

	pointer. Manipulating that vector during playback will result in an access violation!! Def.> mode: 'safe'	
getvolume	Name> getvolume Help> determines actual volume of a device Par.> device: ID of device to query Def.> device: 0 Ret.> LeftVolume;RightVolume	
setvolume	Name> setvolume Help> sets volume of a device Par.> device: ID of device to set volume: volume to set channel: channel to set ('left', 'right' or 'both') Def.> device: 0 volume: 1 channel: 'both'	calculate_while_playing.m mark_buttons.m modulate.m record.m

7.1.3 Recording Commands

Command	Description	Examples
record	Name> record Help> starts recording of a device Par.> device: ID of device to use for recording channels: number of channels to record bitlength: recording bitlength (16 or 32 only. Use 32 for 24bit-devices) sampfreq: recording sampling frequency threshold: threshold between 0 and 1. Recording is started after one sample has exceeded the threshold on the device. 0 or -1 disables threshold recording thrschannel: channel to look for exceeding threshold 'left': only checks left channel 'right': only checks right channel 'both': checks both channels length: lenght to record depending on threshold: -SPECIAL: value of -1: if setting the recording length to -1 not temporary data are saved to disk. Therefore 'getrecorddata' will fail after setting length to -1! Normal saving can be restored by setting length to 0 - enabled threshold: recording is stopped after exceeding the threshold and recording 'length' samples - disabled threshold: if set to 0 samples are recorded to harddisk until device is stopped by a command. Any other value will record to memory. At every time the last 'length' samples are stored in memory, older values are overwritten. Therefore 'getrecorddata' will return the last 'Length' recorded samples after stopping the device prepare: 0 or 1. If 1, the device is _not_ started but opened. Use prepare flag to prepare multiple devices for synchronization and start all prepared devices with 'startprepared' filename: records to file with filename 'filename'. If filename is set the recorded data will not be deleted on exit, while default file would be! NOTE: do not use filenames longer than approx 100 characters. Depending on the actual path creating the file may fail	record.m record_length.m record_thrs.m sync_record.m

	with a corresponding error message Def.> device: 0 channels: 2 bitlength: 16 sampfreq: 44100 threshold: -1 thrschannel: 'both' length: 0 prepare: 0 filename: tmp_DEVICEID.wav	
startprepared	Name> startprepared Help> starts all devices previous prepared using 'playfile', 'playmem', 'record', 'startio' or 'playringbuffer' Par.> type: 'waveout', 'wavein' or 'io' Def.> type: 'waveout'	sync_play.m sync_record.m
isrecording	Name> isrecording Help> determines if a device is recording Par.> device: ID of device to query Def.> device: 0 Ret.> 1 if device is recording, 0 else	record_thrs.m
recposition	Name> recposition\n" Help> determines recording position of a device\n" Par.> device: ID of device to query mode: when recording with threshold 'mode' determines if recposition is absolute number of samples since the device was started with 'record' or 'startprepared' or relative, i.e. the counter starts after the threshold is exceeded. Without threshold both modes are identical 'abs': count samples from the start 'rel': count samples from exceeding threshold Def.> device: 0 mode: 'rel'	
stoprec	Name> stoprec Help> stops recording of a device Par.> device: ID of device to stop Def.> device: 0	
stoprecall	Name> stoprecall Help> stops recording of all devices	record_length.m
saverecord	ame> saverecord Help> saves recorded data from a device Par.> device: ID of device that has recorded the data filename: file to to save to Def.> device: 0 filename: mandatory	
getrecorddata	Name> getrecorddata Help> retrieves recorded data of a device. Data depend on parameters used to start the device in command 'record' Par.> device: ID of device that has recorded the data Def.> device: 0 Ret.> one (mono) or two (stereo) row (!) vector	record.m sync_record.m

7.1.4 Input-Output-Commands

Command	Description	Examples
startio	Name> startio Help> starts recording and mapping recorded sound directly to output device Par.> wavein: ID of device to use for recording waveout: ID of device to use for playback channels: number of channels to record bitlength: recording bitlength (16 or 32 only. Use 32 for 24bit-devices) sampfreq: recording sampling frequency prepare: 0 or 1. If 1, the devices are _not_ started but opened. Use prepare flag to prepare multiple devices for synchronization and start all prepared devices with 'startprepared' length: see command 'record' Def.> wavein: 0 waveout: 0 channels: 2 bitlength: 16 sampfreq: 44100 prepare: 0	start_io.m
startprepared	Name> startprepared Help> starts all devices previous prepared using 'playfile', 'playmem', 'record', 'startio' or 'playringbuffer' Par.> type: 'waveout', 'wavein' or 'io' Def.> type: 'waveout'	sync_play.m sync_record.m

7.1.5 Device Commands

Command	Description	Examples
setbuffer	Name> setbuffer Help> sets buffer properties of device Par.> device: ID of device to set type: 'waveout' or 'wavein' numbuf: number of buffers bufsize: size of each buffer in bytes Def.> device: 0 type: 'waveout' numbuf: 20 bufsize: 4096 Ret.> numbuf;bufsize	device_info.m
getbuffer	Name> getbuffer Help> retrieves buffer properties of a device Par.> device: ID of device to query type: 'waveout' or 'wavein' Def.> device: 0 type: 'waveout' Ret.> numbuf;bufsize	device_info.m
getdeviceinfo	Name> getdeviceinfo Help> returns string info on all installed devices	device_info.m
getdevices	Name> getdevices Help> returns struct arrays with infos on waveout and wavein devices each containing the fields 'DeviceId' and 'DeviceName' Ret.> waveoutdevices_struct;waveindevices_struct	deviceinfo_struct.m

7.1.6 Ringbuffer Commands

Command	Description	Examples
setringbuffersize	Name> setringbuffersize Help> sets size of ringbuffer of device Par.> device: ID of device to set size: size in samples devided by channelnumber Def.> device: 0 size: 2048	ringbuffer.m synchronized_ringbu ffers.m
setringbuffermode	Name> setringbuffermode Help> sets mode of a devices ringbuffer Par.> device: ID of device to set mode: one of three modes (what to do if ringbuffer runs empty): 'error_on_empty': device is stopped with an error 'stop_on_empty': device is stopped 'zero_on_empty': zeros are played (playback of silence) Def.> device: 0 mode: mandatory, no default!	ringbuffer.m synchronized_ringbu ffers.m
playringbuffer	Name> playringbuffer Help> plays ringbuffer on a device Par.> device: ID of device to use for playback channels: number of channels for playback bitlength: bitlength for playback (16 or 32 only. Use 32 for 24bit-devices) sampfreq: sampling frequency for playback prepare: 0 or 1. If 1, the device is _not_ started but opened. Use prepare flag for prepare multiple devices for synchronization and start all prepared devices with 'startprepared' Def.> device: 0 channels: 2 bitlength: 16 sampfreq: 44100 prepare: 0	ringbuffer.m synchronized_ringbu ffers.m
putringbuffer	Name> putringbuffer Help> puts data into ringbuffer of a device Par.> device: ID of device to use for playback data: normalized vector with one (mono) or two (stereo) ROWS(!!) Def.> device: 0 data: mandatory Ret.> '1' if buffer was accepted, '0' else (buffer busy/completely filled)	ringbuffer.m synchronized_ringbu ffers.m

7.1.7 Special Commands

Command	Description	Examples
Tic	Name> tic Help> starts high resolution timer	tictoc.m
Toc	Name> toc Help> measures time using high resolution timer Ret.> milliseconds elapsed since last call of 'tic'	tictoc.m
Setbutton	Name> setbutton Help> enables button marking synchronized with playback. ATTENTION: after marking and unmarking a specific	mark_buttons.m

	button, the marking information is removed. To use the same marking information again you have to call 'setbutton' again!! Par.> device: ID of device to set handle: (window) handle of the button buttoncaption: caption of button startpos: starting point in samples length: marking length in samples mode: 'org': samples are calculated relative to start of playback 'mix': samples are calculated relative to start of mixing Def.> device: 0 handle: mandatory! startpos: 0 length: 5 mode: 'org'	
Debugsave	Name> debugsave Help> saves playback of a device to file passing an empty filename disables saving Par.> device: ID of device for en/disabling debugsave filename: filename of file to save to Def.> device: 0 filename: ''	modulate.m
sendplugin	Help> sendplugin Name> sends a command to a plugin Par.> device: device containing the plugin chain type: 'waveout' or 'wavein' command: command to be sent. Plugin or PluginChain itself must recognize command. PluginChain itself only knows commands 'show' and 'hide'. If passing 'show' you can pass additonal parameters in the string separated by blanks: first parameter: left position of window in pixels second parameter: top position of window in pixels the complete rest: caption of window ATTENTION: the command 'data:x' is special, where x can be any MATLAB® vector. The vector is translated to a double pointer to the first value (as long int),the total number of double values in the vector and the name of the vector and passed to the plugin, colon delimited, e.g.: 'data:123456:1024:x'\n"\ you may use these values in user defined plugins index: index of plugin within chain (first plugin has ID 0, PluginCain itself has ID -1) Def.> device: 0 type: 'waveout' command: mandatory index: 0 Ret.> string based return value from plugin depending on command...	pluginеq.m pluginceq.m pluginwrite.m

8 SoundMex and MATLAB® GUIs

Many users take advantage of the powerful MATLAB® GUI when developing applications with MATLAB®.

However, when using SoundMex (or any other MEX based binary modules), a problem arises when calling SoundMex directly from a GUI (e.g. using the ‘callback’ member of an UI control): it may occur, that the UI controls sends a command to SoundMex, while the main script executes a

SoundMex command at the same time, i.e. one SoundMex command is called asynchronously, while the last SoundMex command has not returned yet. In this case you will get an error message:

```
Warning: Error in command ???: sounddll2 busy: asynchroneous command call
failed!
```

As a consequence serious problems in the thread synchronisation may occur and MATLAB® may crash. Unfortunately this cannot be solved in SoundMex itself due to the special handling of processing time by MATLAB® (a call from a script seems to suspend a GUI callbacks that have not returned).

Therefore you should avoid direct calls SoundMex from MATLAB® GUIs, whenever there is a chance that your script may call SoundMex at the same time as well. You can achieve this by replacing calls to SoundMex in GUI callbacks by setting of global variables, that are polled in the main script in order to call the desired SoundMex command.

Please take a look at the example script ‘guicall.m’ in the examples directory: the example implements a ‘correct’ and an ‘invalid’ call from a GUI to SoundMex and may be taken as ‘guideline’ when using SoundMex with MATLAB® GUI.

9 The SoundMex realtime DSP-Plugin-Pipe

SoundMex contains a built-in realtime DSP-Plugin-Pipe. It is available only with the DSP and DSP-SDK licenses.

General descriptions

1. **Plugins:**
   SoundMex DSP-Plugins are binary modules (DLL's) that can manipulate sound data sent to or received from a wave device on the fly. Several plugins can be *plugged* together using a plugin manager, a so called *plugin chain.* Various (and a growing number of) generic plugins is shipped with SoundMex, user defined plugins can be developed using the SoundMex-DSP-SDK (Software Development Kit, see below).

2. **Plugin chains:**
   Every device (recording or playback device) can define a so called *plugin chain*. A *plugin chain* is a container for a variable number of SoundMex DSP-Plugins. The number of plugins is limited only by the speed of your computer, and how 'expensive' calculations are within the single plugins. The generic plugins are highly optimized and do not consume much processor power (except for the 'Visualisation' plugin). The plugins are *plugged* together by the chain in a manner, that the output data of the first plugin are the input data for the second plugin and so on.

3. **Activating a plugin chain:**
   A plugin chain for one or more devices is activated by adding a line to the SoundMex INI-File (either the default file, or the user defined file, see command 'init'):

   ```
   [WaveOutPlugins]
   0=plugins/plugin0.ini
   1=plugins/plugin1.ini
   …

   [WaveInPlugins]
   0=plugins/plugin7.ini
   1=plugins/plugin8.ini
   …
   ```

The sections *WaveOutPlugins* and *WaveInPlugins* respectively denote if the chain is to be plugged into the signal path of a playback or recording device. The name of the value is the corresponding DeviceID of the recording or playback device. The value (right hand of the '=') specifies another INI-File describing a single plugin chain (see below)

4. **Configuring a plugin chain:**
   Every plugin chains is configured with a INI-File. Such a chain INI-File contains an optional section *GlobalData* and one section for each plugin to be used.

```
[GlobalData]
MyData=100
MyData2=200

[0]
FileName=plugins/visualize.dll
LogView=1

[1]
FileName=plugins/olaeq.dll
EQFile=plugins/conv_1024.cnv
OLAFFTLength=1024
OLAWindowLength=800

[2]
FileName=plugins/visualize.dll
LogView=1
```

   The *GlobalData* section contains fields that are available to all plugins during initialization (i.e. all plugins can 'read' these values).
   All other sections describe one plugin each. The names of the section must be ascending integer values (not subsequent, but ascending). The only required field is *FileName*, which holds the name of the DLL that will be loaded by the plugin chain. All other parameters are plugin-specific. Every plugin can read these values during initialization. Plugins that require further parameters will fail to initialize if a value is missing.

   So the above example describes a plugin chain containing three plugins (for a description of those generic plugins please refer to the *SoundMex DSP-Plugins* documentation):
   - at the first position the data are visualized (spectrum, spectrogram...)
   - the second plugin is an overlapped add equalizer (spectral manipulation)
   - the third is again a visualization (to see changes done by the equalizer)

   The data will 'flow' from top to bottom through these plugins.

5. **Communication with plugins:**
   After initializing SoundMex you can use the *sendplugin* command to communicate with a plugin. String commands can be sent to a single plugin, but the plugin itself must 'recognize' the command. In the example above, you can activate and deactivate the equalizer on the fly using the command

```
soundmex2('sendplugin', 'command', 'active=1', 'index',1); %activation
soundmex2('sendplugin', 'command', 'active=0', 'index',1); %deactivation
```

   assuming the plugin chain defined above was activated for wave out device No. 0 (i.e. the file plugin0.ini from 'Activating a plugin chain' is used). For a list of recognized commands of the generic plugins please refer to the *SoundMex DSP-Plugins* documentation.

   Important note: the indices of the plugins are not necessarily equal to the section names in your inifile: they start with 0 for the first plugin are strictly ascending, i.e. incremented by 1 for each plugin in the chain.

You can send a special command string to a plugin to get access to a MATLAB® vector from a plugin: the command ‘data:x’ (x may be any MATLAB® vector) will be translated to a string containing a pointer to the first double value of vector x as long int, the total number of double values contained in x and the name of the vector, e.g. the command

```
x = zeros(2, 44100);
soundmex2('sendplugin', 'command', 'data:x’);
```

will result in a string command passed to plugin with index 0 for waveout device 0

```
'data:12345678:88200:x’
```

(assuming that ‘12345678’ is the pointer to the first value). A plugin may use these values, please refer to the documentation of the particular plugin. **ATTENTION**: a plugin may access the data of the MATLAB® directly. So depending on the plugin, any change of the corresponding vector within MATLAB® may cause access violations within the plugin!

6. **Communication with the plugin chain and updating global data:**
The plugin chain itself can receive commands using -1 as plugin index in the *sendplugin* command. The following commands are supported:

Command	Description
show	shows the specified plugin chain. The command takes three optional parameters, separated by spaces: first is the window position in pixels from the left, second is the window position in pixels from the top, and complete rest will be set as window caption, for example: `soundmex2('sendplugin', 'command', 'show 10 10 Hello', 'index', -1);` **Attention:** If you want to use the second or third parameter, all prior arguments must be given as well! Calling show without parameters will show the plugin chain either at the top left corner of the screen, or - if the SoundMex main window is visible - to the right of the main window.
hide	hides the plugin chain `soundmex2('sendplugin', 'command', 'hide', 'index', -1);`

notify	with notify you can 'update' or 'add' global data. All plugins (except external plugins, see SDK) will re-read settings that they have read on initialization (if possible). You can specify an arbitrary number of arguments in the form 'name=value' separated by space, e.g.
	
	`soundmex2( 'sendplugin',                                   ...`
	`           'command', 'notify MyData=50 MyData2=100',   ...`
	`           'index', -1);`
	
	Attention: the plugin itself must support notify actions. Please test your application seriously!

You can show/hide a plugin chain by clicking the small button to the right of the levelmeter on the main SoundMex window after calling `soundmex2('show')`, too.

For an example see 'plugin.m' in the examples directory. For a description of the generic SoundMex DSP-Plugins and their parameters please refer to the *SoundMex DSP-Plugins* documentation.

10 The SoundMex DSP-SDK

The SoundMex DSP-SDK is a software development for building user defined realtime plugins for the SoundMex DSP-Plugin-Pipe. It provides two different types of user defined SoundMex DSP-Plugins:

- *plugins for manipulations in the time domain*: the plugin gets block by block access to the wave data as normalized floating point values and can manipulate each sample directly before it is sent to the device (or a subsequent plugin).
- *plugins for manipulations in the frequency domain*: the plugin gets access to the complex FFT-spectrum of signal blocks before they are reconstructed and sent to the device (or a subsequent plugin). Data segmentation before the FFT and data reconstruction after the IFFT is done by overlapped add procedure. FFT-length as well as window length (zero padding if window length < FFT-length) and window feed (half, quarter or eighth) for the overlapped add can be set by the user.

Both plugin types can be realized in two different ways:

- script based MATLAB® plugins: a user defined MATLAB® command (script or mex) is called on runtime within the DSP-Pipe. Wave data are passed block by block to the command. Data can easily be manipulated in the script and passed back to the DSP-Pipe as output argument. This kind of plugin is extremely easy to use, but is limited in the performance.

- ‘binary’ plugins: high performance plugins can be realized in any programming language that can build Windows Dynamic Link Libraries (DLL). These libraries can be written in any programming language that can build Dynamic Link Libraries (DLL). Implementation can be easily done without any knowledge of windows multimedia, sound devices or other 'special' programming skills. Implementing one function with only some ANSI-C floating point calculations (the manipulation of the signal) is sufficient for many applications. A simple API reference and some examples are shipped with the SDK.

For a complete documentation please refer to the *SoundMex DSP-SDK* documentation.

11 Feature Matrix

SoundMex is shipped with different licenses with different registration fees. The feature matrix below gives an overview on the different licenses. The SDK licenses are identical to ProfessionalDSP licenses with the additional feature, that they can load external (user defined) plugins. (x) indicates that command is not supported in with all parameters, e.g. using 'setbuffer' command with parameter 'type', 'wavein' obviously requires a recording license.

	Free	Basic	Professional	ProfessionalDSP	BasicRec	ProfessionalRec	ProfessionalDSPRec
debugsave		x	x	x	x	x	x
exit	x	x	x	x	x	x	x
filetofile		x	x	x	x	x	x
getbuffer	x	(x)	(x)	(x)	x	x	x
getdeviceinfo	x	x	x	x	x	x	x
getlasterror		x	x	x	x	x	x
getrecorddata					x	x	x
getvolume		x	x	x	x	x	x
hasclipped	x	x	x	x	x	x	x
help	x	x	x	x	x	x	x
hide		x	x	x	x	x	x
init	x	x	x	x	x	x	x
isinitialized	x	x	x	x	x	x	x
ismixing	x	x	x	x	x	x	x
isplaying	x	x	x	x	x	x	x
isrecording					x	x	x
memtofile		x	x	x	x	x	x
mixfile	x	x	x	x	x	x	x
mixmem	x	x	x	x	x	x	x
mixposition	x	x	x	x	x	x	x
playfile	x	x	x	x	x	x	x
playmem	x	x	x	x	x	x	x
playposition	x	x	x	x	x	x	x
playringbuffer			x	x		x	x

putringbuffer			x	x		x	x
record					x	x	x
sendplugin				(x)			x
saverecord					x	x	x
setbutton		x	x	x	x	x	x
setbuffer	x	(x)	(x)	(x)	x	x	x
setplaymode			x	x		x	x
setringbuffermode			x	x		x	x
setringbuffersize			x	x		x	x
setvolume		x	x	x	x	x	x
show		x	x	x	x	x	x
showerror		x	x	x	x	x	x
startio					x	x	x
startprepared			(x)	(x)		x	x
stopall	x	x	x	x	x	x	x
stopmix	x	x	x	x	x	x	x
stopplay	x	x	x	x	x	x	x
stopplayall	x	x	x	x	x	x	x
stoprec					x	x	x
stoprecall					x	x	x
tic		x	x	x	x	x	x
toc		x	x	x	x	x	x
version	x	x	x	x	x	x	x

12 Common problems

12.1 FAQ

Problem	Solution
playback or record has dropouts	increase the buffersize and number of buffers using the 'setplaybuffer' and 'setrecordbuffer' commands
MATLAB® GUIs seem to 'hang' while SoundMex commands are executed	Window messages are not processed during SoundMex calls to avoid recurrent calls from MATLAB® GUIs. This leads to hanging GUIs in while loops without a small 'pause' (e.g. pause(0.001)) or 'drawnow' anywhere in loops or in play commands in 'sync' mode.
dropouts or deadlocks when using loops with SoundMex commands in it	put a small 'pause' (e.g. pause(0.001)) or 'drawnow' anywhere in the loop , otherwise the windows message loop is not processed accurately
system hanging and/or MATLAB® crashes when recording	be sure that you do not run SoundMex from a network drive and that the temporary path of SoundMex does not point to a network drive (see Installation above)
when calling 'getrecorddata' errors like File reading error, retrying... occur.	be sure that you do not run SoundMex from a network drive and that the temporary path of SoundMex does not point to a network drive (see Installation above)
when playing or recording 32bit waves (or setting recordformat to 32bit) you get an error like `'specified device cannot play requested format'` although your device really should be able to play it	enter the control panel and select 'No Sounds' in your 'Sounds and Multimedia' settings: otherwise some Windows message box may play 'ding' in 16 bit mode and your sound card is blocked for higher resolutions at that time!
If a 'while' loop calling SoundMex is running and you cal SoundMex from an asynchroneous callback (e.g. a keypress function) simultaneously you get warnings like `Warning: Error in command ???: sounddll2 busy: asynchroneous command call failed!`	Do not call SoundMex from UI. In most cases you can avoid this as in the following example: - a while loop calls 'isplaying' - a button should call 'playposition' simultaneously to retrieve the sample position at the moment when the key is pressed - Solution: call 'playposition in the while loop too and store the result in a global variable. In the keypress function only read this global variable

	See also chapter ‘SoundMex and MATLAB® GUIs’ in the manual.
when recording with synchronized devices, the recorded data do not have same length	Unfortunately the stopping of devices cannot be synchronized. The devices are stopped subsequently and so different buffers may have recorded a different number of buffers when they are ‘really’ stopped’. However, if the devices support synchronization the start is synchroneous and you can cut additional buffers at the end of the recorded data after retrieving them using ‘getrecorddata’.
recorded buffers are ‘too short’ or, when recording extremely short sequences, empty	Recording is done buffer by buffer. When stopping the device, buffers that are not filled completely, may be lost (depends on sound card and corresponding driver). Try to record longer sequences and cut the data afterwards or have a look at the ‘length’ parameter of the ‘record’ command.
start recording fails with an error like ‘error creating file’	This may have three reasons: - be sure that the name in the ‘filename’ option contains a valid filename (e.g. no asterisk in it) - do not use filenames longer than approx. 100 characters in the ‘filename’ option. Depending an the actual path, he creation of the recording file may fail otherwise - be sure that no other application accesses the file. NOTE: if the file is highlighted in the Windows® Explorer, the operating system ‘accesses’ the file, and recording will fail!

12.2 Synchronizing devices

One feature of SoundMex (Professional license and above) is the synchronization of multiple playback and/or recording devices, i.e. to start the devices exactly at the same time (with sample accuracy). To understand how (and why) this feature works with some sound cards, here a short explanation how this feature works.

SoundMex makes use of the Microsoft Windows® multimedia API MMSYSTEM (aka MME). If you are using the ‘prepare’ option, SoundMex opens the corresponding device in a special way, telling him to wait for all other ‘prepared’ devices once it receives a start call. In this way **all** prepared devices will wait until **all** of them have received the ‘start’ command. This has to be supported by the driver of the particular sound card, otherwise SoundMex has no chance to synchronize multiple devices!

To check, whether a particular sound card supports synchronization, you may use some SoundMex commands like this (just the ‘interesting’ code, no ‘init’, ‘exit’ … here):

```
% starting first device with prepare option
soundmex2('playfile',
          'prepare', 1,
          'filename', 'mywave.wav’,
          'device', DeviceToTest1);
```

```
% starting second device without prepare option
soundmex2('playfile',
          'prepare', 0,
          'filename', 'mywave.wav',
          'device', DeviceToTest2);

% wait a second
pause(1);

% start prepared device(s)
soundmex2('startprepared',
          'type', 'waveout');
```

If the devices do **not** support any synchroization (i.e. waiting for each other), then the second device will start its playback immediately, the first will start one second later, after receiving the 'startprepared' command. Otherwise both devices will start after the pause. Note: this does only show that some kind of waiting (synchronization) is implemented! It does not guarantee for zero samples delay! You may implement a similar test for recording devices.

If your sound card supports synchronization across different device types (synchronizing playback with recording devices), you have to take care that you start both types of prepared devices in your scripts. Otherwise the prepared device will wait forever! The following example commands (in this order)

```
soundmex2('playfile',
          'prepare', 1,
          'filename', 'mywave.wav');
soundmex2('record',
          'prepare', 1);
soundmex2('startprepared',
          'type', 'waveout');
```

will **not** start the playback of the file 'mywave.wav' because the playback device will wait until the recording device is started as well (you have to use 'all' or 'io' as type). Note: using this type of script may be useful to determine if your particular sound card does synchronization across device types: if the playback starts, it does **not** synchronize different device types.

You may get a delay between the 'start' of recording and playback devices even if your sound card supports cross device type synchronization, caused either by the anti-aliasing filters of the A/D-converters or any other additional external hardware. A typical delay may be in the order of tens to hundreds of samples, but this offset is fixed, i.e. it will be the same for every synchronized start. So, once this offset is determined (e.g. by playing a pulse and recording it with a shortcut between playback and recording device) it can be taken into account in further experiments. Note: this delay will vanish, if you do a shortcut of the devices without A/D and D/A conversion (e.g. direct optical ADAT shortcut). Take a look at the example script `sync_check.m` that plays and records a pulse at a well defined sample position.

13 Upcoming Features

A new version ‘SoundMexPro’ supporting ASIO is under development and will be released mid of 2008.

14 Version History

Version 2.33.8, Date: 15.04.08

- comlexeq plugin replaced by cplxeq plugin: no performing a full complex multiplication (see plugins manual)

Version 2.31.8, Date: 13.01.08

- bugfix plugin manager (error message ‘Plugin incompatibility: Plugin Master version (2.1) different than Plugin version (2.2)’)

Version 2.30.8, Date: 09.11.07

- bugfix in command ‘setvolume’.
- bugfix in plugins “Overlapped Add Equalizer Plugin” and “Complex Overlapped Add Equalizer”: now changing of filter coefficient files allowed, if filter is disabled.

Version 2.29.8, Date: 20.10.07

- bugfix for ‘setbutton’ command

Version 2.28.8, Date: 21.05.07

- New command ‘getdevices’ returning device information in struct arrays

Version 2.28.7, Date: 03.05.07

- Binary ‘soundmex2.mexw32’ recompiled for MATLAB 7.1 support.

Version 2.27.6, Date: 16.04.07

- New binary ‘soundmex2.mexw32’ added for MATLAB 2007a support

Version 2.27.5, Date: 28.11.06

- recposition now returns number of recorded samples after stoprec as well.
- bugfix for recording: temporary file is no removed after ‘exit’.

Version 2.26.5, Date: 12.07.06

- Command ‘startio’ takes additional parameter ‘length’

Version 2.25.5, Date: 27.06.06

- Bugfix in setup
- Internal minor bugfixes

Version 2.24.4, Date: 17.05.06

- Command interface changed back (as it was before 2.23.4), because script based plugins did not work with new interface. Please read the new chapter ‘SoundMex and MATLAB® GUIs’

Version 2.23.4, Date: 25.04.06

- New parameter ‘channel’ added to command ‘setvolume’
- Command ‘getvolume’ returns two volumes (for each channel)
- Command interface changed for ‘real’ synchroneus command execution within SoundMex. **NOTE:** while a SoundMex command is running, no window messages are processed any more to avoid recurrent calls from MATLAB® GUIs. This may look like a ‘deadlock’ when playing files (or vector) with flag ‘mode’ ‘sync’!!

Version 2.22.4, Date: 27.03.06

- New feature for WriteToBuffer plugin: start writing with offset possible
- New option 'filename' for record command

Version 2.21.4, Date: 05.01.06

- Bugfix in tic and toc commands

Version 2.20.4, Date: 19.12.05

- More bugfixing in command 'setbutton'

Version 2.19.3, Date: 15.12.05

- Bugfix in command 'setbutton' for MATLAB® 5.3 and MATLAB® 6.x: now running with disabled buttons too
- Bugfix in command 'setbutton': resetting positions correct after stopping device in mix mode

Version 2.18.2, Date: 05.12.05

- Subversion introduced
- Bugfix for WriteToBuffer plugin

Version 2.18, Date: 29.11.05

- SoundMex now running on MATLAB® 5.3 (R11.1) and above!
- Bugfix in WriteToBuffer plugin

Version 2.17, Date: 15.11.05

- Bugfix visualize plugin.
- DSP-SDK now included in package and can be tested in demo mode

Version 2.16, Date: 04.10.05

- Bugfix in sendplugin command: now carriage return and line feed are cutted off the returned strings
- Manuals maintenance

Version 2.15, Date: 30.08.05

- Bugfix for fade parameter for commands playfile and playmem
- New parameter 'startpos' for command playfile
- New parameter 'mute' for commands playfile and playmem
-

Version 2.14, Date: 30.08.05

- Bugfix in sendplugin command: internal bugfix for mha usage

Version 2.13, Date: 29.07.05

- Bugfix in plugin chain: performance problem by notifying plugins in all chains fixed

Version 2.12, Date: 14.07.05

- New command 'recposition' added.

- New parameter ‘thrschannel’ added to command ‘record’: exceeding threshold now can be restricted to left, right or both channels.
- New feature: temporary saving of record data can be disabled by setting record length parameter to -1 (may be useful when working with plugins).
- New plugin ‘WriteToBuffer’ added.
- New plugin ‘ComplexEq’ (complex equalizer) added.

Version 2.11, Date: 07.07.05

- Version control for plugins added (1.x plugins are incompatible with v 2.x plugins!). This is checked now
- Wrong error message when returning a ‘data’ parameter to a command that does not exist at all fixed

Version 2.10, Date: 08.06.05

- Initialization timeout increased to 10 seconds for slow plugins
- Filter reading for equalizer plugin rewritten for higher efficiency

Version 2.09, Date: 26.05.05

- Problem with dongle driver on some XP machines fixed

Version 2.08, Date: 02.05.05

- playposition rewritten to get sample accurate position
- Bugfixes: return values are filled with zeroes if command fails

Version 2.07, Date: 08.04.05

- Minor bugfixes

Version 2.06, Date: 11.03.05

- playmem, playfile and stopplay have new parameter ‘fade’ for fade in and fade out of playback
- visualization plugin completely rewritten: sizeable… (see SoundMex-Plugin documentation)

Version 2.04

- setbutton command parameters changed: for MATLAB® 7 support the settbutton function had to be redesigned.

Version 2.01 to 2.03:

- Bugfixes

Version 2.0, Date: 13.12.04

- first release, completely new command interface