ztracker manual

documentation for *ztracker - a tracker-style MIDI sequencer.*

by Daniel Kahlin

Version 0.1

31st October 2001

This documentation is part of ztracker - a tracker-style MIDI sequencer.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

ztracker is a win32 MIDI only tracker with an interface that is almost a 1:1 clone of the popular Impulse Tracker DOS tracking software.

It supports multiple MIDI in and out devices, 64 MIDI tracks (expandable to 256), it can sync to an external sequencer via MIDI clock, `.mid` export, parameter drawing, 96ppqn resolution, and much more.

The original *ztracker* code (upto 0.82) was written completely by Christopher Micali, except for the Impulse Tracker loader code, which was written by Austin Luminais.

Version 0.90 of *ztracker* was the first build to use SDL, the GPLed graphics and input library. Before 0.90 *ztracker* used libCON (`http://www.photoneffect.com/`) *ztracker* would not have been possible without libCON.

1.1 Goals

- a MIDI tracker running under atleast win32
- fast, simple interface
- support for multiple out ports
- well documented

1.2 Features

- near 1:1 copy of Impulse Tracker interface
- 64 track sequencer with variable 32-256 rows/pattern, 256 total patterns
- easy use of multiple machines across multiple MIDI devices/interfaces
- rock solid timing that tested as good as cubase (3/496ppqn error)

- load/save compressed *ztracker* (`.zt`) song files
- volume/effect curve drawing in pattern editor
- Impulse Tracker song file importing
- MIDI file (`.mid`) export
- auto sync via MIDI-clock
- intelligent midi-in with slave to external sync

1.3 What *ztracker* is Not

ztracker is *not* ...

- ... A sample player. If you want samples, use a sampling synthesizer or a virtual sampler w/ an ASIO card. If we wanted sampling ability, We'd be using buzz. Buy something nice, like an akai or a yamaha.
- ... A GM composing system. While you can make GM tunes with zt, that is not what zt is written for. GM, GM2, and XG specific features will not be implemented.

Chapter 2

Getting Started

2.1 Getting *ztracker*

The latest *ztracker* release can be found at the *ztracker* web page. (http://ztracker.sourceforge.n
Releases are named `zt-x.y.zip`, where `x` is the version, and `y` is the revision. For each release there is also a source code release, which is named `zt-x.y-src.zip`. The source code release is only necessary for doing development on *ztracker*.

2.2 Installing

Just unzip the *ztracker* `.zip` archive to wherever you want it.

2.3 Quick Start

- Run `zt.exe`.
- A splash screen will appear. Click to get past it.
- You are now in the pattern editor. `CTRL-Tab` switches the view mode. Press `F1` for quick help, `F12` for the global configuration, `F3` for the instrument editor, `F5` for play, `F8` for stop, and `F2` to get back to the pattern editor again.

2.4 Configuring

Upon installation *ztracker* is configured for 640x480 resolution running in a window. This should be adequate for most people. If you want change the resolution or window mode, exit *ztracker* and run the included `ztconf.exe` utility. This will let you choose between a number of fixed resolutions, and toggle fullscreen mode.

2.5 User Interface

Fill in the blank!

Chapter 3

Basic Usage

3.1 The Pattern Editor

Pressing `F2` bring up the pattern editor. Let's begin by a showing a row in the pattern editor set to *View mode: Big...*

```
E-2 00 7F 024 W2000
```

`E-2` is the note itself. In this case an E played in octave 2.

`00` is the instrument number.

`7F` is the velocity of this note.

`024` is the length of the note in subticks.

`W2000` is a *player command*. This particular command sets the pitch bend to the center position.

The same row looks like this in *View mode: Regular...*

```
E-2 00 7F 024
```

...like this in *View mode: FX...*

```
E-2 7F W2000
```

...and like this in *View mode: Squish*

```
E-2 7F
```

3.2 Player Commands

Command `A..xx`

The player command `A..xx` will set the ticks per beat value to `xx`, where `xx` is the tpb value in hexadecimal numbers. The only allowed tpb values are 2 (`02`), 4 (`04`), 6 (`06`), 8 (`08`), 12 (`0C`), 24 (`18`) and 48 (`30`).

Command `C..xx`

When the player command `C..xx` is encountered, the next tick the player will skip to the next pattern in queue from row `xx`, where `xx` is the row in hexadecimal numbers.

Command `D..xx`

The player command `D..xx` delays the current note, volume or note cut with `xx` subticks, where `xx` is the number of subticks in hexadecimal numbers.

Command `Exxxx`

The player command `Exxxx` makes a pitch slide down. The pitch bend value is decremented by `xxxx` every subtick. `E0000` repeats the parameter from the pitch slide on the previous row.

Command `Fxxxx`

The player command `Fxxxx` makes a pitch slide up. The pitch bend value is incremented by `xxxx` every subtick. `F0000` repeats the parameter from the pitch slide on the previous row.

Command `P....`

The player command `P....` will send a MIDI program change message corresponding to the settings in the current instrument.

Command `Q..xx`

The player command `Q..xx` will retrigger the note once every `xx` subticks.

Command `R..xx`

The player command `R..xx` will start the arpeggio `xx`. If `xx` is `00` the last started arpeggio will be continued.

Command `Sxxyy`

The player command `Sxxyy` sets the MIDI continuous controller `xx` to the value `yy`, where `xx` and `yy` are hexadecimal numbers. If `xx` or `yy` are `80` their respective last used values will be used again.

Command `T..xx`

The player command `T..xx` will set the tempo to `xx`, where `xx` is the tempo in hexadecimal numbers. The minimum allowed tempo is 60 bpm (`3C`), and the maximum tempo is 240 bpm (`F0`).

Command `Wxxxx`

The player command `Wxxxx` sets the pitch bend to `xxxx`, where `xxxx` is the desired value in hexadecimal numbers. `W0000` is maximum down, `W3FFF` is maximum up, and `W2000` is the center position. `W2000` is typically used to reset the pitch bend after a pitch slide command.

Command `X..xx`

The player command `X..xx` will set panning to `xx`, where `xx` is the panning in hexadecimal numbers. `00` is far left, `7F` is far right, and `40` is center. (this command does the same as `S0Axx`)

Command `Zxxyy`

The player command `Zxxyy` will send the midimacro `xx`, with the optional parameter `yy`. If `xx` is `00` the last used midimacro will be used again. If `yy` is `00` the last used value will be used again.

3.3 Keyboard Shortcuts

These are the key commands. Note that name of the keys is from an american keyboard layout, but it is their position that counts. I.e some key names are different if you have a keyboard layout of another country.

A..xx	Set TPB (ticks per beat)
Cxxxx	Pattern break - setting the C marker denotes the end of a pattern. ex: C0000 will skip to the next pattern, row 0
D..xx	Delay note/volume/cut - by xxxx sub-ticks.
Exxxx	Pitch slide down, E0000 will repeat the last portamento command.
Fxxxx	Pitch slide up, F0000 will repeat the last portamento down command.
P....	Send program change message for the current instrument.
Q..xx	Retrig note every xx subticks
R..xx	Start arpeggio xx. R0000 will continue the last arpeggio
Sxxyy	xx is the CC (Continuous Controller) number, yy is the value (00-80). ex. S0142 will set CC 01 (Mod Wheel) to a value of 42. >=80 sends last value
T..xx	Set tempo to xx bpm
Wxxxx	Absolute pitch set, between 0000 and 3FFF, 2000 is no pitch change, 0000 is max down, 3FFF is max up
X..xx	Set panning, 00=left, 40=center, 7F=right
Zxxyy	Send midimacro xx with parameter yy. 00 repeats the last used value

Table 3.1: Player Commands

3.3.1 Pattern Editing Keys

Space	Turns Edit (or "Keyjazz") mode on or off
Home/End	Just try them
PgUp/PgDn	Up/down 16 rows
Ins/Del	Insert/delete row on current track Hold CTRL for ALL tracks
Tab	Next track
SHIFT-Tab	Previous track
CTRL-Tab	Change pattern edit viewmode
ALT-1 => 0	Mute/unmute track 1-10
ALT-F9	Mute current track
ALT-F10	Solo current track
keypad +	Moves ahead one pattern.
keypad -	Moves back one pattern.
SHIFT-keypad +	Moves ahead one order
SHIFT-keypad -	Moves back one order
keypad /	Octave down.
keypad *	Octave up.
SHIFT-,	Select previous instrument.
SHIFT-.	Select next instrument.
SHIFT-	Switch between regular and effect-draw mode Use Tab here to switch edit mode
CTRL-1 => 0	Change the step-value in the pattern editor
ALT-‘	Set previous note's length to distance between note and current cursor position
ScrollLock	Pattern-follow mode (cursor follows playback)

3.3.2 Block Functions Keys

CTRL-B	Marks the beginning of a select block.
CTRL-E	Marks the end of a select block.
CTRL-L	Select whole track, pressing it again will select the entire pattern.
CTRL-U	Unselect block
CTRL-C	Copy the selected block to the clipboard.
CTRL-P	Copy pattern, paste to next empty pattern, and jump to new pattern
CTRL-V	Paste the contents of the clipboard to the cursor location.
CTRL-O	Overwrite paste
CTRL-M	Merge paste the contents of the clipboard with the contents of the location you're pasting to.
CTRL-X	Cut the selected block. (remove and copy to clipboard)
CTRL-Z	Clear block (hold this one)
CTRL-N	Set length of first row of block to length of block
CTRL-W	Clear unused volumes
CTRL-I	Interpolate effect data thru block
CTRL-T	Set all effect and effect data fields of block to same value as first row in block
CTRL-K	Interpolate volume (block)
ALT-Q	Transpose block up
ALT-A	Transpose block down
CTRL-J	Popup scale volume window (10-200 %)
CTRL-‘	Set all note's lengths in selection to distance between note and next note
ALT-S	Set instruments in block to current instrument
SHIFT-move	Block selection (movements are arrows and PgUp/PgDn)

3.3.3 Instrument Editor Keys

ALT-1 => 1	Quick selects midi device
CTRL-1 => 9	Quick selects midi channel
ALT-T	Toggle tracker mode on/off
Mouse-2	Focus a slider

3.3.4 Order Editor Keys

SHIFT-keypad +	Moves current order ahead
SHIFT-keypad -	Moves current order back

3.3.5 Global Keys

F1	Quick Help
F2	Pattern editor - Pressing F2 again while in the pattern editing screen brings up the pattern settings.
F3	Instrument editor
F4	Arpeggio editor
CTRL-F4	Midimacro editor
F10	Song message editor
F5	Song play (F5 PatternDisplay widget, Left-Right/Space/ALT-1 thru ALT-0/ALT-F9/ALT-F10)
F6	Pattern Play - will play the current pattern being edited.
F7	Start the song at the current row - If in the pattern edit mode, F7 will start at the current row, at the first instance of the pattern in the pattern order. If in the pattern order screen, F7 will start the song at the cursor position (by pattern).
SHIFT-F11	Set current order
SHIFT-F7	Play song from the current order
F8	Stop playback
F9	Panic (all midi-off/SHIFT-F9 performs a hard driver panic)
CTRL-F9	Load Song...
CTRL-F10	Save Song As...
F11	Song Configuration and Order editor
F12	System Configuration (midi dev select)
ALT-F12	About
CTRL-S	Save Song
CTRL-ALT-N	New Song
ALT-P	Song Duration
CTRL-ALT-Q	Quit

3.3.6 User Interface Keys

Up/Down	Cycles through widgets
Tab	Cycles forwards through widgets
SHIFT-Tab	Cycles backwards through widgets
Enter	Confirms choice
Space	Toggles an option or clicks a button
Sliders	If you hold CTRL while using Left/Right the slider will move in bigger steps
Listboxes	Up/Down scroll, Space toggles select-items in a list box
Popups	Esc closes, all other UI keys apply

3.4 Pages

Fill in the blank!

3.5 Editing Song

Fill in the blank!

3.6 Editing Tracks

Fill in the blank!

3.7 Loading

Pressing `CTRL-F9` switches to the load page. Here you may select a file for loading. Press `Enter`, or click twice on the file you wish to load. If you have edited the song currently in memory, you will be asked if you wish to lose those edits. *ztracker* loads `.zt` files. Further if the file you select has the `.it` extension, *ztracker* will try to import it as an Impulse Tracker song. (see section 4.3 on page 13)

3.8 Saving

Pressing `CTRL-F10` switches to the save page. Here you may select a directory and file name for your song file. Find the correct directory by using the cursor keys and `Tab` to cycle through the windows (or use the mouse). Then enter the desired file name into the text box below the file name selector, and press `Enter` to save. If the file already exists, you will be asked if you wish to overwrite the previous file. You can also select an already existing file by selecting it as in the load page.
There are two boxes just below the file name text box, selecting if a `.zt` file or a `.mid` file shall be created. This must always be `.zt` for *ztracker* to be able to load the resulting file. Selecting `.mid` makes *ztracker* export a MIDI file. (see section 4.2 on page 13)

Chapter 4

Advanced Usage

4.1 Synchronizing

4.1.1 Receiving Sync from a Hardware Sequencer

ztracker can receive MIDI clock synchronization from a specified MIDI port. It will start and stop when the master sequencer is started/stopped. It will also follow MIDI song position pointer messages. Note however that the latter does not work correctly if the song contains tempo changes or tpb changes.

4.1.2 Receiving Sync from a Software Sequencer

This works in the same manner as with the hardware sequencer, but you need to have a software MIDI port which you can transfer the MIDI clocks through. This is easily done by setting up Hubi's Loopback Device or MIDI Yoke.

4.1.3 Sending Sync to External Equipment

ztracker can send MIDI start/stop, and MIDI clock synchronization to a specified MIDI port. MIDI clocks occur 24 times per quarter note. (24 ppqn). These pulses can be used by external equipment to synchronize to the tempo of *ztracker*.

4.2 Exporting MIDI Files

Fill in the blank!

4.3 Importing Impulse Tracker Songs

Fill in the blank!

Appendix A

Details

A.1 Processing Order

Tracks are processed in ascending order. Any commands/data in track 1 is processed and sent before commands/data in track 2, and so on... Each track has its own parameter memory. For example, if you play the same midi channel from both track 1 and 2 and do a pitch slide in track 1, when you try continue the pitch slide from track 2, it will start from the same point that track 1 started from.

A.2 Pitch Bend Values

Pitch effects are ways of setting the MIDI Pitch bend value. Normally a synth has a pitch bend *range* defined for each sound. This *range* determines how many semitones the pitch is shifted for the maximum pitch bend *value*, up or down. Mosts synths allow you to set the range to between 0 and 12, some even up to 24 (two octaves)

The pitch bend value corresponding to a particular number of semitones pitch shift can be calculated by the following formula:

$$value = round\left(8192 + \frac{8192}{range} \cdot \Delta P_{semi}\right) \tag{A.1}$$

Or if cents are preferred:

$$value = round\left(8192 + \frac{8192}{range \cdot 100} \cdot \Delta P_{cents}\right) \tag{A.2}$$

Note that these formulae gives values from 0 to 16384, and that pitch bend values range from 0 to 16383. To solve this 16384 is replaced with 16383, the error introduced is unnoticable.

12	11	10	9	8	7	6	5	4	3	2	1
3FFF	3D55	3AAB	3800	3555	32AB	3000	2D55	2AAB	2800	2555	22AB
-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12
1D55	1AAB	1800	1555	12AB	1000	0D55	0AAB	0800	0555	02AB	0000

Table A.1: Pitchbend Values for Bend Range 12

24	23	22	21	20	19	18	17	16	15	14	13
3FFF	3EAB	3D55	3C00	3AAB	3955	3800	36AB	3555	3400	32AB	3155
12	11	10	9	8	7	6	5	4	3	2	1
3000	2EAB	2D55	2C00	2AAB	2955	2800	26AB	2555	2400	22AB	2155
-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12
1EAB	1D55	1C00	1AAB	1955	1800	16AB	1555	1400	12AB	1155	1000
-13	-14	-15	-16	-17	-18	-19	-20	-21	-22	-23	-24
0EAB	0D55	0C00	0AAB	0955	0800	06AB	0555	0400	02AB	0155	0000

Table A.2: Pitchbend Values for Bend Range 24

A.3 Pitch Slides

Pitch slides are instructions to the player to update the pitch bend value every subtick (which is 24 ppqn). *ticks* is the duration of the slide in number of rows. *tpb* is the number of ticks per beat. Δ_{value} is the desired resulting pitch bend change after *ticks* ticks have elapsed. *slide* is the parameter to use with the slide command.

$$slide = round\left(\Delta_{value} \cdot \frac{tpb}{subticks \cdot ticks}\right) \tag{A.3}$$

This equation can then be merged with the previous pitch formulae.

$$slide = round\left(\left(\frac{8192}{range} \cdot \Delta P_{semi}\right) \cdot \frac{tpb}{24 \cdot ticks}\right) \tag{A.4}$$

$$slide = round\left(\left(\frac{8192}{range \cdot 100} \cdot \Delta P_{cents}\right) \cdot \frac{tpb}{24 \cdot ticks}\right) \tag{A.5}$$

semi	12	11	10	9	8	7	6	5	4	3	2	1
12	3FFF	----	----	----	----	----	----	----	----	----	----	----
11	3D55	3FFF	----	----	----	----	----	----	----	----	----	----
10	3AAB	3D17	3FFF	----	----	----	----	----	----	----	----	----
9	3800	3A2F	3CCD	3FFF	----	----	----	----	----	----	----	----
8	3555	3746	399A	3C72	3FFF	----	----	----	----	----	----	----
7	32AB	345D	3666	38E4	3C00	3FFF	----	----	----	----	----	----
6	3000	3174	3333	3555	3800	3B6E	3FFF	----	----	----	----	----
5	2D55	2E8C	3000	31C7	3400	36DB	3AAB	3FFF	----	----	----	----
4	2AAB	2BA3	2CCD	2E39	3000	3249	3555	399A	3FFF	----	----	----
3	2800	28BA	299A	2AAB	2C00	2DB7	3000	3333	3800	3FFF	----	----
2	2555	25D1	2666	271C	2800	2925	2AAB	2CCD	3000	3555	3FFF	----
1	22AB	22E9	2333	238E	2400	2492	2555	2666	2800	2AAB	3000	3FFF
0	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000
-1	1D55	1D17	1CCD	1C72	1C00	1B6E	1AAB	199A	1800	1555	1000	0000
-2	1AAB	1A2F	199A	18E4	1800	16DB	1555	1333	1000	0AAB	0000	----
-3	1800	1746	1666	1555	1400	1249	1000	0CCD	0800	0000	----	----
-4	1555	145D	1333	11C7	1000	0DB7	0AAB	0666	0000	----	----	----
-5	12AB	1174	1000	0E39	0C00	0925	0555	0000	----	----	----	----
-6	1000	0E8C	0CCD	0AAB	0800	0492	0000	----	----	----	----	----
-7	0D55	0BA3	099A	071C	0400	0000	----	----	----	----	----	----
-8	0AAB	08BA	0666	038E	0000	----	----	----	----	----	----	----
-9	0800	05D1	0333	0000	----	----	----	----	----	----	----	----
-10	0555	02E9	0000	----	----	----	----	----	----	----	----	----
-11	02AB	0000	----	----	----	----	----	----	----	----	----	----
-12	0000	----	----	----	----	----	----	----	----	----	----	----

Table A.3: Pitchbend Values for Bend Ranges 1-12

Appendix B

File Formats

B.1 Configuration File Formats

B.1.1 The `zt.conf` File Format

The configuration file `zt.conf` is a text file which contains the settings for *ztracker*. Normally this is updated when exiting *ztracker* or when using the `ztconf` utility, however in rare cases you may want to edit it yourself using a text editor.

An example `zt.conf`:

```
screen_width: 640
screen_height: 480
default_pattern_length: 128
default_instrument_global_volume: 127
default_highlight_increment: 8
default_lowlight_increment: 8
key_repeat: 30
key_wait: 250
prebuffer_rows: 8
default_view_mode: 3
skin: default
send_panic_on_stop: no
midi_in_sync: no
fullscreen: no
auto_open_midi: yes
step_editing: yes
autoload_ztfile: no
centered_edit: no
autoload_ztfile_filename: autoload.zt
```

B.1.2 The `colors.conf` File Format

The configuration file `colors.conf` is a text file which contains the color settings for *ztracker*.

These entries contain the colors used to draw the beveled *ztracker* panel.

```
Background:     #A49054
Highlight:      #FFDC84
Lowlight:       #504428
```

Text that goes on the *ztracker* panel and on muted track names.

```
Text:           #000000
```

Text that goes above each track when they are not muted.

```
Brighttext:     #CFCFCF
```

Colors used for the information boxes at the top of the screen. `Black` is the background color, and `Data` is the text color.

```
Data:           #00FF00
Black:          #000000
```

Colors for the beat display at the bottom left corner of the *ztracker* window.

```
LCDHigh:        #FF0000
LCDMid:         #A00000
LCDLow:         #600000
```

The color of text that is in the pattern editor and all other boxes except the top info boxes.

```
EditText:       #808080
```

The background color of the pattern editor. `EditBGlow` on lowlight rows, `EditBGhigh` on highlight rows, and `EditBG` everywhere else.

```
EditBG:         #000000
EditBGlow:      #14100C
EditBGhigh:     #202014
```

`SelectedBGLow` is the background color of the pattern editor on a selected row which is not on a lowlight or highlight row. If a selected row is on a lowlight or highlight row, `SelectedBGHigh` is used instead.

```
SelectedBGLow:  #000080
SelectedBGHigh: #0000A8
```

`CursorRowLow` is the background color of the pattern editor on the row the cursor is positioned in if it is not a lowlight or highlight row. If it is positioned on a lowlight or highlight row, `CursorRowHigh` is used instead.

```
CursorRowLow:   #202020
CursorRowHigh:  #303030
```

An example `default.colors.conf`**:**

```
Background:     #A49054
Highlight:      #FFDC84
Lowlight:       #504428
Text:           #000000
Data:           #00FF00
Black:          #000000
EditText:       #808080
EditBG:         #000000
EditBGlow:      #14100C
EditBGhigh:     #202014
Brighttext:     #CFCFCF
SelectedBGLow:  #000080
SelectedBGHigh: #0000A8
LCDHigh:        #FF0000
LCDMid:         #A00000
LCDLow:         #600000
CursorRowHigh:  #303030
CursorRowLow:   #202020
```

B.2 Song File Format

The Song File format is the format *ztracker* uses to store songs on disk. *ztracker* song files have the `.zt` extension.

B.2.1 Current Song Format

Song File Header (`ZThd`)

```
0x00  char      chunk_name[4]  (="ZThd")
0x04  u_int32_t chunk_size
0x05  u_int8_t  bpm
0x06  u_int8_t  tpb
0x07  u_int8_t  max_tracks
0x08  u_int8_t  flags
              bit 0: SEND_MIDI_CLOCK
              bit 1: SEND_MIDI_STOP_START
              bit 2: FILE_COMPRESSED
              bit 3: SLIDE_ON_SUBTICK
              bit 4: SLIDE_PRECISION
            bit 5-7: UNDEFINED (always set to zero)
0x09  char      title[26]
0x23  <next chunk>
```

Song File Pattern Lengths (`ZTpl`)

```
0x00  char      chunk_name[4]  (="ZTpl")
0x04  u_int32_t chunk_size     (=256 * sizeof(u_int16_t) = 512)
0x08  u_int16_t pattern_length[256]
0x208 <next chunk>
```

Song File Order List (`ZTol`)

```
0x00  char      chunk_name[4]  (="ZTol")
0x04  u_int32_t chunk_size     (=256 * sizeof(u_int16_t) = 512)
0x08  u_int16_t orderlist[256]
0x208 <next chunk>
```

Song File Track Mutes (`ZTtm`)

```
0x00  char      chunk_name[4]  (="ZTtm")
0x04  u_int32_t chunk_size     (= max_tracks / 8)
0x08  u_int8_t  track_mutes[max_tracks / 8]
0x08 + chunk_size <next chunk>
```

Song File Instrument (`ZTin`)

```
0x00  char      chunk_name[4]  (="ZTin")
0x04  u_int32_t chunk_size
0x08  u_int8_t  inst_number
0x09  u_int16_t bank
0x0b  u_int8_t  patch
0x0c  u_int8_t  midi_device
0x0d  u_int8_t  channel + (flags << 4)
0x0e  u_int8_t  default_volume
0x0f  u_int8_t  global_volume
0x10  u_int16_t default_length
0x12  u_int8_t  transpose
0x13  char title[25]
0x2c  <next chunk>
```

Song File Event List (`ZTev`)

```
0x00  char      chunk_name[4]  (="ZTev")
0x04  u_int32_t chunk_size
0x08  <stream of events>
0x08 + chunk_size <next chunk>
```

The stream of events is encoded as follows:
Each event starts with a command byte (`u_int8_t cmd`). If `cmd` is between

`0x00` and `0x3f`, the next value is an `u_int8_t` (byte event). If `cmd` is between `0x40` and `0x7f`, the next value is an `u_int16_t` (word event). If `cmd` is between `0x80` and `0xbf`, the next value is an `u_int32_t` (dword event). If `cmd` is between `0xc0` and `0xff`, the next value is an `u_int16_t` telling how many string bytes that follow. (string event)

In the current Song File revision only byte events and word events are used. The data first gets set up. After that the data is inserted at the appropriate row by issuing the `0x41` command. For each new row, only the data that is actually different from the previous row need to be inserted.

```
Byte events:
  0x01 Note
  0x02 Instrument
  0x03 Volume
  0x04 Effect
  0x05 Track (default is initially 0)
  0x06 Pattern (default is initially 0)

Word events:
  0x41 Insert event at row x
  0x42 Length
  0x43 Effect data
```

B.2.2 New Song Format

This is intended as a new format completely replacing the old format for introduction somewhere before *ztracker* 1.0. Parts of this specification will under a transitional period be used together with the old format header.

Song File Header (`ZTHD`)

(to be defined)

```
0x00  char      chunk_name[4]  (="ZTHD")
0x04  u_int32_t chunk_size
0x08  u_int16_t name_len        | <= short string
0x0a  char      name[name_len]  | (not 0 terminated)
<to be defined>
0x08 + chunk_size <next chunk>
```

Song File Song Message (`SMSG`)

(new as of *ztracker* 0.94)

```
0x00  char      chunk_name[4]  (="SMSG")
0x04  u_int32_t chunk_size
0x08  u_int32_t song_message_len                  | <= long string
0x0c  char      song_message[<song_message_len>]  | (not 0 terminated)
0x08 + chunk_size <next chunk>
```

Song File Arpeggio (ARPG)

(new as of *ztracker* 0.94)

```
0x00  char      chunk_name[4]  (="ARPG")
0x04  u_int32_t chunk_size
0x08  u_int16_t number
0x0a  u_int16_t name_len          | <= short string
0x0c  char      name[<name_len>]  | (not 0 terminated)
0x0c  u_int16_t length
0x0c  u_int8_t  num_cc
0x0c  u_int16_t speed
0x0c  u_int16_t repeat_pos
0x0c  u_int8_t  cc[<num_cc>]
0x0c  u_int16_t pitch[<num_entries>]
0x0c  u_int8_t  ccval[<num_cc>][<num_entries>]
0x08 + chunk_size <next chunk>
```

Song File Midi Macro (MMAC)

(new as of *ztracker* 0.94)

```
0x00  char      chunk_name[4]  (="MMAC")
0x04  u_int32_t chunk_size
0x08  u_int16_t number
0x0a  u_int16_t name_len          | <= short string
0x0c  char      name[<name_len>]  | (not 0 terminated)
0x0c + name_len  u_int16_t num_entries
0x0e + name_len  u_int16_t mididata[<num_entries>]
0x08 + chunk_size <next chunk>
```

Song File Pattern (PATT)

(to be defined)

```
0x00  char      chunk_name[4]  (="PATT")
0x04  u_int32_t chunk_size
0x08  u_int16_t number
0x0a  u_int16_t name_len          | <= short string
0x0c  char      name[<name_len>]  | (not 0 terminated)
```

```
<to be defined>
0x08 + chunk_size <next chunk>
```

Song File Instrument (INST)

(to be defined)

```
0x00  char      chunk_name[4]  (="INST")
0x04  u_int32_t chunk_size
0x08  u_int16_t number
0x0a  u_int16_t name_len          | <= short string
0x0c  char      name[<name_len>]  | (not 0 terminated)
<to be defined>
0x08 + chunk_size <next chunk>
```

Song File Track Mutes (TMUT)

(to be defined)

```
0x00  char      chunk_name[4]  (="TMUT")
0x04  u_int32_t chunk_size
<to be defined>
0x08 + chunk_size <next chunk>
```

Song File Order List (OLST)

(to be defined)

```
0x00  char      chunk_name[4]  (="OLST")
0x04  u_int32_t chunk_size
0x0c + name_len  u_int16_t num_entries
0x0e + name_len  u_int16_t order[<num_entries>]
0x08 + chunk_size <next chunk>
```

B.3 Skin File Format

NOTE: This format is no longer in use. This section is provided only for the sake of completeness.
Skin files are containers for a group of files, much like zip archives. An application can load a contained file from a skin file by refering to its file name. In *ztracker* skin files are use to contain graphics data which *ztracker* needs.

Each file to be contained in the skin is encoded according to the following scheme and then concatenated together to form a skin file. The encoding is split into two parts, the *header*, and the *payload*. All data in the header is stored in little endian order. (i.e the least significant byte is always first)

Each entry begins with the header. First is the name of the contained file. `namelen` is the length of the name, and `name` is the actual name without a trailing 0:

```
0x00          u_int32_t namelen
0x04          char name[namelen]
```

Then follows the offset in bytes from the beginning of the skin file to where the payload begins:

```
0x04+namelen  u_int32_t dataoffset
```

After that follows the uncompressed size, which is the size of the payload will be after decompression:

```
0x08+namelen  u_int32_t realsize
```

Then follows the compressed size, which is the size of the payload before decompression:

```
0x0c+namelen  u_int32_t compressedsize
```

After that follows a flag which tells us if the payload is compressed. If `compressedflag` is 0 the payload is uncompressed. If `compressedflag` is 1 the payload is compressed.

```
0x10+namelen  u_int32_t compressedflag
```

Last follows the payload. If `compressedflag` is 1 this must be encoded in a way compliant with the zlib specification algorithm [3]. If `compressedflag` is 0 this is just a copy of the whole file.

```
dataoffset    char payload[compressedsize]
```

Appendix C

Troubleshooting

Fill in the blank!

Appendix D

Contributing to *ztracker*

D.1 Introduction

ztracker is an open source project released under the BSD license. This means you are very welcome to participate in the development of *ztracker*. The license guarantees that you work on equal terms with the other developers. The most urgent need is help writing documentation. Visit the *ztracker* web site at http://ztracker.sourceforge.net/ for more information.

D.2 Bug/request

There is a bug and request tracking system at http://ztracker.sourceforge.net/. Here you can submit bugs, and add suggestions for new features. You can also send comments, questions, feedback, bugreports, and flames to: <ztracker-feedback@lists.sourceforge.net>.

D.3 Mailing lists

<ztracker-devel@lists.sourceforge.net> is a mailing list for internal developer discussions. Feel free to join and read, however non-technical suggestions should be directed to the <ztracker-feedback@lists.sourceforge.net> address.

D.4 Making Skins

Skins are a collection of graphics files and a color description that *ztracker* uses for its apperance.

```
directory skins\default\
colors.conf
font.fnt
```

`buttons.png`
`logo.png`
`load.png`
`save.png`
`toolbar.png`
`about.png`

The file `font.fnt` is the font used by *ztracker*. To edit this you need itf 1.65 or a similar font editor.
The file `colors.conf` lists the colors that ztracker shall use. Its format is described in the File Formats chapter.
When your `.png`'s have been created you may run `pngcrush` on them to compress them further. This does not affect the appearance of the graphics, but instead just optimizes the way the `.png`'s are stored on disk. The `default` and `professional` skins were reduced by some 20% using this method.

D.5 Coding

Fill in the blank!

D.6 Writing Documentation

The documentation is written in LaTeX. Good help on how to write documents in LaTeX can be found in the book *The LaTeX Companion* [1], and the freely downloadable *The Not So Short Introduction to LaTeX 2ε* [2].

If you are writing documentation on a *Windows* machine, we recommend that you use MiKTeX `http://www.miktex.org/`, which is an easy to use LaTeX distribution.

Appendix E

Glossary

tick A tick is when the player moves to the next row.

subtick There are many subticks per tick. How many?

player command A special instruction to the player. Has the form `Xyyzz`.

hexadecimal number A number written in base=16 instead of base=10 as usual. (0 = `0`, 1 = `1`, ..., 9 = `9`, 10 = `A`, ..., 15 = `F`, 16 = `10`, ..., 31 = `1F`, ...

Appendix F

Acknowledgements

First of all credits to Christopher Micali who came up with the idea of *ztracker* and wrote nearly all of the code.

Thanks to Donald E. Knuth who wrote and currently maintains TEX, the motor which enables LATEX to work.

Patrik Wallström helped out alot on the chapter partitioning of this manual.

Bibliography

[1] Michel Goossens, Frank Mittelbach and Alexander Samarin: *The LaTeX Companion*, Second Printing, 1994, Addison Wesley, ISBN 0-201-54199-8

[2] Tobias Oetiker, Hubert Partl, Irene Hyna and Elisabeth Schlegl: *The Not So Short Introduction to LaTeX 2ε*, Version 3.19, 02 April, 2001 ftp://ftp.tex.ac.uk/tex-archive/info/lshort/english/lshort.pdf

[3] L. Peter Deutsch and Jean-Loup Gailly: *RFC1950 ZLIB Compressed Data Format Specification*, Version 3.3, 1996 ftp://ftp.uu.net/graphics/png/documents/zlib/zdoc-index.html

[4] Donald E. Hall: *Musical Acoustics*, Second Edition, 1991, Brooks/Cole, ISBN 0-534-13248-0