Mark Henning

Specifications

AnaMark tuning file format (*.tun; *.tun.*; *.msf)

Version 2.00

17th July 2009

Mark Henning, Germany
http://www.mark-henning.de

Contents

1 Introduction

The tuning file format is a capable yet simple format for scale data (→ micro tuning). Advantages:

- Easy implementation of read/write functionality: **In most cases there is no need to implement the complete specifications!**
- Database functionality: additional informations associated with the scale
- Multiple scales per file: scales can be assigned to MIDI channels
- Embedding functionality: data can be embedded directly into e.g. HTML

Restrictions (mainly one):

- Limited editing: The format provides some algorithmic capabilities to build a scale. However, it is *not* intended for scale editing as each scale editor has its own specific features.

Free C++ source code for reading/writing AnaMark tuning files as well as open source tools for editing and converting them is available for download at:
http://www.mark-henning.de/eternity/tuningspecs.html

1.1 Format history and Compatibility

- **Version 0:**
  The software synthesizer VAZ 1.5 Plus by Software Technology[1] used a very basic implementation of .tun files.

  The format contained only the section `[Tuning]` and thus had some essential shortcomings which hindered it from spreading.

- **Version 1:**
  Extended tuning file format used in the VSTi software synthesizer AnaMark by Mark Henning[2]. This tuning file format became one of the most used formats.

[1] http://www.software-technology.com
[2] http://www.mark-henning.de

Version 1 is downwards compatible (thus called AnaMark/VAZ 1.5 Plus tuning file format), and contains the new section `[Exact Tuning]` allowing exact tuning specification and the definition of a base frequency. The specifications and example source code for reading/writing are available for free.

The present format specification (version 2.00, called *AnaMark tuning file*) is mostly downwards compatible. Minor compromises were necessary due to current needs, but most files should work with older software.[3]

[3] You may use the free TUN-Tools available at
http://www.mark-henning.de/eternity/tuningspecs.html
to convert between the different format versions.

2 Basic file structure

- Files are 8-Bit ASCII.
- Max. line length 1 000 000 bytes[1]
- Allowed line delimiters are: 0x00, 0x0a, 0x0d. *rightarrow* WIN/DOS-style as well as UNIX-style files may be used.
- If in these specs parts of the file are declared as "to be ignored", consider that these parts may have *any* content, of *any* syntax. So you should really *ignore* them, especially do *not* perform syntax checking there.
- Leading/trailing white spaces have to be removed before further processing a line.
- Empty lines are ignored
- Lines beginning with a ';' are comment lines. They are ignored.
- The file is subdivided into sections. Lines beginning with a '[' mark the start of a new section. A section may (but does not have to) contain key-value definitions and/or function calls.

2.1 Sections

A section start has the format

```
[sectionname]
```

where the section name *sectionname* is *not* case sensitive. There are section names defined with blanks within, e.g. `[Exact Tuning]`. The string `[sectionname]` must be the one and only string in the line, there must be no leading/following characters.

All lines preceeding the first section of a file have to be ignored. The content of unknown sections has to be ignored. These cases may produce a warning, but not an error.

[1] For downwards compatibility: It is recommended to keep lines below 255 characters, which was the limit in the version 1 specifications.

A section ends, when the next section starts or the file end is reached.

Within one scale dataset (see 3.2 for details), a section must not be repeated / divided into several parts. So this would not be O.K.:

```
[Section1]
; Content of Section1

[Section2]
; Content of Section2

[Section1]
; ERROR, because Section1 is repeated
```

2.2 Key-value pairs

Unless specified otherwise, within sections key values are set using

```
key = value
```

where *key* denotes the name of the key; it is *not* case sensitive. *value* represents the value assigned to the key. The key must begin with an underscore or a normal letter (a–z). The first '=' in the line separates key and value.[2]

NOTE: Key names may contain white spaces *within*. White spaces within key names act as separators and are used e.g. when a key refers to an array of values, e. g.:

```
note 1 = 100
note 2 = 200
note 3 = 300
; ... and so on ...
```

IMPORTANT: Key assignments have to be done immediately in the order they occur in the file.

The formatting of the values is key specific, but always single-line. Typical formats are:

- integer values: written as plain integer number e.g.

  ```
  keyA = 123
  keyB = -123
  ```

- float values: written as integer or as plain floating point number, e.g.

[2] A line may contain more than one '=', e. g. if the value is a string!

```
keyA = 123.45
keyB = -123.45
```

- scientific values: written as float value or as plain scientific numbers, e.g.

```
keyA = 1.23e2
keyB = -1.23e-2
```

- string values: enclosed in quotation marks using C-style conversions of special characters, e.g.

```
keyA = "\n line break, \t tabulator and hex ch\0x41racter"
```

- string list entry: written as string value

  Keys specifying string lists can be written multiple times. Each value is appended at the end of the string list, e.g.

```
keyA = "Hello"
keyA = "how are"
keyA = "you?"
```

  This results in the string list `keyA` containing the three entries "Hello", "how are" and "you?".

Key values are section specific. The key `KeyA` in Section `[SectionA]` is thus different from the key `KeyA`, but located in Section `[SectionB]`.

IMPORTANT: Unknown keys and their values have to be ignored. This may produce a warning, but not an error.

2.3 Function calls

Within sections functions are called using

```
function = (parameters)
```

where *function* denotes the name of the function. The same syntax than that of key-value pairs is used here (see 2.2). The only difference is that the value is enclosed in round brackets containing no, one or several values (separated by ',').

3 The sections

3.1 Overview

In the table below, sections are labelled "'recommended"', if their inclusion in the file is optional due to downwards compatibility. The abbreviations are:

REQ Required

rec Recommended

opt Optional

Section/ `required key`	Content	Existence	Since
Scale Begin	Start of a scale dataset	REQ	V2.00
`Format`	= `"AnaMark-TUN"`		V2.00
`FormatVersion`	= `"200"`		V2.00
`FormatSpecs`	= `"http:\\www.mark-henning.de\` $\rightsquigarrow$ `eternity\tuningspecs.html"`		V2.00
Info	Informations about the scale	REQ	V2.00
`Name`	Name of the scale		V2.00
`ID`	Scale identifier		V2.00
Assignment	Assign scale to a MIDI channel	opt	V2.00
Tuning	Quantized scale (cents)	rec	V0.00
`note` x	note tuning (cents)		V0.00
Exact Tuning	Exact scale (cents), base frequency (Hz)	rec	V1.00
Functional Tuning	Algorithmic scale definition	REQ	V2.00
Mapping	Keyboard mapping	opt	V2.00
Editor Specifics	Editor specific settings	opt	V2.00
Scale End	End of a scale dataset	REQ	V2.00

3.2 Begin and end of a scale dataset

3.2.1 Section [Scale Begin]

<Required in version 2.00> A scale dataset must start with the section

```
[Scale Begin]
```

It contains the following keys:

- `Format` = *string value* **<required>**

  Default value: `"AnaMark-TUN"`

  This identifier declares the format of the following scale dataset. To indicate a format according to the present specs, set the value to `"AnaMark-TUN"`.

- `FormatVersion` = *integer value* **<required>**

  Default value: `100`

  Format version. To indicate a format according to the present specs, set the value to `200`. The value `100` denotes AnaMark/VAZ 1.5 Plus tuning file format (version 1 tuning files).

- `FormatSpecs` = *string value* **<required>**

  Default value: `"http:\\www.mark-henning.de\eternity\tuningspecs.html"`

  Links to the website containing the official format specs. The URL for these specs is `http:\\www.mark-henning.de\eternity\tuningspecs.html`.

3.2.2 Section [Scale End]

<Required in version 2.00> A scale dataset must end with the section

```
[Scale End]
```

The content of this section has to be ignored.

3.2.3 Files containing multiple scales (*.MSF files)

It is possible to put several scale datasets within one file. This allows building databases or — using the section `[Assignment]` (see 3.4) — files containing multiple scales which are assigned to different MIDI channels. There is no restriction on the maximum number of datasets per file. The file extension is **MSF** which means **M**ultiple**S**cales**F**ile. The structure is straight forward:

```
[Scale Begin]
; The data of the first scale dataset
[Scale End]

[Scale Begin]
; The data of the second scale dataset
[Scale End]

[Scale Begin]
; The data of the third scale dataset
[Scale End]

; ...and so on...
```

If for a MIDI channel no scale is specified, the default tuning is used. If two scales apply to the same MIDI channel, the first scale in the file is used.

IMPORTANT: If a scale does not have an `[Assignment]` section with MIDI channel restrictions, it is considered applicable to each MIDI channel. Thus all following scales in the MSF-File are ignored!

3.2.4 Embedding scales in e.g. HTML

As all lines outside

```
[Scale Begin]
; Data of the scale dataset
[Scale End]
```

are ignored, it is possible to embed scale datasets in other text files such as e.g. HTML:

```
<HTML>
<BODY>

Here I can write something about the scale.
I can open the file directly with my browser
viewing the description and open the same
file directly in my music software.

As long as no line starts with a '[' the
lines are ignored.

<!--
```

```
  I don't wanna see the scale date in the browser,
  therefore I put it into a HTML comment.

  So here's the scale dataset:

  [Scale Begin]
  ; Data of the scale dataset
  [Scale End]

  I can embed more than one scale in the file,
  thus it becomes a Multiple Scales File (MSF):

  [Scale Begin]
  ; Let's have another one here...
  [Scale End]

-->
</BODY>
</HTML>
```

Maybe you want downwards compatibility. To embed a scale dataset so that even older software supporting only AnaMark/VAZ 1.5 Plus tuning files (= version 1) can deal with it, you a restricted to one scale per file and have to consider a few limitations. Here is how it looks like:

```
<HTML>
<BODY>

Let's see how scale embedding is done
downwards compatible...

According to the version 1 specs, everything
trailing the first section has to be ignored,
so we are free to write anything here as
long as it does not look like a section.

<!--
  I don't wanna see the scale date in the browser,
  therefore I put it into a HTML comment.

  So here's the scale dataset:
```

```
[Scale Begin]
; Data of the scale dataset
[Scale End]

; To ensure, that the rest of the file does
; not produce an error we have to put it
; into a comment:
; --></BODY></HTML>
```

Note, that you should *not* write this:

```
; -->
; </BODY>
; </HTML>
```

This will make the ';' visible in your browser and may corrupt your HTML syntax. As an alternative you can fake the `key = value` syntax. Unknown keys are ignored according to the version 1 specs, thus we are fine writing:

```
FakeKey = --></BODY></HTML>
```

To distinguish files containing embedded scales, put an extra ".tun" before the original file extension. The file in the above example may be called *MyEmbeddedScaleFile.tun.html*. To filter a list of files for embedded scales use the filter "*.tun.*".

3.3 Section [Info]

<Required in version 2.00> The section

```
[Info]
```

contains informations associated with the scale. The keys are:

- `Name =` *string value* **<required>**

  Default value: *file name*

  Name of the scale

- `ID =` *string value* **<required>**

  Default value: "ID_" + *file name without spaces*

  A shorter alternate identifier. It must begin with an alphabetic character (a–z or A–Z) or an underscore and must not contain white spaces.

- `Filename = string value` **<recommended>**

  Default value: *current file name without the extension*

  Filename suggested for the scale *without the extension*. When putting multiple scales together in one file this gives the ability to re-extract them with their original file names.

- `Author = string value` **<optional>**

  Default value: *empty string*

  Who created the scale

- `Location = string value` **<optional>**

  Default value: *empty string*

  Where the author resides

- `Contact = string value` **<optional>**

  Default value: *empty string*

  Contact information of the author

- `Date = string value` **<optional>**

  Default value: *empty string*

  Creation date of the scale. Date format is YYYY-MM-DD, according to ISO 8601.

- `Editor = string value` **<optional>**

  Default value: *empty string*

  Software used to create/edit the file/scale.

- `EditorSpecs = string value` **<optional>**

  Default value: *empty string*

  URL containing the specs of the section `[Editor Specifics]` (see 3.9).

- `Description = string value` **<optional>**

  Default value: *empty string*

  Description of the scale.

- `Keyword = string list entry` **<optional>**

  Default value: *empty string list*

  Keywords which may help categorizing the scale.

- `History = string value` **<optional>**

  Default value: *empty string*

  Description of the "historical context" of the scale.

- `Geography = string value` **<optional>**

  Default value: *empty string*

  Description of the "geographical context" of the scale.

- `Instrument = string value` **<optional>**

  Default value: *empty string*

  Description of the instrument the scale is typical for.

- `Composition = string list entry` **<optional>**

  Default value: *empty string list*

  Compositions in which the scale was used. Format:

  ```
  Musician or Band|Album|Title|Year|Misc
  ```

  As the '|' is the separator, it is not allowed within the field values.

- `Comments = string value` **<optional>**

  Default value: *empty string*

  Comment for additional informations not covered by the other keys.

3.4 Section [Assignment]

<Optional> The section

```
[Assignment]
```

is intended for files containing multiple scales (MSF-Files, see 3.2.3) to assign the current scale dataset to MIDI channels. However, it can also be used in TUN files containing single scales only.[1] The keys are:

[1] If a software supports only standard single scale TUN files, it might ignore this setting and take the scale valid for each MIDI channel. If a software supports MSF-Files, it has to consider this field in TUN-files too.

- `MIDIChannels = string value` **<optional>**

  Default value: *empty string*

  Defines the MIDI channel(s) where the scale is to be applied. A scale can be assigned to multiple MIDI channels or to a range of MIDI channels, e.g.:

```
MIDIChannels = "1-3,5,7"
```

  This assigns the scale to the MIDI channels 1 to 3, 5 and 7. The range of MIDI channel numbers is from 1 to 65535. If the key is not specified or an empty string is given, the scale is applied to each MIDI channel.

3.5 Section [Tuning]

<Recommended to provide version 0 data> The section

```
[Tuning]
```

contains a quantized scale in cents (integer values). The keys are:

- `note x = integer value` **<required>**

  Default value: $100 \cdot x$

  x denotes the MIDI note number (No keyboard mapping supported here!). Keys must be given for each note number from 0–127. If a note is missing, this may produce a warning. The value is the relative tuning in cents referred to 8.1757989156437073336 Hz (corresponds to the standard tuning A=440 Hz ('A' is MIDI note 69).

Example for a valid `[Tuning]` section representing the default settings:

```
[Tuning]
note 0 = 0
note 1 = 100
note 2 = 200
; This has to be continued for each note
; in the range 3-127, including 127
```

This section has the lowest priority of all tuning sections. This means: It is ignored if any of the sections `[Exact Tuning]` or `[Functional Tuning]` is given.

3.6 Section [Exact Tuning]

<Recommended for version 1 downwards compatibility> The section

```
[Exact Tuning]
```

contains an exact scale in cents (scientific values). It additionally provides a base frequency and autocomplete functionality for periodic scales. The keys are:

- `BaseFreq =` *scientific value* **<optional>**

  Default value: 8.1757989156437073336

  The absolute frequency in Hz. Its default value corresponds to the standard tuning A=440 Hz ('A' is note 69).

- `note x =` *scientific value* **<optional>**

  Default value: $100 \cdot x$

  x denotes the MIDI note number and lies in the range 0–127 (No keyboard mapping supported here!) The value is the relative tuning in cents referred to `BaseFreq`.

After processing the `[Exact Tuning]` section, auto completion is performed as described in the following section

This section has medium priority. This means: It overwrites the values given in an `[Tuning]` and is ignored if the section `[Functional Tuning]` is given.

3.6.1 Auto completion (Periodic scales)

Let `H` be the highest MIDI note number explicitly specified in the section, let `P` be its tune given in the file (called `P`, because it is the period length). `f(x)` is the current tune value of the note with the MIDI note number `x`, which is initially set to the values given in the file.

First ensure that the tunes `f(x)` of all scale notes in the range [0; `H`] which are not set in the file represent the default values `100*x`. Then complete the tunings [`H`; 127] according to this algorithm:

```
if ( H < 127 ) then
  from i=H to i=127 including 127, stepsize 1 do:
    Let f(i) = f(i-H) + P
  end from
end if
```

3.6.2 Example of a valid section

Example for a valid "[Exact Tuning]" section using auto completion:

```
; All 'E' should be tuned +12.5 cents compared with
; the default scale. All other notes should have the
; default tuning.

[Exact Tuning]
; Set the tune of the first E:
note 4 = 412.5
; To let the period be one octave,
; we have to set the "periodic point":
note 12 = 1200
```

3.7 Section [Functional Tuning]

<Required for version 2.00> The section

```
[Functional Tuning]
```

contains a more versatile definition of the scale which also allows some exact algorithmic calculations. The keys are:

- `note x =` *string value* **<optional>**

  Default value: `""`

  x denotes the scale note number and lies in the range 0–127 (Here, keyboard mapping is supported, thus x does NOT represent the MIDI note number!) Definition of the formula to calculate the notes frequency.

The following function calls are available:

- `InitEqual = (BaseNote,BaseFreqHz)` **<optional>**

  `BaseNote`: *integer value*

  `BaseFreqHz`: *scientific value*

  Set the frequency $f(x)$ of each note x to

$$f(x) = BaseFreqHz \cdot 2^{\frac{x-BaseNote}{12}} \tag{3.1}$$

When the section is entered a virtual function call

```
InitEqual = (69,440)
```

must be executed which leads to the standard tuning A=440 Hz ('A' is note 69).

This section has highest priority. This means: It overwrites the values given in any of the other tuning sections ([Tuning] or [Exact Tuning]).

3.7.1 Formula syntax

In a formula you can define parameters for an equation to calculate the note frequency. Furthermore there are special command tokens, explained later on.

The frequency assigned to a note x when a key `note x` is found, is calculated using the equation

$$f(x) = f_{\text{Range}} \cdot \frac{MUL}{DIV} \cdot 2^{CENTS/1200} + f_{\text{Shift}} \tag{3.2}$$

where x denotes the index of the current note and $f(x)$ their frequency in Hz. The default values are:

$$\begin{aligned} f_{\text{Range}} &= f(x) \\ MUL &= 1 \\ DIV &= 1 \\ CENTS &= 0 \text{ cents} \\ f_{\text{Shift}} &= 0 \text{ Hz} \end{aligned} \tag{3.3}$$

In the formula string, the variables of this equation are referred to by one-char tokens followed by the value of the variable (spaces are allowed to improve readability). The tokens and their meaning are:

- '#' = f_{Range} (Frequency range in Hz)
- '*' = MUL (Multiplication factor)
- '/' = DIV (Divisor)
- '%' = $CENTS$ (Value in cents by which the frequency is increased)
- '+' = f_{Shift} (Frequency shift in Hz)

After the tokens, the values are given directly as positive or negative float value, e.g. `"123.456"`. However, f_{Range} and f_{Shift} can also be given by reference:

- As absolute reference to another note's frequency. The value is an integer preceeded by `'='`.

- As relative reference to another note's frequency. The value is an integer preceeded by `'>'`.

IMPORTANT: If a token is found more than once in the formula string, only the last occurence will be effective! But you may use the same key several times alternating a notes frequency by self-reference.

An example will demonstrate the possibilities:

```
InitEqual = (0,8)
note 0 = ""
note 1 = "*2 /3"
note 2 = "#>-1 %1200 +-3"
note 3 = "#=1"
note 1 = "*3 /2"
```

This results in

$$\begin{aligned} f(0) &= 8\text{ Hz} \\ f(1) &\approx 8.4757\text{ Hz} \\ f(2) &\approx 8.3010\text{ Hz} \\ f(3) &\approx 5.6505\text{ Hz} \end{aligned} \tag{3.4}$$

as

1. `InitEqual = (8)`

   Initialize scale using 8Hz as base frequency, this means:

   $$\begin{aligned} f(0) &= 8\text{ Hz} \\ f(1) &\approx 8.4757\text{ Hz} \\ f(2) &\approx 8.9797\text{ Hz} \\ f(3) &\approx 9.5137\text{ Hz} \end{aligned} \tag{3.5}$$

   ...and so on...

2. `note 0 = ""`

   Let note 0 be where it is as placing the default values eq. (3.3) into eq. (3.2) gives:

   $$f(0) = 8\text{ Hz} \cdot \frac{1}{1} \cdot 2^{0/1200} + 0\text{ Hz} = 8\text{ Hz} \tag{3.6}$$

3. `note 1 = "*2 /3"`

$$\Rightarrow f(1) = f(1) \cdot \frac{2}{3} \cdot 2^{0/1200} + 0 \text{ Hz}$$
$$\approx 8.4757 \text{ Hz} \cdot \frac{2}{3} = 5.6505 \text{ Hz} \tag{3.7}$$

4. `note 2 = "#>-1 %1200 +-3"`

$$\Rightarrow f(2) = f(2-1) \cdot \frac{1}{1} \cdot 2^{1200/1200} + (-3 \text{ Hz})$$
$$\approx 5.6505 \text{ Hz} \cdot 2 - 3 \text{ Hz} = 8.3010 \text{ Hz} \tag{3.8}$$

5. `note 3 = "#=1"`

$$\Rightarrow f(3) = f(1) \cdot \frac{1}{1} \cdot 2^{0/1200} + 0 \text{ Hz}$$
$$= 5.6505 \text{ Hz} \tag{3.9}$$

6. `note 1 = "*3 /2"`

$$\Rightarrow f(1) = f(1) \cdot \frac{3}{2} \cdot 2^{0/1200} + 0 \text{ Hz}$$
$$\approx 5.6505 \text{ Hz} \cdot \frac{3}{2} = 8.4757 \text{ Hz} \tag{3.10}$$

3.7.2 Formula command tokens

The followin command tokens can be used in formulas:

- '~' = loop
- '!' = ensure frequency in Hz by shifting the scale

An example for the loop token:

```
note 3 = "*2 ~5"
note 100 = "*3 ~-3"
```

is equivalent to:

```
note 3 = "*2"
note 4 = "*2"
note 5 = "*2"
note 6 = "*2"
note 7 = "*2"
note 100 = "*3"
note 99 = "*3"
note 98 = "*3"
```

This can be used for periodic scales. Assume that your scale is the equal tempered scale except that all notes A are detuned by +22 cents. You can do it like this:

```
note 9 = "%22"
note 21 = "%22"
note 33 = "%22"
note 45 = "%22"
note 57 = "%22"
note 69 = "%22"
note 81 = "%22"
note 93 = "%22"
note 105 = "%22"
note 117 = "%22"
```

or much shorter:

```
note 9 = "%22"
note 12 = "#>-12 *2 ~116"
```

If a function is to be looped until the begin or end of the available notes, use `~-999` or `~999`, respectively. So you don't need to calculate exactly how many notes are left and you can write:

```
note 9 = "%22"
note 12 = "#>-12 *2 ~999"
```

Finally, the token '!' shifts the scale, so that the current note has the given frequency in Hz. Assume that your scale looks like this:

$$\begin{aligned} f(0) &= 8\,\text{Hz} \\ f(1) &= 9\,\text{Hz} \\ f(2) &= 10\,\text{Hz} \\ f(3) &= 11\,\text{Hz} \end{aligned} \tag{3.11}$$

then entering

```
note 2 = "!440"
```

will result in

$$\begin{aligned} f(0) &= 352 \text{ Hz} \\ f(1) &= 396 \text{ Hz} \\ f(2) &= 440 \text{ Hz} \\ f(3) &= 484 \text{ Hz} \end{aligned} \tag{3.12}$$

So, if you want to ensure that in any case, the note A equals 440 Hz, just put

```
note 69 = "!440"
```

at the end of your scale definition.

IMPORTANT: The token '!' must be the only token in the formula. Other tokens might be ignored, thus the result is not defined!

3.8 Section [Mapping]

<Optional> The section

```
[Mapping]
```

defines the keyboard mapping of the scale defined in the section `[Functional Tuning]`[2]

MIDI note number sent $\rightarrow$ *scale note number sounding*

The keys are:

- `Keyboard x =` *integer value* **<optional>**

  Default value: x

  x denotes the MIDI note number sent, the integer value assigned denotes the scale note number sounding.

- `LoopSize =` *integer value* **<optional>**

  Default value: 0

  The keyboard mapping is looped, if `LoopSize` > 0. The value $N(x)$ for the key `Keyboard x` with $x \geq$ `LoopSize` is

$$\begin{aligned} \text{Octave} &= \text{int}(x \,/\, \texttt{LoopSize}) \\ \text{Offset} &= x \bmod \texttt{LoopSize} \\ N(x) &= N(\text{Offset}) + \text{Octave} \cdot \texttt{LoopSize} \end{aligned} \tag{3.13}$$

[2] It is **not** to be applied to the sections `[Tuning]` and `[Exact Tuning]`!

3.9 Section [Editor Specifics]

<Optional> The section

```
[Editor Specifics]
```

allows the scale editor to store editor specific informations. It must *not* be placed ahead the section `[Info]` but somewhere behind it as its contents depend on the key `Editor` there!

The software is free to place any kind of textual content here but must ensure that there is no line beginning with a '['!

4 Acknowledgments

I am grateful to Aaron Andrew Hunt for his critical remarks and suggestions, especially concerning the database part of these specifications: the key definitions are arranged with the TuningXML format.