Open file format for MR sequences

Version 1.3.1

Maxim Zaitsev
Stefan Kroboth
Kelvin Layton

Medical University of Vienna
maxim.zaitsev@meduniwien.ac.at
University Medical Centre Freiburg
maxim.zaitsev@uniklinik-freiburg.de

This file specification is part of the Pulseq project:
http://pulseq.github.io/

Contents

Revision History

Version	Date	Description
1.0	26 Jun 2014	Draft specification
1.01	11 Jun 2015	Included draft of binary specification
1.1.0	11 Jul 2017	Changed versioning scheme
1.2.0	06 Jul 2018	Events can now be delayed individually; Delay events and other events overlap within blocks
1.2.1	13 Dec 2018	Matlab code does not use zero-filling prior to the actual RF shape to account for RF dead time and uses delay instead. Interpreter code also does not attempt to detect RF zero-filling.
1.3.0	02 Jul 2019	Support for generic extensions: added extensions column to the event table, which references the new [extensions] section; added support for one extension: triggers (including two types: input and output)
1.3.1	25 Sept 2020	Added support for the LABEL extension, updated figures

1 Introduction

The purpose of this file format is to compactly represent a magnetic resonance (MR) sequence in an open and vendor independent manner. Sequences for both NMR spectroscopy and MRI can be represented. The format is intended for research and educational purposes and currently omits complex sequence features such as physiological triggering or logical coordinate-frame transformations. The format has been developed with the following **design goals**:

1. **Human-readable:** The basic sequence structure should be easily understood without processing.
2. **Easily parsed:** The format should be easy for a computer program to parse without the need for external libraries.
3. **Vendor independent:** The sequence format must not contain definitions specific to a particular hardware manufacturer.
4. **Compact:** The sequence must not contain redundant definitions. As such, re-use of existing definitions at different times is inherent in the sequence format.
5. **Low-level:** The format should be sufficiently low-level. This allows for

maximum flexibility of sequence specifications and does not limit the high-level design tools.

The first goal of human readability necessitates a text-file format. The file format is intended to describe a sequence that can be run on scanner hardware. Therefore, the second goal of machine readability ensures that the file can be read and interpreted without the use of external libraries that might be unavailable on different platforms. This prohibits the use of an existing markup language like XML, which are not straightforward to parse. Further, units that are inherently hardware dependent have been avoided, such as 'Volts' or 'Tesla'.

1.1 Example

Before defining the detailed specification, a simple example is presented to demonstrate the main structure of a sequence. Below is a simple FID experiment with RF pulse, delay and ADC readout.

```
# Pulseq sequence file
# Created by MATLAB mr toolbox

[VERSION]
major 1
minor 3
revision 1

[DEFINITIONS]
Name fid

# Format of blocks:
#  #  D RF  GX  GY  GZ ADC EXT
[BLOCKS]
1  0  1   0   0   0  0  0
2  1  0   0   0   0  0  0
3  0  0   0   0   0  1  0

# Format of RF events:
# id amplitude mag_id phase_id delay freq phase
# ..        Hz   ....     ....    us   Hz   rad
[RF]
1         2500 1 2 100 0 0

# Format of ADC events:
# id num dwell delay freq phase
# ..  ..    ns    us   Hz   rad
[ADC]
1 1024 312500 20 0 0

# Format of delays:
# id delay (us)
[DELAYS]
```

```
1 5000

# Sequence Shapes
[SHAPES]

shape_id 1
num_samples 120
1
0
0
97
-1
0
0
17

shape_id 2
num_samples 120
0
0
118
```

2 Specification

2.1 Overall Description

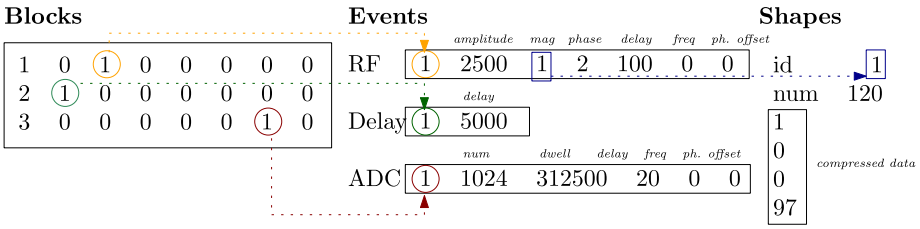

Figure 1: Visualisation of the hierarchical structure of the file format for an FID example.

The sequence description consists of a three-level hierarchical structure as demonstrated in Figure 1. This ensures the most compact representation, since repeating events (or shapes) are only declared once.

Blocks At the top level the sequence is specified by a number of blocks. Each block refers to one or more event(s) to be executed simultaneously.

Events The definition of events are dependent on the type (e.g. gradient, ADC)

but some types may refer to one or more basic shapes.

Shapes A shape is a compressed list of samples. The uncompressed samples can describe, for example, the RF pulse shape or an arbitrary gradient shape. The compression scheme is a type of run-length encoding.

2.2 Identification numbers

A key idea of the hierarchical sequence structure is the use of ID numbers to refer to objects defined one level below in the hierarchy. For example, blocks contain the ID of one or more event(s). Likewise, events may contain the ID of one or more shape(s). Restrictions are placed on these IDs to ensure consistency:

- IDs must be positive integers.
- The ID of each shape must be unique.
- The ID of events within a single class must be unique. For example, an RF event and delay event may both use an ID of 1, since the events are of a different class. An exception is the `[GRADIENTS]` and `[TRAP]` events where the union of these sets must contain unique event IDs.
- There are no restrictions on the ordering of IDs, although sequential ordering is often implemented.
- There are no restrictions on the first ID of an event class.

2.3 General

Comments are specified by a line starting with a hash.

```
# This is a comment
```

The sequence may contain general definitions indicated by the `[DEFINITIONS]` keyword. Each definition is defined on a new line with the following format

```
<key> <value>
```

This defines a list of user-specified key/value pairs.

Tag	Type	Description	Units
`<key>`	text	Name of user definition	–
`<value>`	any	Value of definition	–

Example:

```
[DEFINITIONS]
Scan_ID 237
FOV 0.150 0.150 0.005
TE 0.005
TR 0.01
Num_Blocks 3072
TotalDuration 12.96
```

It is important to note that all definitions are optional and do not affect the basic sequence execution. Precise timing is given by the low-level specification of events. The definitions section may be used for user-specific purposes, including attaching metadata or hardware-dependent parameters.

2.4 Version

The versioning scheme is ⟨major⟩.⟨minor⟩.⟨revision⟩.

```
[VERSION]
major ⟨major⟩
minor ⟨minor⟩
revision ⟨revision⟩
```

Providing the version of the standard is vital for sequence execution. If this section is not provided, the interpreter sequence may either refuse execution or assume version 1.0.0.

Example:

```
[VERSION]
major 1
minor 3
revision 1
```

2.5 Blocks

The section containing sequence blocks is declared with the `[BLOCKS]` keyword. Each subsequence line declares a single block specified by a list of six event IDs. An ID of 0 indicated no event.

```
<id> <delay> <rf> <gx> <gy> <gz> <adc> <ext>
```

Each tag represents the ID of the corresponding event.

Tag	Type	Description	Units
<id>	integer	ID of the sequence block	–
<delay>	integer	ID of the delay event	–
<rf>	integer	ID of the RF event	–
<gx>	integer	ID of the gradient event on the X channel	–
<gy>	integer	ID of the gradient event on the Y channel	–
<gz>	integer	ID of the gradient event on the Z channel	–
<adc>	integer	ID of the ADC event	–
<ext>	integer	ID of the extension table entry	–

The sequence must declare at least one block. Any non-zero number of blocks may be defined. The blocks are executed sequentially. The duration of each block is defined by the longest event comprising this block. X, Y and Z refer to physical scanner gradient channels. Block duration may be zero, e.g. for blocks containing only extension events with zero durations such as data labels.
Example:

```
[BLOCKS]
   1   0   1   0   0   2    0    0
```

The block above is the first in the sequence and contains an RF pulse with ID of 1 and a z-gradient pulse with ID of 2. The block has no delay, X gradient, Y gradient or ADC events, indicated by zero IDs.

2.6 Events

Events are defined in sections, each starting with one the following keywords: [RF], [GRADIENTS], [TRAP], [ADC] or [DELAYS]. Each event is specified on a single line and contains an ID followed by type-specific definition.

2.6.1 RF

The RF section is declared with the [RF] keyword. Following this declaration, each RF event is specified by a single line containing seven numbers.

```
<id> <amp> <mag_id> <phase_id> <delay> <freq> <phase>
```

The specifiers are

Tag	Type	Description	Units
<id>	integer	ID of the RF event	–
<amp>	float	Peak amplitude	Hz
<mag_id>	integer	Shape ID for magnitude profile	–
<phase_id>	integer	Shape ID for phase profile	–
<delay>	integer	Delay before starting the RF pulse	μs
<freq>	float	Frequency offset	Hz
<phase>	float	Phase offset	rad

Example:

```
[RF]
  1    2500  1   2  0  0.000  0.000
```

In the example above, the RF pulse has ID of 1, peak amplitude of 2500Hz. The magnitude and phase profiles are defined with the shapes of ID 1 and 2, respectively. The RF pulse does not have any frequency or phase offset or delay.

2.6.2 Gradients

Gradient events are declared in two sections. Arbitrary gradients are in a section declared with the [GRADIENTS] keyword. Each line in the section is an arbitrary gradient specified by four numbers,

```
<id> <amp> <shape_id> <delay>
```

Trapezoidal gradients are in a section declared with the [TRAP] keyword. Each line in the section is a trapezoidal gradients specified by six numbers,

```
<id> <amp> <rise> <flat> <fall> <delay>
```

The specifiers are

Tag	Type	Description	Units
<id>	integer	ID of the gradient event	–
<amp>	float	Peak amplitude	Hz/m
<shape_id>	integer	Shape ID for arbitrary gradient waveform	–
<rise>	integer	Rise time of the trapezoid	μs
<flat>	integer	Flat-top time of the trapezoid	μs
<fall>	integer	Fall time of the trapezoid	μs
<delay>	integer	Delay before starting the gradient event	μs

The gradient ID must be unique across both arbitrary and trapezoid gradients.

That is, a trapezoid gradient cannot have the same ID as an arbitrary gradient.

Example:

```
[GRADIENTS]
  1    159154.9  3  0
[TRAP]
  2       25000    30  940  30  100
  3    20066.89    10  980  10  100
```

The example above contains two gradients: an arbitrary gradient with peak amplitude of approximately 159kHz/m and shape ID 3 and two trapezoid gradients (IDs 2 and 3) with duration 1ms specified by amplitude, rise time, flat-top time and fall time. Both gradients have a delay of 100μs.

2.6.3 ADC

The ADC section is declared with the `[ADC]` keyword. Each line in the section is an ADC event specified by six numbers,

```
<id> <num> <dwell> <delay> <freq> <phase>
```

The specifiers are

Tag	Type	Description	Units
`<id>`	integer	ID of the ADC event	–
`<num>`	integer	Number of samples	–
`<dwell>`	float	The ADC dwell time	ns
`<delay>`	integer	Delay between start of block and first sample	μs
`<freq>`	float	Frequency offset of ADC receiver	Hz
`<phase>`	float	Phase offset of ADC receiver	rad

The duration of the ADC readout is given by the product of `<num>` and `<dwell>`.

Example:

```
[ADC]
  1    512  5000  0  0.000   0.000
```

The example above contains an ADC with 512 samples, and dwell time of 5000ns, and no frequency and phase offsets. The frequency and phase offset are used, for example, for RF spoiling or inplane shifting of the FOV.

2.6.4 Delays

The delay section is declared with the `[DELAYS]` keyword. Each line in the section is a delay event specificed by two numbers,

```
<id> <delay>
```

The specifiers are

Tag	Type	Description	Units
`<id>`	integer	ID of the delay event	–
`<delay>`	integer	Delay which overlaps with other events within a block and therefore defines the minimum duration of a block	μs

2.6.5 Extensions

Extensions concept allows for implementing additional features without requiring major revisions of the Pulseq format specification. Extensions can be defined in the design tool in a good hope that the particular interpreter will know how to handle it. The interpreter MUST detect unknown extensions and MAY chose to ignore them. It is recommended that the interpreter issues a warning in this case.

The EXTENSIONS section is declared with the `[EXTENSIONS]` keyword. Each line in the section is an extension table entry specified by four numbers,

```
<id> <type> <ref> <next>
```

Tag	Type	Description
`<id>`	integer	ID of the extension list entry
`<type>`	integer	ID of the type of the extension
`<ref>`	integer	ID of the extension object
`<next>`	integer	ID of the next entry in the list

Extensions form zero-terminated single-linked list. Therefore event table `[BLOCKS]` can reference a chain of extensions, which allows one to add more than one extension per block.

Extension list is followed by the actual specification of particular extensions. The specification has the following format:

```
extension <STRING_ID> <type>
```

where

Tag	Type	Description
`extension`	keyword	Keyword specifying the beginning of the particular extension specification
`<STRING_ID>`	integer	text ID of the present extension used by the interpreter to recognize the extension
`<type>`	integer	ID of the type of the particular extension, referenced by the extension list

The interpreters MUST only use the `STRING_ID` to recognize the extensions and do not count on particular value of the `<type>` parameter. The latter is only valid within the single Pulseq file and MAY change from one file to another, or different unique type ID values may be used by different Pulseq export implementations.

Example:

```
[EXTENSIONS]
1 1 1 0
2 1 2 0

extension TRIGGERS 1
1 2 1 0 2000
2 1 3 500 100
```

The example above contains a specification of the TRIGGERS extension, which is not a part of the core Pulseq format and MAY be subject to rapid changes between revisions. Current example above specifies one cardiac trigger (id=1 type=2 channel=1) and one digital output signal (id=2 type=1 channel=3) on the "external trigger" channel (Siemens-specific).

Label Extension

Starting from revision 1.3.1 both Matlab Pulseq toolbox and the Siemens interpreter sequence support LABEL extension. The extension is intended to provide a possibility for the pulse sequence to pass counters and flags over to the raw data object and in this way to communicate with the image reconstruction routines.

Two types of extensions are currently defined: LABELSET and LABELINC. The former allows one to set the counter or flag value, the latter can be used to increment the counter (not compatible with flags). The following counters are

supported at the moment: LIN, PAR, SLC, SEG, REP, AVG, SET, ECO, PHS that can take any integer values and flags NAV, REV, SMS that can be 0 or 1 corresponding to boolean 'false' and 'true' values.

At the start of the sequence all counters and flags are automatically initialized to zero. During the run-time sequence maintains current values of all counters and flags, which can be modified with LABELSET and LABELINC directives at any time in the sequence. Also zero-duration blocks containing only LABEL extension directives are allowed. The values of the counters have no immediate effect until the next ADC directive is executed. At the moment of the ADC event the values of the counters are recorded and in case of the Siemens interpreter are copied to the ADC Data Header. Prior to the pertinent ADC direction the values of the counters can take any arbitrary excursions, e.g. can be negative or exceed matrix dimensions temporarily. This has no negative effects.

In case the same block contains LABELSET, LABELINC and ADC directives the following priorities apply (independent of the order the events were submitted to the block e.g. in the Matlab toolbox): First all LABELSET directives are processed, followed by all LABELINC and only then the values of the counters are captured and passed over to the ADC directive.

2.7 Shapes

The shape section is declared with the `[SHAPES]` keyword. Each shape is then declared with a header followed by a list of samples values (one per line). The end of the shape definition is declared with a blank line.

```
Shape_ID <id>
Num_Uncompressed <num>
<sample_1>
<sample_2>
...
```

The specifiers are

Tag	Type	Description	Units
`<id>`	integer	ID of the shape	–
`<num>`	integer	Number of samples of the uncompressed shape	–
`<sample_`n`>`	integer	The n^{th} sample of the compressed shape	–

The decompressed samples must be in the normalised range [0, 1]. Since the purpose of this section is to define the basic shape of a gradient or RF pulse, the amplitude information is defined in the events section. This allows the same shape to be used with different amplitudes, without repeated definitions.

The number of points after decompressing all samples defined in a shape must equal the number declared in `<num>`.

2.7.1 Compression

Storing every sample of the shape would lead to very large sequence descriptions. Suppose a sequence contains a block RF pulse for 4ms and a sinusoidally-ramped constant gradient for 100ms. Assuming sampling times of 1μs and 10μs for the RF and gradients, respectively, 14000 samples would be required. Instead, the shapes are compressed by **encoding the derivative in a run-length compressed** format.

Example 1: A shape consisting of a ramp-up, constant and ramp-down is encoded as follows

Shape		Step 1 (derivative)		Step 2 (compression)
0.0		0.0		0
0.1		0.1		0.1
0.25		0.15		0.15
0.5		0.25		0.25
1.0		0.5		0.5
1.0		0.0		0.0
1.0	$\rightarrow$	0.0	$\rightarrow$	0.0
1.0		0.0		4
1.0		0.0		-0.25
1.0		0.0		-0.25
1.0		0.0		2
0.75		-0.25		
0.5		-0.25		
0.25		-0.25		
0.0		-0.25		

Example 2: A shape with 100 zeros values

Shape		Step 1 (derivative)		Step 2 (compression)
0.0		0.0		0
0.0	→	0.0	→	0
...		...		98
0.0		0.0		

Example 3: A shape with a constant value of 1.0 for 100 samples

Shape		Step 1 (derivative)		Step 2 (compression)
1.0		1.0		1.0
1.0	→	0.0	→	0
...		...		0
1.0		0.0		97

3 Binary files

The specification described in Section 2 can be implemented as a binary file. The same general principles apply with specific modifications outlined here. The basic structure of a binary *pulseq* file is depicted below,

0x01	p	u	l	s	e	q	0x02
version major							
version minor							
version revision							
section code							
number of events							
data							
section code							
number of events							
data							
...							

3.1 File and section codes

A binary *Pulseq* file begins with the 64 bit code `0x0170756C73657102` (the characters `pulseq` enclosed by `0x01` and `0x02`) followed by three integers describing the file version (major, minor, revision). The remaining file is made up

of multiple sections each with an integer section code followed by section-specific storage. The section codes corresponding to text file tags are

Section	Section code
[DEFINITIONS]	0xFFFFFFFF 0x00000001
[BLOCKS]	0xFFFFFFFF 0x00000002
[RF]	0xFFFFFFFF 0x00000003
[GRADIENTS]	0xFFFFFFFF 0x00000004
[TRAP]	0xFFFFFFFF 0x00000005
[ADC]	0xFFFFFFFF 0x00000006
[DELAYS]	0xFFFFFFFF 0x00000007
[SHAPES]	0xFFFFFFFF 0x00000008

3.2 Section details

The number of entries in each section is written as an integer directly after the section code.

Unlike the text format, the [DEFINITIONS] section is declared with the section code followed by *four* fields per definition

```
<key> <type> <num> <value_1> <value_2>...
```

The <key> is a null-terminated ASCII encoded string. <type> is an integer code either 1 to denote integer or 2 to denote floating-point. <num> is an integer defining the number of values in this definition and <value_n> are the values of given type. The next definition immediately follows otherwise a new section code follows or it is the end of the file.

The [SHAPES] section is declared with the appropriate section code followed by

```
<id> <num_uncompressed> <num_compressed> <sample_1> <sample_2>...
```

The <id> is an integer shape ID, <num_uncompressed> is the number of samples after decompression, <num_compressed> is the number of compressed samples for this shape. Following are exactly the specified number of compressed samples stored as floating-point numbers. The next shape begins immediately after the last otherwise a new section code is declared (or end-of-file) to indicate the end of the [SHAPES] section.

The remaining sections are declared exactly as in Section 2 with storage defined by the value type. In general, **integer** types are stored as 64-bit unsigned

integers while **floating-point** numbers are in the 64-bit IEEE 754 floating-point format. Exceptions are `<id>` values, which are 32-bit integers; and `<type>` and `<num>` values of the `[DEFINITIONS]` entries, which are 8-bit integers.

4 Source code

This specification is distributed with source code for reading and writing the sequences file format described here. MATLAB code is provided for detailed sequence generation, visualisation, as well as reading and writing sequence files. A C++ class and example program is also provided for reading sequence files. Detailed documentation and latest updates of this code are available here: http://pulseq.github.io/.

5 Examples

5.1 Free induction decay

```
# Pulseq sequence file
# Created by MATLAB mr toolbox

[VERSION]
major 1
minor 3
revision 1

[DEFINITIONS]
Name fid

# Format of blocks:
#  #  D RF  GX  GY  GZ ADC EXT
[BLOCKS]
1  0  1   0   0   0  0  0
2  1  0   0   0   0  0  0
3  0  0   0   0   0  1  0

# Format of RF events:
# id amplitude mag_id phase_id delay freq phase
# ..        Hz   ....     ....    us   Hz   rad
[RF]
1         2500 1 2 100 0 0

# Format of ADC events:
# id num dwell delay freq phase
# ..  ..    ns    us   Hz   rad
[ADC]
1 1024 312500 20 0 0

# Format of delays:
# id delay (us)
[DELAYS]
1 5000

# Sequence Shapes
[SHAPES]

shape_id 1
num_samples 120
1
0
0
97
-1
0
0
17

shape_id 2
num_samples 120
```

```
0
0
118
```

5.2 Point-resolved spectroscopy (PRESS)

```
# Sequence Blocks
# Created by JEMRIS 2.8
# WARNING: this file relies on an old Pulse revision and cannot
#          be loaded by the current software.
#          It is provided here purely for illustrative purposes

[DEFINITIONS]
Scan_ID 3
Num_Blocks 8

# Format of blocks:
##  D RF  GX  GY  GZ ADC
[BLOCKS]
1  0  1   0   1   0  0
2  0  0   0   2   0  0
3  1  0   0   0   0  0
4  0  2   3   0   0  0
5  2  0   0   0   0  0
6  0  2   0   0   4  0
7  3  0   0   0   0  0
8  0  0   0   0   0  1

# Format of RF events:
# id amplitude mag_id phase_id freq phase
# ..        Hz   ....     ....   Hz   rad
[RF]
1   246.863892 1 2 0 0
2   493.727784 1 2 0 0

# Format of trapezoid gradients:
# id amplitude rise flat fall
# ..      Hz/m   us   us   us
[TRAP]
 1      200106.3  30 3940  30
 2     -201629.9  30 1940  30
 3      200106.3  30 3940  30
 4      200106.3  30 3940  30

# Format of ADC events:
# id num dwell delay freq phase
# ..  ..    ns    us   Hz   rad
[ADC]
 1 256  25000   0 0 0

# Format of delays:
# id delay (us)
[DELAYS]
1 19000
```

```
2 46000
3 44800

# Sequence Shapes

[SHAPES]

shape_id 1
num_samples 4000
3.856279e-11
3.588134e-08
3.588134e-08
8
2.323701e-07
2.323701e-07
8
6.188511e-07
6.188511e-07
8
1.197773e-06
1.197773e-06
8
1.970951e-06
1.970951e-06
8
2.939553e-06
2.939553e-06
8
4.104086e-06
4.104086e-06
8
5.464388e-06
5.464388e-06
8
<<truncated>>

shape_id 2
num_samples 4000
0.5
0
0
997
-0.5
0
0
1997
0.5
0
0
997
```

5.3 Gradient echo

```
# Sequence Blocks
```

```
# Created by JEMRIS 2.8
# WARNING: this file relies on an old Pulse revision and cannot
#          be loaded by the current software.
#          It is provided here purely for illustrative purposes

[DEFINITIONS]
Scan_ID 2
Num_Blocks 160

# Format of blocks:
# #  D RF  GX  GY  GZ ADC
[BLOCKS]
  1  0  1   0   0   1  0
  2  0  0   2   3   4  0
  3  1  0   0   0   0  0
  4  0  0   5   0   0  1
  5  2  0   0   0   0  0
  6  0  1   0   0   1  0
  7  0  0   2   6   4  0
  8  1  0   0   0   0  0
  9  0  0   5   0   0  1
 10  2  0   0   0   0  0
 11  0  1   0   0   1  0
 12  0  0   2   7   4  0
<<truncated>>

# Format of RF events:
# id amplitude mag_id phase_id freq phase
# ..        Hz   ....     ....   Hz   rad
[RF]
1   276.908986 1 2 0 0

# Format of trapezoid gradients:
# id amplitude rise flat fall
# ..      Hz/m   us   us   us
[TRAP]
 1      159154.9  30 4000  30
 2     -26797.66  10 2980  10
 3     -26755.85  10 2980  10
 4     -107616.5  20 2960  20
 5         25000  10 6400  10
 6     -25083.61  10 2980  10
 7     -23411.37  10 2980  10
<<truncated>>

# Format of ADC events:
# id num dwell delay freq phase
# ..  ..    ns    us   Hz   rad
[ADC]
 1  32 200000  10 0 0

# Format of delays:
# id delay (us)
[DELAYS]
1 1760
2 184770
```

```
# Sequence Shapes

[SHAPES]

shape_id 1
num_samples 4000
0.0002500625
0.0005026691
0.0005026691
8
0.0005072244
0.0005072244
8
0.0005113487
0.0005113487
8
0.0005150341
0.0005150341
8
0.0005182732
0.0005182732
<<truncated>>

shape_id 2
num_samples 4000
0.5
0
0
997
−0.5
0
0
1997
0.5
0
0
997
```