Open file format for MR sequences

Version 1.5.3(DRAFT)

Maxim Zaitsev
Stefan Kroboth
Kelvin Layton

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.
1.4.0	7 Jul 2021	Substantial revisions of the file format: added required definitions; we now explicitly specify sampling and raster alignment conventions; flexible shape timing for arbitrary gradients and RF pulses; explicit block duration; removed [DELAYS]; added uncompressed shape option; added signature section.
1.4.1	16 Feb 2023	New flags to the LABEL extension added.
1.5.0	14 Feb 2025	Substantial format revision, including: RF pulses define RF center and intended use fields; RF and ADC objects declare an additional frequency offset in units of PPM; ADC objects optionally declare phase offset per sample as a vector; Extension "Soft Delay" is introduced.
1.5.1	14 Aug 2025	Minor format revision, new extensions and bugfixes: New concept of *required extensions* wich must be supported by the interpeter; Introduce new Rotation Extension; Introduce new RF Shimming Extension.
1.5.2	28 Apr 2026	Minor format update for the binary format and bugfixes. TODO: update figures and examples.

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 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 constructs specific to a particular hardware manufacturer.
4. **Compact:** The sequence should avoid 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', 'Tesla' or 'DAC units'.

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 5
revision 1

[DEFINITIONS]
AdcRasterTime 1e-07
BlockDurationRaster 1e-05
GradientRasterTime 1e-05
Name fid
RadiofrequencyRasterTime 1e-06

# Format of blocks:
# NUM DUR RF  GX  GY  GZ  ADC  EXT
[BLOCKS]
1  42   1   0   0   0  0  0
2 500   0   0   0   0  0  0
3 10244   0   0   0   0  1  0

# Format of RF events:
# id ampl. mag_id phase_id time_shape_id center delay freqPPM phasePPM freq phase use
# ..   Hz      ..       ..            ..     us    us     ppm  rad/MHz   Hz
rad  ..
# Field 'use' is the initial of:
#   excitation refocusing inversion saturation preparation other undefined
[RF]
1      833.333 1 2 0 150 100 0 0 0 0 e

# Format of ADC events:
# id num dwell delay freqPPM phasePPM freq phase phase_id
# ..  ..    ns    us     ppm  rad/MHz   Hz   rad       ..
[ADC]
1 1024 100000 20 0 0 0 0 0

# Sequence Shapes
[SHAPES]

shape_id 1
num_samples 300
1
0
0
297

shape_id 2
num_samples 300
0
0
298


[SIGNATURE]
# This is the hash of the Pulseq file, calculated right before the [SIGNATURE]
# section was added. It can be reproduced/verified with md5sum if the file
# trimmed to the position right above [SIGNATURE]. The new line character
# preceding [SIGNATURE] BELONGS to the signature (and needs to be sripped away
# for recalculating/verification)
Type md5
Hash 217d8a34f69ccfab42d754cefa333d7f
```

2 Specification

2.1 Overall Description

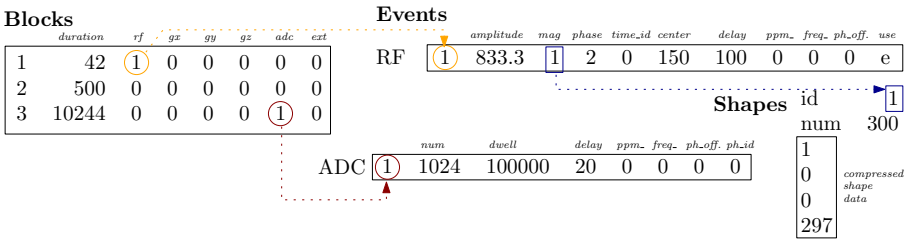

Figure 1: Visualisation of the hierarchical structure of the file format for an FID example.

The sequence description consists of a three-level hierarchical structure as demonstrated in Figure 1. This ensures a 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.

Comments are specified by a line starting with a hash.

```
# This is a comment
```

The numeric values are declared exactly as described below 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. Shapes are stored as 32-bit single-precision floats.

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.
- The ID of 0 for a time shape corresponds to the "default", meaning an incremental time sampling with the corresponding default time raster.
- 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 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. It is recommended to reject Pulseq files without `[VERSION]` section.

Example:

```
[VERSION]
major 1
minor 5
revision 3
```

2.4 Signature

The Pulseq file may end with an optional section `[SIGNATURE]` as in the example below:

```
[SIGNATURE]
# This is the hash of the Pulseq file, calculated right before
# the [SIGNATURE] section was appended to the file
Type md5
Hash 237b85c2aa46ba98076359cde263d6ee
```

The signature is generated by hashing the readily exported file with a corresponding algorithm (in this example md5sum) and appending the `[SIGNATURE]` section. The new line character preceding the keyword `[SIGNATURE]` is part of the signature and needs to be stripped away for the signature verification. The `[SIGNATURE]` section must contain two following fields along with any additional information in the format identical to that of `[DEFINITIONS]` (see below).

Tag	Type	Description
`Type`	string	ID of the hashing algorithm (e.g. md5, sha1 or sha256)
`Hash`	string	Hash value produced by the hashing function in the hexadecimal format

2.5 Definitions

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	–

The `<value>` field is separated by a single space or tab character from `<key>` and is terminated at the end of the line. Any (additional) white space at the beginning and at the end of `<value>` is ignored. The `<key>` field cannot contain any white space caracters.

Example:

```
[DEFINITIONS]
AdcRasterTime 1e-07
BlockDurationRaster 1e-05
GradientRasterTime 1e-05
RadiofrequencyRasterTime 1e-06
FOV 0.256 0.256 0.004
Name epi
TE 0.005
TR 0.01
TotalDuration 0.04268
```

Starting from the Pulseq format revision 1.4.0, the following definitions are reserved, some of which are optional, whereas the definitions labeled as **required** are compulsory.

Definition	Description	Status
`GradientRasterTime`	Default raster time (dwell time) of the shaped gradient events, specified in seconds	required
`RadiofrequencyRasterTime`	Default raster time (dwell time) of the radio-frequency pulse shapes, specified in seconds	required
`AdcRasterTime`	The value defining the alignment of the ADC dwell times; ADC dwell time must be integer multiple of the specified AdcRasterTime; AdcRasterTime is specified in seconds	required
`BlockDurationRaster`	The value defining the alignment of the block durations, specified in seconds; the physical block duration must be integer multiple of the specified BlockDurationRaster; Block duration in the `[BLOCKS]` section are specified in the units of BlockDurationRaster	required
`Name`	Human-readable name of the sequence	optional
`FOV`	Field of view specified in meters in x, y and z directions given as a space-separated list of numbers	optional
`TotalDuration`	Total duration of the sequence is seconds	optional

It is important to note that all definitions that are not **required** 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 arbitrary user-specific purposes, including attaching metadata or hardware-dependent parameters. Some Pulseq file interpreters use optional Definitons to make decisions about the sequence execution or aid user parameter choises on the scanner.

2.6 Time raster, temporal alignment and shape sampling conventions

Accurate timing is essential for NMR and MRI experiments. To guarantee unambiguous time definition Pulseq operates at discrete time raster. Depending on the particular hardware implementation details different electronic components may have different clock resolutions and correspondingly different raster times.

For understanding the time raster alignment it is important to differentiate between the `edges` and `centers` of the time raster steps. All blocks and events are aligned to the corresponding `edges` of the time raster. Block duration is required to be multiple of `BlockDurationRaster`, therefore each block can only start and end at the `edges` of the block duration raster. Gradient events may only start and end at time points which are multiple of `GradientRasterTime`, that is at the edges of `GradientRasterTime`; for trapezoid events, ramp times and plateau duration may also only be equal to multiples of `GradientRasterTime`. In contrast to that, any default sampling points (e.g. sampling points of arbitrary-shape gradients) are aligned to the `centers` of the corresponding raster steps. The same convention applies to the default RF shape sampling and ADC sampling points. This means that the time of the n-th point on the RF pulse shape, gradient shape or ADC sampling point can be calculated as $t_n = t_{start} + \Delta t(0.5 + n)$. We will eventually include some figures and diagrams to this section to explain this important concept, but for now please refer to the slide set at: https://github.com/pulseq/pulseq/blob/master/doc/pulseq_shapes_and_times.pdf.

2.7 Blocks

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

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

Each non-zero value among the six tags represents the ID of the corresponding event.

Tag	Type	Description
<id>	integer	ID of the sequence block
<duration>	integer	duration of the current block in units of BlockDurationRaster
<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 explicitly by the second field of the block description. No event comprising this block is allowed to last longer than the specified block duration. The interpreters must to throw an error, should they detect such condition. 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. A block with nonzero duration containing no nonzero event IDs is a delay block. Interpreters must gracefully ignore delay blocks of 0 duration.

Example:

```
[BLOCKS]
   1   319   1   0   0   2    0    0
```

The block above is the first in the sequence which has a duration of 319*BlockDurationRaster and contains an RF pulse with ID of 1 and a z-gradient pulse with ID of 2. The block has no X gradient, Y gradient or ADC events, indicated by zero IDs and also no extension is specified.

2.8 Events

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

2.8.1 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> <time_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	–
<time_id>	integer	Shape ID for the time sampling points, specified in the units of RadiofrequencyRasterTime; 0 means default time raster	–
	float	Time point relative to the beginning of the RF shape at which the effective rotation takes place	μs
<delay>	integer	Delay before starting the RF pulse	μs
<freq_ppm>	float	Frequency offset relative to the main system's frequency	ppm
<phase_ppm>	float	Phase offset proportional to the main system's frequency	rad/MHz
<freq_off>	float	Frequency offset in absolute units	Hz
<phase_off>	float	Phase offset	rad
<use>	char	Intended use fo the RF pulse; it is the initial of one of: excitation, refocusing, inversion, saturation, preparation, other or undefined	-

Please note that for the calculation of the final frequency and phase offsets the contributions from <freq_ppm> and <freq_off> (or <phase_ppm> and <phase_off>) are added together. Prior to the addition <freq_ppm> and <phase_ppm> are weighted with the current system frequency of the active nucleus, expressed in units of MHz.

Example:

```
[RF]
  1    550  1   2  0  1000  100  0.000  0.0000  0.000  0.000  e
  2    130  3   4  0  4000  100 -3.350  0.0841  0.000  0.000  s
```

In the example above, the RF pulse ID of 1 is the excitation pulse with the peak amplitude of 550Hz and a delay of 100 μs. The magnitude and phase profiles are defined with the shapes of ID 1 and 2, respectively, using the default time axis sampling with a dwell time of RadiofrequencyRasterTime. The effective center of this RF pulse is located 1000 μs away from the start of the RF shape

and the pulse does not have any frequency or phase offset. The RF pulse with ID of 2 is a saturation pulse with the peak amplitude of 130Hz and also a delay of 100 μs, that uses shape IDs 3 and 4 for the magnitude and phase, respectively. It is a fat saturation pulse as can be seenfrom the `<freq_ppm>` of -3.35 ppm. The corresponding `<phase_ppm>` value of 0.0841 rad/MHz is introduced to compensate for the phase due to the frequency offset, leading to the phase at the RF pulse center of 0.

2.8.2 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> <first> <last> <shape_id> <time_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
`<first>`	float	The amplitude of the gradient at the start of the gradient. Only relevant for the arbitrary gradients (shaped gradients and extended trapezoids)	Hz/m
`<last>`	float	The amplitude of the gradient at the end of the gradient. Only relevant for the arbitrary gradients	Hz/m
`<shape_id>`	integer	Shape ID for arbitrary gradient waveform	–
`<time_id>`	integer	Shape ID for the time sampling points, specified in the units of `GradientRasterTime`; 0 means default time raster; -1 means 1/2 of the default time raster (gradient oversampling case)	–
`<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    790127    0       -550073 6   0 980
  2    793249    0        574045 7  -1 980
  3   -550073   -550073   0       8   9 0
  4    574045    574045   0       8   9 0
[TRAP]
  5        25000     30   940   30   100
  6     20066.89     10   980   10   100
```

The example above contains six gradients: two arbitrary gradients with peak amplitudes of approximately 790kHz/m (in fact this is a center-out spiral readout). The first one has shape ID 6 and time shape ID of 0, which means the shape ID 6 contains gradient samples according to the convention stated in subsection 2.6, e.g. sampled at the centers of the gradient raster cells. The gradient starts with a zero amplitude but ends at the amplitude of -550kHz/m. The second shaped gradient is similar, but has a time shape ID of -1, meaning that the shape ID 7 is defined with the oversampling (e.g. it contains N-1 additional points between the nominal samples sampled at the gradient raster cell edges). Normally yo don't combine oversampled and non-oversampled gradients like that, here it is given as a toy example. The gradient with ID 2 ends at the amplitude of 574kHz/m. Because both these gradients end at non-zero values,

they need to be aligned to the end of the block, which is achieved in this example by the start delay of 980 μs. They also require additional 'ramp down' gradients in the next block, which realized in this example by gradient objects with IDs 3 and 4. Here gradient with ID 3 starts at -550kHz/m and ends at 0 and has a shape described by the shape ID of 8 and time ID of 9. Gradients that start at non-zero values must be aligned to the beginning of the block, meaning that the start delay of such gradients must be 0. Gradient with ID 4 is the other 'ramp down' gradient, defined analogously. Note that both these gradients share the shape and time IDs. The example als defines two trapezoid gradients (IDs 5 and 6) with duration 1ms specified by amplitude, rise time, flat-top time and fall time. Both trapeyoid gradients have a delay of 100μs.

2.8.3 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_ppm> <phase_ppm> <freq> <phase> <phase_shape_id>
```

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_ppm>`	float	Frequency offset of ADC receiver relative to the system frequency	ppm
`<phase_ppm>`	float	Phase offset of ADC receiver that is proportional to the system frequency	rad/MHz
`<freq>`	float	Frequency offset of ADC receiver	Hz
`<phase>`	float	Phase offset of ADC receiver	rad
`<phase_shape_id>`	integer	Shape ID of the phase modulation, 0 for no modulation	–

The duration of the ADC readout is given by the product of `<num>` and `<dwell>` The final phase and frequency coffsets are calculated as sums of the corresponding frequency-dependent and frequency-independent contributions. Prior to the addition `<freq_ppm>` and `<phase_ppm>` are weighted with the current system frequency of the active nucleus, expressed in units of MHz.

Example:

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

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 in-plane shifting of the FOV. Offsets with the 'ppm' suffix are usable for fat navigators.

2.8.4 Extensions

Extensions concept allows for implementing additional pulse sequence format 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. Since the format revision v1.5.1 there is a concept of *required extensions*, that is the extensions that are critical for the interpretability of the particular pulse sequence. All extensions listed under the definition `RequiredExtensions` must be supported by the interpreter. If any of the declared required extensions are unknown to the interpreter, it MUST generate an error and MUST not attempt executing the sequence. The definition `RequiredExtensions` contains a space-separated list of extension's `STRING_ID(s)`.

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, ACQ, SLC, SEG, REP, AVG, SET, ECO, PHS that can take any integer values and flags NAV, REV, SMS, OFF, NOISE, REF, IMA, PMC, NOPOS, NOROT, NOSLC that can be 0 or 1 corresponding to boolean 'false' and 'true' values. There is on three-state flag ONCE, described below. There is also a special label TRID used by some interpreters to recognize repeating parts of the sequence.

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.

New experimental labels have been introduced in revision 1.4.1. These include `PMC`, `NOROT`, `NOPOS`, `NOSLC` and `ONCE` flags (the latter explained further below).

The meaning of the flags `PMC`, `NOROT`, `NOPOS` and `NOSLC` is as follows:

ID	Type	Description
`PMC`	flag	Defines whether prospective motion correction (PMC) may be applied to the current block
`NOROT`	flag	Instructs the interpreter to ignore rotation components of the FOV positioning applied through the UI on the scanner or by the PMC processing (EXPERIMENTAL)
`NOPOS`	flag	Instructs the interpreter to ignore translation components of the FOV positioning applied through the UI on the scanner or by the PMC processing (EXPERIMENTAL)
`NOSLC`	flag	Instructs the interpreter to ignore FOV / gradient scaling parameters applied through the UI on the scanner (EXPERIMENTAL)

Additionally, there is a three-state flag `ONCE`, that is intended to control the execution flow for sequences with repetitions. It allows to implement dummy scans, warm-up sub-sequences, calibration scans or closing-up sub-sequences. The flag instructs the interpreter to alter the sequence when executing multiple repeats as follows: blocks with ONCE=0 are executed on every repetition; ONCE=1 marks the blocks that are only executed in the first repetition; blocks with ONCE=2 (or any other number) are only executed in the last repetition. When number of repetitions is equal to 1 no changes are applied.

Soft Delay Extension

The *soft delay* extension has been introduced in revision 1.5.0. It provides a posibility for an exported Pulseq file to redefine some aspects of the sequence tming at the run time. Label extension can be used in conjunction with pure delay blocks (blocks of non-zero duration containing no RF, gradient or ADC events). For such blocks Soft Delay extension is able to rewrite the duration of the block at run time based on the rule provided by the user. Typical use cases include adjusting the cardiac delay based on the current subject pulse rate, adjusting delays in composite pulses for MR spectroscopy or setting TE and TR for readily exported sequences.

Example:

```
# Extension specification for soft delays:
# id num offset factor hint
# ..  ..     us     ..   ..
extension DELAYS 2
1 0 -7840    2 TE
2 0 -9320    2 TE
3 0  120000 -1 TE
4 1 -126760 11 TR
5 2  0       1 TD
```

In the example above, line 1 defines a Soft Delay with numeric ID 0 and text ID 'TE', with the actual block durarion calculated as $D_1 = TE/2 - 7840\mu s$. Line 2 defines another instance of the Soft Delay with the same numeric and text IDs, but where the delay duration is calculated as $D_2 = TE/2 - 9320\mu s$. As one can guess, these two delays facilitate the TE setting in a spin-echo sequence. A somewhat special case is the Soft Delay define in line 3: it defines an additional delay which makes the execution time of the inner sequence loop independent of TE by introducing a delay with a duration $D_3 = 120000\mu s - TE$. parameter values (TE in this example) leading to negative delays are not allowed, leading to the automatic definition of the minimal and maximum allowable TE values in this example. Soft delay in line 4 defines the TR delay $D_4 = TR/11 - 126760\mu s$, from which it is easy to conclude, that it was a sequence with 11 slices. Line 5 defines the most trivial Soft Delay, where the value entered in the scanner's UI for the parameter marked TD is simply interpreted as the new block duration.

Rotation Extension

The *rotation extension* has been introduced in revision 1.5.1. It allows the pulse sequence developer to reuse the gradient object, expecially in case of arbitrary shapes, e.g. as needed for multi-shot spiral seuqnces, and specify a rotation operation that is applied to the entire block as a whole. Only one *rotation extension* object can be specified per block. Each rotation operation is described as a normalized rotation quaternion as follows:

Example:

```
# Extension specification for rotations:
# id RotQuat0 RotQuatX RotQuatY RotQuatZ
extension ROTATIONS 1
1  0.99171 0 0 -0.128498
2  1 0 0 0
```

Rotation quaternions are represented by their real part `RotQuat0`, which corresponds to the cosine of the halved rotation angle, and the three imaginary components `RotQuatX`, `RotQuatY`, and `RotQuatZ`, which together describe the axis of rotation, scaled by the sine of the halved rotation angle. In the example above, line 1 defines a rotation by 14.77° about the negative Z axis, whereas line 2 specifies a unity transform (zero rotation). Please consult functions `mr.aux.quat.toRotMat()` and `mr.aux.quat.fromRotMat()` for more information on transformations between the rotation matrices and quaternions. A great care should be taken when implementing rotation support in interpreters because of the misinterpretation possibilities (e.g. due to inconsistencies between raw-major vs. column-major matrix math libraries). Therefore we recommend to compare measurement results generated by the sequences using the *rotation extension* with the sequence relying on explicit rotations using `mr.rotate()` or `mr.rotate3D()` functions.

RF Shimming Extension

The *RF shimming* extension has been introduced in revision 1.5.1. Only one *RF shimming* extension object can be specified per block. RF shimming parameters only act on the RF pulse in the current block. This means RF shimming parameters specified for a block containing no RF have no effect and can be ignored. RF shimming settings are highly experimental and more experience is needed to possibly allow future revisions of the extension. For instance RF shimming parameters start with explicitly specifying for how many RF transmit channels these are designed. If this number does not correspond to the number of channels available in the hardware, the interpreter should probably ignore this setting and may generate an error or warning.

Example:

```
# Extension specification for RF shimming:
# id num_chan magn_c1 phase_c1 magn_c2 phase_c2 ...
extension RF_SHIMS 4
1 2 1 0 1 1.5708
2 2 1 0 1 3.14159
3 2 0.7 0 1 0
```

In the example above, line 1 defines a 2to-channel RF shimming with both channels sharing the nominal amplitude but a phase shift of 90° is applied to the second channel. Line 2 applies a phase shift of 180° to the second channel, whereas in line 3 all phases are at 0 but the amplitude of channel 1 is scaled down by a factor of 0.7.

2.9 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. Pulseq export software may opt for storing **uncompressed** shapes, e.g. to avoid rounding errors during compression/decompression or to save space for shapes that cannot be effectively compressed. The reading routines must interpret the shape as **uncompressed** if the number of samples in the stored shape is equal to `num_samples`. Therefore the saving routine **must** save the shape in uncompressed format if the result of compression has the same length as the original shape.

```
shape_id <id>
num_samples <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`>`	float	The n^{th} sample of the compressed shape	–

When used as amplitude shapes for gradient or RF objects, the decompressed samples must be in the normalised range of [-1, 1] (e.g. the absolute value of the shape must be normalized to the range of [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** be equal the number declared in `<num_samples>`.

2.9.1 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	$\rightarrow$	0.0	$\rightarrow$	0
$\cdots$		$\cdots$		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	$\rightarrow$	0.0	$\rightarrow$	0
...		...		0
1.0		0.0		97

3 Binary files

The specification described in Section 2 can also be implemented as a binary file. The binary stream starts with a fixed 8-byte file header followed by version numbers and then a list of sections.

0x01	p	u	l	s	e	q	0x02
version major (int64)							
version minor (int64)							
version revision (int64)							
section code (int64)							
section-specific payload							
section code (int64)							
section-specific payload							
...							

Unless specified otherwise, integer values are little-endian and stored as `int64` or `int32`, and floating-point values are `float64`.

3.1 Binary numeric conventions

The current binary implementation uses a unified representation for time-like quantities:

- Generic time values are stored as `int64` picoseconds (ps). This applies to RF center/delay, arbitrary-gradient delay, trapezoid rise/flat/fall/delay, ADC dwell/delay, trigger delay/duration, and soft-delay offset.
- Block duration is a special case and is stored as `int64` in units of `BlockDurationRaster` (not in ps).
- Shape payload samples in binary files are currently written as `float32` compressed values.

3.2 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]` (legacy, deprecated)	`0xFFFFFFFF 0x00000007`
`[SHAPES]`	`0xFFFFFFFF 0x00000008`
`[EXTENSIONS]`	`0xFFFFFFFF 0x00000009`
`extension TRIGGERS` payload	`0xFFFFFFFF 0x0000000A`
`extension LABELSET` payload	`0xFFFFFFFF 0x0000000B`
`extension LABELINC` payload	`0xFFFFFFFF 0x0000000C`
`extension DELAYS` payload (soft delay)	`0xFFFFFFFF 0x0000000D`
`extension RF\_SHIMS` payload	`0xFFFFFFFF 0x0000000E`
`extension ROTATIONS` payload	`0xFFFFFFFF 0x0000000F`
`[SIGNATURE]` binary payload	`0xFFFFFFFF 0x00FFFFFF`

3.3 Section payload formats

Each section starts with its `section code (int64)`, see the table above. The payload formats are listed below.

Section	Binary payload format
[DEFINITIONS]	<num_defs:int64> then for each definition: <key_length:int32> followed by <char[]> <count:int32> <type:char> where type is c, i or f, followed by <value[count]> as char/int32/float64.
[BLOCKS]	<num_blocks:int64> then per block: <duration_raster:int64> <rf:int32> <gx:int32> <gy:int32> <gz:int32> <adc:int32> <ext:int32>.
[RF]	<num:int64> then per event: <id:int32> <amp:float64> <mag_id:int32> <phase_id:int32> <time_id:int32> <center_ps:int64> <delay_ps:int64> <freq_ppm:float64> <phase_ppm:float64> <freq_off:float64> <phase_off:float64> <use:char>
[GRADIENTS]	<num:int64> then per event: <id:int32> <amp:float64> <first:float64> <last:float64> <shape_id:int32> <time_id:int32> <delay_ps:int64>
[TRAP]	<num:int64> then per event: <id:int32> <amp:float64> <rise_ps:int64> <flat_ps:int64> <fall_ps:int64> <delay_ps:int64>
[ADC]	<num:int64> then per event: <id:int32> <num_samples:int64> <dwell_ps:int64> <delay_ps:int64> <freq_ppm:float64> <phase_ppm:float64> <freq_off:float64> <phase_off:float64> <phase_shape_id:int32>
[DELAYS] (legacy)	<num:int64> then repeated <id:int32> <delay_ps:int64>. Current readers ignore this section for compatibility with old files.
[SHAPES]	<num_shapes:int64> then per shape: <id:int32> <num_uncompressed:int64> <num_compressed:int64> <compressed_data:float32[num_compressed]>
[EXTENSIONS]	<num:int64> then per entry: <id:int32> <type_id:int32> <ref:int32> <next:int32>.

See the next page for the continuation of the table.

Section	Binary payload format
`extension TRIGGERS` payload	`<ext_type_id:int32> <num:int64>` then per trigger: `<id:int32> <type:int32> <channel:int32> <delay_ps:int64> <duration_ps:int64>`
`extension LABELSET` payload	`<ext_type_id:int32> <num:int64>` then per object: `<id:int32> <value:int32> <label_index:int32>`
`extension LABELINC` payload	`<ext_type_id:int32> <num:int64>` then per object: `<id:int32> <value:int32> <label_index:int32>`
`extension DELAYS` payload	`<ext_type_id:int32> <num:int64>` then per soft delay: `<id:int32> <num_id:int32> <offset_ps:int64> <factor:float64> <hint_len:int32> <hint:char[hint_len]>`
`extension RF\_SHIMS` payload	`<ext_type_id:int32> <num:int64>` then per shim object: `<id:int32> <num_chan:int32> <mag_phase_pairs:float64[2*num_chan]>`
`extension ROTATIONS` payload	`<ext_type_id:int32> <num:int64>` then per rotation: `<id:int32> <q0:float64> <qx:float64> <qy:float64> <qz:float64>`
`[SIGNATURE]` binary payload	`<type_len:int32>` `<type:char[type_len]>` `<hash_len:int32>` `<hash:uint8[hash_len]>` `<orig_size:int64>` where `orig_size` is the original file length before appending the signature trailer. It is also the position in the final file, where the `[SIGNATURE]` section starts.

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 5
revision 1

[DEFINITIONS]
AdcRasterTime 1e-07
BlockDurationRaster 1e-05
GradientRasterTime 1e-05
Name fid
RadiofrequencyRasterTime 1e-06

# Format of blocks:
# NUM DUR RF  GX  GY  GZ  ADC  EXT
[BLOCKS]
1  42   1   0   0   0  0  0
2 500   0   0   0   0  0  0
3 10244   0   0   0   0  1  0

# Format of RF events:
# id ampl. mag_id phase_id time_shape_id center delay freqPPM phasePPM freq phase use
# ..   Hz      ..       ..            ..     us    us     ppm  rad/MHz   Hz
rad  ..
# Field 'use' is the initial of:
#   excitation refocusing inversion saturation preparation other undefined
[RF]
1      833.333 1 2 0 150 100 0 0 0 0 e

# Format of ADC events:
# id num dwell delay freqPPM phasePPM freq phase phase_id
# ..  ..    ns    us     ppm  rad/MHz   Hz   rad       ..
[ADC]
1 1024 100000 20 0 0 0 0 0

# Sequence Shapes
[SHAPES]

shape_id 1
num_samples 300
1
0
0
297

shape_id 2
num_samples 300
0
0
298
```

```
[SIGNATURE]
# This is the hash of the Pulseq file, calculated right before the [SIGNATURE]
# section was added. It can be reproduced/verified with md5sum if the file
# trimmed to the position right above [SIGNATURE]. The new line character
# preceding [SIGNATURE] BELONGS to the signature (and needs to be sripped away
# for recalculating/verification)
Type md5
Hash 217d8a34f69ccfab42d754cefa333d7f
```

5.2 Gradient echo

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

[VERSION]
major 1
minor 5
revision 1

[DEFINITIONS]
AdcRasterTime 1e-07
BlockDurationRaster 1e-05
FOV 0.256 0.256 0.005
GradientRasterTime 1e-05
Name gre
RadiofrequencyRasterTime 1e-06

# Format of blocks:
# NUM DUR RF  GX  GY  GZ  ADC  EXT
[BLOCKS]
  1 138   1   0   0   1   0   0
  2 200   0   2   3   4   0   0
  3 210   0   0   0   0   0   0
  4 642   0   5   0   0   1   0
  5 1010   0   0   0   0   0   0
  6 138   1   0   0   1   0   0
  7 200   0   2   6   4   0   0
  8 210   0   0   0   0   0   0
  9 642   0   5   0   0   1   0
 10 1010   0   0   0   0   0   0
 11 138   1   0   0   1   0   0
 12 200   0   2   7   4   0   0
 13 210   0   0   0   0   0   0
 14 642   0   5   0   0   1   0
 15 1010   0   0   0   0   0   0
 16 138   1   0   0   1   0   0
 17 200   0   2   8   4   0   0
 18 210   0   0   0   0   0   0
 19 642   0   5   0   0   1   0
 20 1010   0   0   0   0   0   0
 21 138   1   0   0   1   0   0
 22 200   0   2   9   4   0   0
 23 210   0   0   0   0   0   0
 24 642   0   5   0   0   1   0
 25 1010   0   0   0   0   0   0
 26 138   1   0   0   1   0   0
```

```
27 200    0    2   10    4   0   0
28 210    0    0    0    0   0   0
29 642    0    5    0    0   1   0
30 1010    0    0    0    0   0   0
31 138    1    0    0    1   0   0
32 200    0    2   11    4   0   0
33 210    0    0    0    0   0   0
34 642    0    5    0    0   1   0
35 1010    0    0    0    0   0   0
36 138    1    0    0    1   0   0
37 200    0    2   12    4   0   0
38 210    0    0    0    0   0   0
39 642    0    5    0    0   1   0
40 1010    0    0    0    0   0   0
41 138    1    0    0    1   0   0
42 200    0    2   13    4   0   0
43 210    0    0    0    0   0   0
44 642    0    5    0    0   1   0
45 1010    0    0    0    0   0   0
46 138    1    0    0    1   0   0
47 200    0    2   14    4   0   0
48 210    0    0    0    0   0   0
49 642    0    5    0    0   1   0
50 1010    0    0    0    0   0   0
51 138    1    0    0    1   0   0
52 200    0    2   15    4   0   0
53 210    0    0    0    0   0   0
54 642    0    5    0    0   1   0
55 1010    0    0    0    0   0   0
56 138    1    0    0    1   0   0
57 200    0    2   16    4   0   0
58 210    0    0    0    0   0   0
59 642    0    5    0    0   1   0
60 1010    0    0    0    0   0   0
61 138    1    0    0    1   0   0
62 200    0    2   17    4   0   0
63 210    0    0    0    0   0   0
64 642    0    5    0    0   1   0
65 1010    0    0    0    0   0   0
66 138    1    0    0    1   0   0
67 200    0    2   18    4   0   0
68 210    0    0    0    0   0   0
69 642    0    5    0    0   1   0
70 1010    0    0    0    0   0   0
71 138    1    0    0    1   0   0
72 200    0    2   19    4   0   0
73 210    0    0    0    0   0   0
74 642    0    5    0    0   1   0
75 1010    0    0    0    0   0   0
76 138    1    0    0    1   0   0
77 200    0    2   20    4   0   0
78 210    0    0    0    0   0   0
79 642    0    5    0    0   1   0
80 1010    0    0    0    0   0   0
81 138    1    0    0    1   0   0
82 200    0    2   21    4   0   0
83 210    0    0    0    0   0   0
84 642    0    5    0    0   1   0
85 1010    0    0    0    0   0   0
86 138    1    0    0    1   0   0
87 200    0    2   22    4   0   0
```

```
 88 210    0    0    0    0   0   0
 89 642    0    5    0    0   1   0
 90 1010    0    0    0    0   0   0
 91 138    1    0    0    1   0   0
 92 200    0    2   23    4   0   0
 93 210    0    0    0    0   0   0
 94 642    0    5    0    0   1   0
 95 1010    0    0    0    0   0   0
 96 138    1    0    0    1   0   0
 97 200    0    2   24    4   0   0
 98 210    0    0    0    0   0   0
 99 642    0    5    0    0   1   0
100 1010    0    0    0    0   0   0
101 138    1    0    0    1   0   0
102 200    0    2   25    4   0   0
103 210    0    0    0    0   0   0
104 642    0    5    0    0   1   0
105 1010    0    0    0    0   0   0
106 138    1    0    0    1   0   0
107 200    0    2   26    4   0   0
108 210    0    0    0    0   0   0
109 642    0    5    0    0   1   0
110 1010    0    0    0    0   0   0
111 138    1    0    0    1   0   0
112 200    0    2   27    4   0   0
113 210    0    0    0    0   0   0
114 642    0    5    0    0   1   0
115 1010    0    0    0    0   0   0
116 138    1    0    0    1   0   0
117 200    0    2   28    4   0   0
118 210    0    0    0    0   0   0
119 642    0    5    0    0   1   0
120 1010    0    0    0    0   0   0
121 138    1    0    0    1   0   0
122 200    0    2   29    4   0   0
123 210    0    0    0    0   0   0
124 642    0    5    0    0   1   0
125 1010    0    0    0    0   0   0
126 138    1    0    0    1   0   0
127 200    0    2   30    4   0   0
128 210    0    0    0    0   0   0
129 642    0    5    0    0   1   0
130 1010    0    0    0    0   0   0
131 138    1    0    0    1   0   0
132 200    0    2   31    4   0   0
133 210    0    0    0    0   0   0
134 642    0    5    0    0   1   0
135 1010    0    0    0    0   0   0
136 138    1    0    0    1   0   0
137 200    0    2   32    4   0   0
138 210    0    0    0    0   0   0
139 642    0    5    0    0   1   0
140 1010    0    0    0    0   0   0
141 138    1    0    0    1   0   0
142 200    0    2   33    4   0   0
143 210    0    0    0    0   0   0
144 642    0    5    0    0   1   0
145 1010    0    0    0    0   0   0
146 138    1    0    0    1   0   0
147 200    0    2   34    4   0   0
148 210    0    0    0    0   0   0
```

```
149 642   0   5   0   0  1  0
150 1010   0   0   0   0  0  0
151 138   1   0   0   1  0  0
152 200   0   2  35   4  0  0
153 210   0   0   0   0  0  0
154 642   0   5   0   0  1  0
155 1010   0   0   0   0  0  0
156 138   1   0   0   1  0  0
157 200   0   2  36   4  0  0
158 210   0   0   0   0  0  0
159 642   0   5   0   0  1  0
160 1010   0   0   0   0  0  0

# Format of RF events:
# id ampl. mag_id phase_id time_shape_id center delay freqPPM phasePPM freq phase use
# ..   Hz      ..       ..            ..     us    us     ppm  rad/MHz   Hz
rad  ..
# Field 'use' is the initial of:
#   excitation refocusing inversion saturation preparation other undefined
[RF]
1      41.6667 1 2 3 500 100 0 0 0 0 e

# Format of trapezoid gradients:
# id amplitude rise flat fall delay
# ..      Hz/m   us   us   us    us
[TRAP]
 1       800000 190 1000 190    0
 2     -31456.1  10 1980  10    0
 3       -31407  10 1980  10    0
 4      -850000 200  360 200    0
 5      19531.2  10 6400  10    0
 6     -29444.1  10 1980  10    0
 7     -27481.2  10 1980  10    0
 8     -25518.2  10 1980  10    0
 9     -23555.3  10 1980  10    0
10     -21592.3  10 1980  10    0
11     -19629.4  10 1980  10    0
12     -17666.5  10 1980  10    0
13     -15703.5  10 1980  10    0
14     -13740.6  10 1980  10    0
15     -11777.6  10 1980  10    0
16      -9814.7  10 1980  10    0
17     -7851.76  10 1980  10    0
18     -5888.82  10 1980  10    0
19     -3925.88  10 1980  10    0
20     -1962.94  10 1980  10    0
21            0  10 1980  10    0
22      1962.94  10 1980  10    0
23      3925.88  10 1980  10    0
24      5888.82  10 1980  10    0
25      7851.76  10 1980  10    0
26       9814.7  10 1980  10    0
27      11777.6  10 1980  10    0
28      13740.6  10 1980  10    0
29      15703.5  10 1980  10    0
30      17666.5  10 1980  10    0
31      19629.4  10 1980  10    0
32      21592.3  10 1980  10    0
33      23555.3  10 1980  10    0
34      25518.2  10 1980  10    0
35      27481.2  10 1980  10    0
```

```
36      29444.1  10 1980  10   0

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

# Sequence Shapes
[SHAPES]

shape_id 1
num_samples 2
1
1

shape_id 2
num_samples 2
0
0

shape_id 3
num_samples 2
0
1000


[SIGNATURE]
# This is the hash of the Pulseq file, calculated right before the [SIGNATURE]
# section was added. It can be reproduced/verified with md5sum if the file
# trimmed to the position right above [SIGNATURE]. The new line character
# preceding [SIGNATURE] BELONGS to the signature (and needs to be sripped away
# for recalculating/verification)
Type md5
Hash 3e27a278f497a21ba29f6846a021ee2a
```