PRT File Format v1.1

The Krakatoa Particle File Format (.PRT) format is the standard file format used in Krakatoa, which best supports all the options and capabilities that Krakatoa has to offer. The file format allows for the use of an arbitrary number of channels, each with a customizable type and name. This document specifies version 1.1 of the format, which adds metadata support to the file format while retaining backwards compatibility. The additions have been crafted in a way which allows v1.0 PRT file readers to continue operating without being aware of the additional data.

The file format is designed so the header is uncompressed while the particle data is compressed. This provides the ability to efficiently write a PRT file without knowing ahead of time how many particles will be written, by seeking back into the header and updating it with the correct particle count once all the particles have been written. When doing this, it is recommended that the value -1 be written to the particle count initially, so that Krakatoa and other programs can detect it as an error condition when loading the file.

Channel names are restricted, and should start with a letter or '_' character, and contain only letters, numbers and '_'. This restriction is imposed to provide for the possibility of implementing scripting support in Krakatoa and other systems where the channel names are directly used in the language to access those channels. Consider, for example, what a Krakatoa shading language would be like. Channel names are case sensitive.

Changes in v1.1

The PRT 1.1 specification is backwards compatible with conforming PRT v1.0 readers, while adding arbitrary chunks of metadata. The backwards compatibility is achieved by increasing the header version number to 2 and using the header length field to include the PRT 1.0 header as well as the 1.1 extended metadata section. No changes have been made to the way particle data is stored in the file. We define a standard chunk layout for specifying name/value pairs associated with a specific channel, or the entire particle dataset. The format is specified in a way to allow future chunk formats to be specified.

Format Specification

The PRT file format consists of:

- A header for general file information.
- A chunk section storing one or more chunks of arbitrary data.
- A channel definitions section, detailing the channel-wise data included in each particle.
- A block of binary data for the particles. This is compressed using zlib's deflate method.

All data is in little-endian byte order.

1. PRT File Header

(56 bytes)

```
(8 bytes)  Magic number that indicates the PRT file format. The number is defined by the following sequence of ASCII characters:
           {192, 'P', 'R', 'T', '\r', '\n', 26, '\n'}
(4 bytes)  A 32 bit int indicating the length of the header and chunk section (Has value 56 + size of chunk section).
(32 bytes) A human readable signature null-terminated string describing the file, currently "Extensible Particle Format".
(4 bytes)  A 32 bit int indicating version (Has value 2 in v1.1).
(8 bytes)  A 64 bit int indicating particle count.
```

2. Chunk Section (new in v1.1)

(Variable Length)

Following the header, there is a variable number of data chunks. Chunks consist of arbitrary amounts of binary data that are prefixed with a type identifier and a length. The identifier allows PRT file readers to interpret the binary data held in the chunk, while the length field allows unknown chunk types to be skipped.

The general structure of a chunk is:

Chunk Type
: A 4-byte chunk type code consisting 4 of uppercase and lowercase ASCII letters (A-Z and a-z). There is no NULL terminator. Only chunk types defined in the PRT specification can contain capital letters (ie. non-standard chunks are always in lowercase).

Chunk Data Length
: A 32 bit integer giving the number of bytes in the chunk's data field. The length counts only the data field, not itself or the chunk type code. Zero is a valid length.

Chunk Data
: The data bytes appropriate to the chunk type, if any. The length of this field is indicated by the *Chunk Data Length* value preceding it. The layout of this data is specific to the chunk type with standard chunk layouts described later in this document (in the Standard Chunks section).

The PRT specification defines two types of chunks, but developers are free to create their own types of chunk. The only caveat is that custom chunks cannot use uppercase letters in their identifier.

In a PRT file, chunks are placed contiguously until a 'Stop' chunk marks the end. A stop chunk is mandatory even when there are no other chunks.

Chunk types that are not recognized should be skipped using the length field to seek over the chunk's data section.

Chunks can appear in any order, subject to the restrictions (if any) placed on each chunk type.

Multiple chunks of the same type can appear, but only if specifically permitted for that type.

Only chunks types defined in the PRT specification can contain capital letters.

3. Reserved Bytes

(4 bytes)

```
(4 bytes) A 32 bit int, should be set to the value 4.
```

4. Channels Definition Section

(Variable Length)

Header (8 bytes)

```
(4 bytes)  A 32 bit int indicating the number of channels.
(4 bytes)  A 32 bit int indicating the length of one channel definition structure (Has value 44).
```

Channel definitions (44 bytes each)

```
(32 bytes) A null-terminated string indicating the channel name.  Must match the regex "[a-zA-Z_][0-9a-zA-Z_]*".
(4 bytes)  A 32 bit int indicating channel data type.  Supported channel data types are indicated in the table below.
(4 bytes)  A 32 bit int indicating channel arity (number of data values that each particle has for this channel).
(4 bytes)  A 32 bit int indicating channel offset, relative to the start of the particle.
```

Data Types

Data Type	Integer Value
int16	0
int32	1
int64	2
float16	3
float32	4
float64	5
uint16	6
uint32	7
uint64	8
int8	9
uint8	10

5. Particle Data

(Variable Size)

The particle data is compressed using a zlib z_streamp object, with the basic zlib deflate API. In Krakatoa, this is implemented with the deflateInit, deflate, and deflateEnd functions for compression, and the inflateInit, inflate, and inflateEnd functions for decompression.

The particles are byte-packed one after another, in the layout specified by the channels definition section.

The float16 data type is the same as the half data type in OpenEXR. The most convenient way to work with them is using the half library component from OpenEXR.

Standard Chunk Types

This section defines the layout of the binary data contained in standard chunks (which are located in the chunk section of a PRT file, following the initial header). Recall from section 2 that each chunk begins with the type identifier and the length of the data section of the chunk. After that the data is laid out in a manner specific to that chunk. The PRT spec defines two standard chunk types {'M', 'e', 't', 'a'} for named values and {'S', 't', 'o', 'p'} which marks the end of chunks.

1. {'M', 'e', 't', 'a'} Chunk Type

A {'M', 'e', 't', 'a'} section (ie. 'Meta') consists of a name/value pair that is associated with a specified channel. An empty channel name denotes global metadata that applies to the whole particle set. A PRT file will typically have many 'Meta' sections.

Associated Channel

A null terminated string with maximum of 32 characters indicating the channel these name/value pairs are associated with. The NULL is included in the 32 count. Must match the regex "[a-zA-Z_][0-9a-zA-Z_]*".

Name

A null terminated string with maximum of 32 characters indicating the name associated with this value. The NULL is included in the 32 count. Must match the regex "[a-zA-Z_][0-9a-zA-Z_]*".

Type

A 32 bit int indicating the type of the value. The numeric value for the type is taken from the table below. Unknown types should be skipped by using the length field to seek over their values.

Value

The value associated with the name. The length of the value (and therefore the arity for non-string types) is implied by the length of the chunk minus the channel, name, and type fields.

Metadata Types

Data Type	Integer Value	Notes
UTF8 string	-1	Can be of arbitrary length, and must have a NULL terminator.
int16	0	
int32	1	
int64	2	
float16	3	
float32	4	
float64	5	
uint16	6	
uint32	7	
uint64	8	
int8	9	
uint8	10	

Standard Global Name/Value Pairs

There are standard name/value pairs defined for the global 'Meta' section. Implementations are encouraged to support them and always write them to PRT v1.1 compliant files. If one of these values is not specified in the PRT file, implementations should act as if the default value (indicated below) was provided.

- **LengthUnit** An int32 representing the measurement unit of length values. Its value is drawn from this table:

Value	Name
0 (default)	unitless
1	inches
2	feet
3	miles
4	millimeters
5	centimeters
6	meters
7	kilometers

- **CoordSys** An int32 representing the handedness and up vector of the coordinate system used by 3D data. Its value is drawn from this table:

Value	Meaning
0 (default)	Unspecified
1	right handed Y-up
2	right handed Z-up
3	left handed Y-up
4	left handed Z-up

- **BoundBox** A float32[6] bounding box of the file's particle data, stored {Min_x, Min_y, Min_z, Max_x, Max_y, Max_z}. This should not be provided by the user, but instead generated automatically when writing the file. The default value should be all NaN values indicating the bounding box did not exist.

Standard Channel Name/Value Pairs

There are standard name/value pairs defined for per-channel 'Meta' chunks. Implementations are encouraged to support them and always write them to PRT v1.1 compliant files. If one of these values is not specified in the PRT file, implementations should act as if the default value (indicated below) was provided.

- **Interpretation** An int32 indicating the interpretation of a channel's data. Can be used to automatically transform data when converting units, coordinate systems, etc.

Value	Name	Description	Compatible type and arity
0 (default)	Unspecified	The semantic type is unspecified so it will not be affected by transforms	Anything
1	Point	A 3D position vector (ex. Position)	3 floats [x, y, z]
2	Vector	A 3D direction and magnitude (ex. Velocity)	3 floats [x, y, z]
3	Normal	A 3D direction orthogonal to a surface (ex. Normal)	3 floats [x, y, z]
4	Orientation	A 3D orientation encoded as a unit quaternion (ex. Orientation)	4 floats with imaginary parts first (ie. [i, j, k, r])
5	Rotation	A 3D rotation encoded as an angle & axis (ex. Spin)	4 floats with axis vector first and angle in radians (ie. [x, y, z, angle])
6	Scalar	A scalar value affected by the scale portion of a transformation (ex. Radius)	1 float

Note: When encountering an interpretation value > 6, it is better to consider that equivalent to unspecified instead of considering it an error.

2. {'S', 't', 'o', 'p'} Chunk

The {'S', 't', 'o', 'p'} section (ie. 'Stop') marks the last chunk. It must have data length 0 and appear only once, as the last chunk in the stream.

Example

Here's an example PRT file saved from Krakatoa using the vertices from a box. It has 8 particles, two channels "Position" and "Velocity", and 3 global metadata values "BoundBox", "CoordSys", and "DistanceUnit".

```
; Magic number: {192, 'P', 'R', 'T', '\r', '\n', 26, '\n'}
000000 c0 50 52 54 0d 0a 1a 0a
; Header length: 244
000008 F4 00 00 00
; Identification null-terminated string: "Extensible Particle Format"
00000c 45 78 74 65 6e 73 69 62 6c 65 20 50 61 72 74 69
00001c 63 6c 65 20 46 6f 72 6d 61 74 00 00 00 00 00 00
; Version number: 2
00002c 02 00 00 00
; Particle count: 8
000030 08 00 00 00 00 00 00 00

; Chunk type: {'M','e','t','a'}
000038 4d 65 74 61
; Chunk length: 38
00003c 26 00 00 00
; 'Meta' chunk channel name: "" (ie. The global metadata section)
000040 00
; 'Meta' chunk value name: "BoundBox"
000041 42 6f 75 6e 64 42 6f 00
; 'Meta' chunk value type: float32
00004a 04 00 00 00
; 'Meta' chunk value: {-1.0, -1.0, 0.0, 1.0, 1.0, 2.0}
00004e 00 00 80 bf 00 00 80 bf 00 00 00 00 00 00 80 3f
00005e 00 00 80 3f 00 00 00 40

; Chunk type: {'M','e','t','a'}
000066 4d 65 74 61
; Chunk length: 18
00006a 12 00 00 00
; 'Meta' chunk channel name: "" (ie. The global metadata section)
00006e 00
; 'Meta' chunk value name: "CoordSys"
00006f 43 6f 6f 72 64 53 79 73 00
; 'Meta' chunk value type: int32
000078 01 00 00 00
; Meta' chunk value: 2 (ie. right handed z-up)
00007c 02 00 00 00

; Chunk type: {'M','e','t','a'}
000080 4d 65 74 61
; Chunk length: 20
000084 14 00 00 00
; 'Meta' chunk channel name: "" (ie. The global metadata section)
000088 00
; 'Meta' chunk value name: "LengthUnit"
000089 43 6f 6f 72 64 53 79 73 00
; 'Meta' chunk value type: int32
000094 01 00 00 00
; Meta' chunk value: 1 (ie. inches)
000098 01 00 00 00
```

```
; Chunk type: {'M','e','t','a'}
00009c 4d 65 74 61
; Chunk length: 32
0000a0 20 00 00 00
; 'Meta' chunk channel name: "Position"
0000a4 50 6f 73 69 74 69 6f 6e 00
; 'Meta' chunk value name: "Interpretation"
0000ad 43 6f 6f 72 64 53 79 73 00
; 'Meta' chunk value type: int32
0000bc 01 00 00 00
; Meta' chunk value: 1 (ie. point)
0000c0 01 00 00 00

; Chunk type: {'M','e','t','a'}
0000c4 4d 65 74 61
; Chunk length: 32
0000c8 20 00 00 00
; 'Meta' chunk channel name: "Velocity"
0000cc 56 65 6c 6f 63 69 74 79 00
; 'Meta' chunk value name: "Interpretation"
0000d5 43 6f 6f 72 64 53 79 73 00
; 'Meta' chunk value type: int32
0000e4 01 00 00 00
; Meta' chunk value: 2 (ie. vector)
0000e8 02 00 00 00

; Chunk type: {'S','t','o','p'}
0000ec 4d 65 74 61
; Chunk length: 0
0000f0 00 00 00 00

; Reserved 32-bit value: 4
00000f4 04 00 00 00

; Number of channels: 2
00000f8 02 00 00 00
; Channel definition entry length: 44
00000fc 2c 00 00 00

; Name of channel 0: "Position"
000100 50 6f 73 69 74 69 6f 6e 00 00 00 00 00 00 00 00
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
; Data type of channel 0: float32
000120 04 00 00 00
; Arity of channel 0: 3
000124 03 00 00 00
; Data offset of channel 0: 0
000128 00 00 00 00

; Name of channel 1: "Velocity"
00012c 56 65 6c 6f 63 69 74 79 00 00 00 00 00 00 00 00
00013c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
; Data type of channel 1: float32
00014c 04 00 00 00
; Arity of channel 1: 3
000150 03 00 00 00
; Data offset of channel 1: 12
000154 0c 00 00 00

; Zlib compressed binary particle data
000158 78 9c 63 60 68 d8 cf 00 c6 e8 a0 c1 1e 87 f8 7e
000168 88 1c 56 f5 d8 c4 61 e6 3b e0 30 1f 5d 1c 66 3e
000178 36 f5 18 e2 00 eb aa 10 f1
```

The compressed binary particle data will consist of 24 bytes per particle, 12 for the position triplet and 12 for the velocity triplet:

```
[float32][float32][float32][float32][float32][float32][float32][float32][float32][float32][float32][float32]...
|_________________________||_________________________||_________________________||_________________________|
        position                     velocity                  position                     velocity
|____________________________________________________||____________________________________________________|
                     particle 1                                            particle 2
```

As mentioned above, it is recommended that when the particle count isn't known ahead of time, the value -1 be written to the particle count initially. This way, Krakatoa and other programs can detect it as an error condition of an incomplete file when loading the file if the count value didn't get fixed up to the correct value on completion. Once the per particle information has been packed into the file, the particle count in the header can then be updated to the true value.