Protocol Documentation

Table of Contents

Top

midi.proto

Introduction

To generate material three protobuf messages must be supplied to the mmm::sample function. The specification for each of these messages (Piece, Status and SampleParam) is outlined in this document. Any field which starts with internal_ should be ignored, as these are for internal use only. Or if you really know what you are doing ;). There are examples of midi::Piece, midi::Status and midi::SampleParam objects in the docs folder. For working examples, please consult the testing suite, which is found in MMM_API/src/mmm_api/test/unit.cpp.

Functionality Overview

Functionality	Scope	Description
Velocity	Always Enabled	32 levels of loudness for individual notes.
Instrument	Track	The General MIDI instrument (i.e. Timbre).
Max Polyphony Hard-Limit	Track	A hard-limit on the number of simultaneously sounding notes.
Note Duration (upper/lower soft bounds)	Non-Drum Tracks	Tells the model what the 15th (lower) and 85th (upper) quantiles of the note duration (i.e. Quarter, Whole) distrbution should be.
Polyphony (upper/lower soft bounds)	Non-Drum Tracks	Tells the model what the 15th (lower) and 85th (upper) quantiles of the polyphony distrbution should be.
Density (10 levels)	Drum Tracks	Tells the model the number of notes per bar to produce
Auto-regressive Sampling Mode	Track	When enabled, bars are always sampled in chronological order.
Time Signature	Bar	A unique time-signature can be specified for each bar.
Temperature	per API call	A higher value increases entropy of generated output. Temperature=1 applies no modification to the probabilities produced by the model.
Context size (model_dim)	per API call	The number of bars that the model can process in one API call

Parameter Constraints and Considerations

There are two sampling methods: autoregressive generation, where we progressively sample musical material forwards in time on each track; and conditional generation (bar-infilling), where generated material is conditioned on past and future material.

Note that a single call the the model mmm:sample() may involve both autoregressive and conditional generation, as these can be specified on a per-track basis. These constraints are

Sample Param Constraints

1. tracks_per_step :

- must be on range [1,number of tracks in piece]

2. bars_per_step :

- must be on the range [1,model_dim] - for conditional generation it is ill-advised for the user to have bars_per_step == model_dim, as this means generation will not be conditioned on any bars

3. shuffle :

- this only applies in cases where one or more tracks are conditionally generated (i.e. resample = False && 1+ selected_bars = True)

4. percentage :

- this only applies in cases where one or more tracks are conditionally generated (i.e. resample = False && 1+ selected_bars = True)

Status Constraints

1. density :

- this control only applies to drum tracks. This works will both infilling and autoregressive mode.

2. note duration / polyphony :

- this control only applies to non-drum tracks. This works will both infilling and autoregressive mode.

3. autoregressive :

- you can only enable autoregressive mode (resample = True) when all the bars are selected in a track. - note you may have autoregressive disabled when all bars are selected in a track

4. ignore :

- bars which have 1+ selected_bars = True may not be ignored, as they are needed to condition the generation

Protobuf Specification

Bar

The Bar message specifies the events occuring in a bar.

Field	Type	Label	Description
events	int32	repeated	A list of integers, which are simply the indices of the messages found in the Piece.events repeated message. Note offsets which occur at the end of the bar (i.e. event.time = 48 with a time signature of 4/4 and piece.resolution of 12) should be included in the current bar rather than the next bar. In other words, no note offsets should even have an event.time = 0, as these note offset events would belong in the previous bar.
ts_numerator	int32	optional	Numerator for the time-signature of the bar. Note that while time signatures can vary from bar to bar, they cannot vary from track to track. In other words if the second bar in track 0 has a time signature of 4/4, the second bar in track 1 must also have a time signature of 4/4.
ts_denominator	int32	optional	Denominator for the time-signature of the bar.
internal_beat_length	float	optional	
internal_has_notes	bool	optional	
internal_feature	ContinuousFeature	repeated	

Event

The Event Message is used to represent a MIDI note onset or offset.

Field	Type	Label	Description
time	int32	optional	The time of the event (either a note onset or note offset) relative to the current bar in quantized steps. Currently, most model quantize each quarter note beat into 12 subdivisions. As a result, if the event happens an eighth note after the start of the bar, this value would be 6. If the event occurs three quarter notes after the start of the bar, this value would be 3 * 12 = 36.
velocity	int32	optional	The MIDI velocity. This value must be 0 for note off messages.
pitch	int32	optional	The MIDI pitch value of on the range [0,128).
internal_instrument	int32	optional	
internal_track_type	int32	optional	
internal_duration	int32	optional	

Piece

The Piece message specifies the actual musical material in a track-separated event-based format, specifying the note onsets and offsets for each bar in each track.

Field	Type	Label	Description
tracks	Track	repeated	Organizes MIDI events into tracks and bars. In short, each track contains a list of bars, which in turn contains a list of event indices (corresponding to the repeated events message in the Piece.
events	Event	repeated	A list of MIDI events which the tracks and bars reference
resolution	int32	optional	The time resolution used to quantize / discretize musical material. Unless otherwise instructed, this should be set to 12.
tempo	int32	optional	Optionally the tempo can be specified. However this is not taken into consideration by the model.
internal_valid_segments	int32	repeated	
internal_valid_tracks	uint32	repeated	
internal_segment_length	int32	optional	
internal_valid_tracks_v2	ValidTrack	repeated	
internal_genre_data	GenreData	repeated	

SampleParam

The SampleParam message specifies hyper-parameters for generation.

Field	Type	Label	Description
tracks_per_step	int32	optional	For multi-step generation (typically employed when the entire piece is too large to be considered by the model simultaneously) this parameter specifies the number of tracks that are generated in each step.
bars_per_step	int32	optional	For multi-step generation this parameter specifies the number of bars that are generated in each step. This value should be set in relation to model_dim. If bars_per_step = model_dim, then there will be no horizontal conditioning, which will typically produce inferior results. A good rule of thumb is to use bars_per_step == model_dim / 2.
model_dim	int32	optional	The size of the model. In most cases this will be 4.
percentage	int32	optional	The percentage of the selected material (selected bars in the Status message) that will be generated.

Field	Type	Label	Description
batch_size	int32	optional	The number of outputs to be generated. Currently we only support batch_size=1. With multi-step sampling its is likely more efficient to simply make several calls in series.
temperature	float	optional	Allows for the entropy of generation to be adjusted. When temperature=1, the probability distributions output by the model are unaltered. When temperature<1 the probability distribution is increasingly biased towards the most probable tokens. With a very small temperature value this would be equivalent to argmax sampling. When temperature>1 the probability distribution moves towards a random uniform distribution. It is recommended to keep this value close to 1 in most cases.
max_steps	int32	optional	The max number of tokens to generate before terminating generation. Can be used to avoid memory overload. When this value is set to zero it is ignored, and no limitations are set of the number of generated tokens.
polyphony_hard_limit	int32	optional	Sets a hard limit on the polyphony accross all tracks. This is implemented by keeping a record of all the currently sounding notes, and preventing the model from generating note-onset tokens when the limit is reached.
shuffle	bool	optional	When shuffle=true the generation steps are randomly ordered. For obvious reasons, auto-regressive sampling cannot be used with shuffle=true, as it would cease to be auto-regressive.
verbose	bool	optional	Mainly for debugging purposes.
ckpt	string	optional	The path to the ckpt, which should either be an absolute path or relative to the executable.
internal_skip_preprocess	bool	optional	
internal_random_sample_mode	bool	optional	
internal_disable_masking	bool	optional	

Status

The Status message specifies which bars or tracks are to be generated/conditioned on, and provides extra information about conditioning such as instrument, density, polyphony and note-duration.

Field	Type	Label	Description
tracks	StatusTrack	repeated	

StatusTrack

The StatusTrack message specifies per-track information for generation.

Field	Type	Label	Description
track_id	int32	optional	The index of a track in the Piece message. For a track to be seen by the model, it must be referenced by a StatusTrack message via the track_id field. Tracks that are not referenced by a StatusTrack message will not be considered by the model.
track_type	TRACK_TYPE	optional	This must be a value in the TRACK_TYPE enum. This should be equivalent to the TRACK_TYPE specified in the corresponding Track, unless you are giving the model an option to choose either a drum or instrument track. In this case use the STANDARAD_BOTH value here, and the TRACK_TYPE in the piece will be ignored.
instrument	GM_TYPE	optional	This must be a value in the GM_TYPE enum. It specifies the set of possible instruments that the model may choose from. The mapping between GM_TYPE and instrument numbers can be found in src/mmm_api/enum/gm.h. For example, using midi::GM_TYPE::piano will allow the model to use any piano instrument.

Field	Type	Label	Description
selected_bars	bool	repeated	A list of boolean values which specifies whether a bar is to be generated (true) or conditioned on (false). This must be the same length as the number of bars in the corresponding Track message.
autoregressive	bool	optional	Indicates whether or not to use auto-regressive sampling. Note that you can only use auto-regressive sampling when each value in selected_bars is true (i.e. the entire track is being generated). Note that you do not have to use auto-regressive sampling when all selected bars is all true.
ignore	bool	optional	This indicates that the track should be ignored. The model will not be conditioned on this track, and it will in no way effect the generated outcome.
density	DensityLevel	optional	
min_polyphony_q	PolyphonyLevel	optional	
max_polyphony_q	PolyphonyLevel	optional	
min_note_duration_q	NoteDurationLevel	optional	
max_note_duration_q	NoteDurationLevel	optional	
internal_ts_numerators	int32	repeated	
internal_ts_denominators	int32	repeated	
internal_embeds	ContinuousFeature	repeated	
internal_genre	string	optional	

Track

The piece message contains a list of bars, and specifies the instrument and track_type.

Field	Type	Label	Description
bars	Bar	repeated	A list of bars. Note that each track must have the same number of bars.
instrument	int32	optional	The MIDI instrument number for the track.
track_type	TRACK_TYPE	optional	This must be a value in the TRACK_TYPE enum. In most cases, using STANDARAD_TRACK and STANDARD_DRUM_TRACK will suffice to denote a non-drum instrument track and a drum track respectively.
internal_train_types	TRACK_TYPE	repeated	
internal_features	TrackFeatures	repeated	

DensityLevel

Specify the minimum or maximum amount of note density using these values. Using DENSITY ANY lets the model choose the level of density.

Name	Number	Description
DENSITY_ANY	0	
DENSITY_ONE	1	
DENSITY_TWO	2	
DENSITY_THREE	3	
DENSITY_FOUR	4	

Name	Number	Description
DENSITY_FIVE	5	
DENSITY_SIX	6	
DENSITY_SEVEN	7	
DENSITY_EIGHT	8	
DENSITY_NINE	9	
DENSITY_TEN	10	

NoteDurationLevel

Specify the minimum or maximum bounds for note-duration using these values. Using DURATION_ANY lets the model choose the bounds for note duration.

Name	Number	Description
DURATION_ANY	0	
DURATION_THIRTY_SECOND	1	
DURATION_SIXTEENTH	2	
DURATION_EIGHTH	3	
DURATION_QUARTER	4	
DURATION_HALF	5	
DURATION_WHOLE	6	

PolyphonyLevel

Specify the minimum or maximum amount of polyphony using these values. Using POLYPHONY_ANY lets the model choose the level of polyphony.

Name	Number	Description
POLYPHONY_ANY	0	
POLYPHONY_ONE	1	
POLYPHONY_TWO	2	
POLYPHONY_THREE	3	
POLYPHONY_FOUR	4	
POLYPHONY_FIVE	5	
POLYPHONY_SIX	6	

Top

track_type.proto

TRACK_TYPE

Name	Number	Description
OPZ_KICK_TRACK	0	
OPZ_SNARE_TRACK	1	

Name	Number	Description
OPZ_HIHAT_TRACK	2	
OPZ_SAMPLE_TRACK	3	
OPZ_BASS_TRACK	4	
OPZ_LEAD_TRACK	5	
OPZ_ARP_TRACK	6	
OPZ_CHORD_TRACK	7	
AUX_DRUM_TRACK	8	
AUX_INST_TRACK	9	
STANDARD_TRACK	10	
STANDARD_DRUM_TRACK	11	
STANDARD_BOTH	12	
OPZ_INVALID_DRUM_TRACK	13	
OPZ_INVALID_SAMPLE_TRACK	14	
OPZ_INVALID_BASS_TRACK	15	
NUM_TRACK_TYPES	16	

Scalar Value Types

.proto Type	Notes	C++	Java	Python	Go	C#	PHP	Ruby
double		double	double	float	float64	double	float	Float
float		float	float	float	float32	float	float	Float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int	int32	int	integer	Bignum or Fixnum (as required)
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long	int64	long	integer/string	Bignum
uint32	Uses variable-length encoding.	uint32	int	int/long	uint32	uint	integer	Bignum or Fixnum (as required)
uint64	Uses variable-length encoding.	uint64	long	int/long	uint64	ulong	integer/string	Bignum or Fixnum (as required)

.proto Type	Notes	C++	Java	Python	Go	C#	PHP	Ruby
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int	int32	int	integer	Bignum or Fixnum (as required)
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long	int64	long	integer/string	Bignum
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int	uint32	uint	integer	Bignum or Fixnum (as required)
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long	uint64	ulong	integer/string	Bignum
sfixed32	Always four bytes.	int32	int	int	int32	int	integer	Bignum or Fixnum (as required)
sfixed64	Always eight bytes.	int64	long	int/long	int64	long	integer/string	Bignum
bool		bool	boolean	boolean	bool	bool	boolean	TrueClass/FalseClass
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode	string	string	string	String (UTF-8)
bytes	May contain any arbitrary sequence of bytes.	string	ByteString	str	[]byte	ByteString	string	String (ASCII-8BIT)