MDCG101-201710-R00

Standard MIDI File Compiler

技術解説書

wnc develop

□目次

◇技術解説書を作成するにあたって 注） としているところは2017年現在のバージョンにおいて変更しているところや現在の記述にそぐわない内容に注意として加えたものです

注）２００１年当時の開発記録を加筆修正しているため現在のバージョンに搭載されていない機能があります。また、内部使用を想定した解説書ですので書き方に統一性が無く不快な表現がある場合があります

■mdcompの開発目的

簡単に入力でき昔のＢＡＳＩＣで使用していた記号やルールで作曲できる。移植がソースをほとんど変更せずにできることを目的および目標とする

■製作に対する使用機材など

コンパイラ(GUI版)
Borland C++Builder Standard(キャンペーン版)
Borland C++Builder 3 Professional
Borland TurboC++2006 Explorer　　(mdcomp32 v1系 , 2017年)

コンパイラ(16bitDOS版)
LSI-C86試食版 v3.30c(LSI-Japan社)
Turbo C English v2.01(Borland社)
Digital Mars C/C++ Compilers v8.57 + DOS16 libraries v8.50 (2017年)

コンパイラ(32bitDOSプロンプト版)
Borland C++Builder 3 Professional
Borland C++Compiler 5.5(無償配布版)
Borland TurboC++2006 Explorer
Digital Mars C/C++ Compilers v8.57 (2017年)

■開発機材

TOSHIBA DynaBook SS 3010 (2000年10月14日購入)
→同機械上でBorland C++Builder3 Professionalを使用
NEC PC98-NX VS26D (2000年４月廃棄→2000年９月修理済)
→同機械上でBorland C++Builder Standard(ｷｬﾝﾍﾟｰﾝ版)使用,部分開発のみ
NEC PC-9821Xv13/W16(1996年9月導入)
EPSON AT971(2010年3月導入)
NEC PC-GL21DJ5(2016年6月導入, 修理済)
(Pentium P6200@2.13GHz, 6GMBメモリ, Windows10 Pro64b1703,MSGS)

注) PC-GL21DJ5以外は当時の開発に使用していた機材です

■mdcompとFMCOMPの違い

Standard MIDI File コンパイラ(以下mdcomp)は、FMCOMPの別バージョンであり、mdcompはすべての機能を含んでいる。特にコード解析部分は共通で統合開発に移った現在のmdcomp(GUI版)でも同様の解析アルゴリズムを使用している。また、内部用コードもmdcompではFMCOMPと同じコードである。

mdcompでは追加してこの内部用コードからMIDIコードに変換する関数がある。内部用コードはアドレス情報を含まないfmdrv(PC-98版：DOS)用コードそのもので、２バイトからなるデータの集合体である。ここでいう２バイトのコードは１バイト目が機能コード、２バイト目は長さまたは意味のある８ビットの数値としている。機能コードは２種類あり、音階コードまたは、制御コードである。音階コードは０～９６までで、０は無音１～９６まではオクターブごとの音階コードで制御コードは２５５から逆順に使用する。

■内部仕様(V1.01.00時点)

１行最大処理文字数	2000バイト分(16bitDOS版は512バイト分)
制御コード数	10命令
1CHメモリ確保数	64000+3 byte(16bitDOS版は6000+3 byte)
最大行数	999999行(コンパイルリミッタ)
エディタ最大格納領域	1000000バイト(GUI版のみ)
チャンネル数	9
連符最大数	32
CFGサイズ	512バイト(GUI版のみ)
ﾐﾃﾞｨｷｰADJUST	23(24でないのは0が無音のため)

■定義

@音階コード表

	o0	o1	o2	o3	o4	o5	o6	o7
C	1	13	25	37	49	61	73	85
C#	2	14	26	38	50	62	74	86
D	3	15	27	39	51	63	75	87
D#	4	16	28	40	52	64	76	88
E	5	17	29	41	53	65	77	89
F	6	18	30	42	54	66	78	90
F#	7	19	31	43	55	67	79	91
G	8	20	32	44	56	68	80	92
G#	9	21	33	45	57	69	81	93
A	10	22	34	46	58	70	82	94
A#	11	23	35	47	59	71	83	95
B	12	24	36	48	60	72	84	96

※0番は無音コード

@長さ表

長さ	定義値	長さ	定義値
1.	192	16.	12
1	128	16	8
2.	96	32.	6
2	64	32	4
4.	48	64.	3
4	32	64	2
8.	24	128	1
8	16		

注)fmcompデータの定義です

注)SMF時、長さは15倍した値を出力

@機能コード（内部専用） V1.01.00までにおいて

機能番	値	意味
255	255	終了コード
254	楽器No.	楽器変更 0～127(初期値=0)
253	音量	音量変更 0 ～ 16 (初期値=10)
252	テンポ	テンポ変更 10～250 (初期値=120)
251	0	&命令
250	割合	Q命令 1～8(初期値=8または7)
249	割合	音の左右振り分け(0～127)

■制御コード(データ外命令)

命令コード	文字	読み	意味
0	//	ダブルスラッシュ	改行コードまでコメントとする
1	ch	シーエイチ	チャンネルオープン準備
2	{	ナミカッコヒラキ	チャンネルオープン
3	″	ダブルｸｫｰﾃｰｼｮﾝ	データ開始または終了
4	}	ナミカッコトジ	チャンネルクローズ
5	/*	ｽﾗｯｼｭアステリスク	コメント開始
6	*/	アステリスクｽﾗｯｼｭ	コメント終了
7	op	オーピー	オプション指定
8	cr	シーアール	著作権表示文字列
9	sn	エスエヌ	シーケンス名文字列

■パラメータ定義

音量	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	(16)
SMF	0	7	15	23	31	39	47	55	63	71	79	87	95	103	111	119	127

※BASICでは16は無いためSMF最大値使用向け

@基本値定義

```
MEL_MU = 0  無音
MEL_C  = 1  ド
MEL_CS = 2  ド＃
MEL_D  = 3  レ
MEL_DS = 4  レ＃
MEL_E  = 5  ミ
MEL_F  = 6  ファ
MEL_FS = 7  ファ＃
MEL_G  = 8  ソ
MEL_GS = 9  ソ＃
MEL_A  = 10 ラ
MEL_AS = 11 ラ＃
MEL_B  = 12 シ
```

```
L_0    = 0      0
L_1T   = 1      1.
L_1    = 2      1
L_2T   = 3      2.
L_2    = 4      2
L_4T   = 5      4.
L_4    = 6      4
L_8T   = 7      8.
L_8    = 8      8
L_16T  = 9      16.
L_16   = 10     16
L_32T  = 11     32.
L_32   = 12     32
L_64T  = 13     64.
L_64   = 14     64
L_128  = 15     128
```

■データコード

命令	読み	意味	初期値
o	オー	オクターブ変更	4
L	エル	無指定時に使用する長さ変更	4
R	アール	休符	―――
@	アットマーク	楽器番号変更	0
T	ティー	テンポ変更	120
#	ナンバー(*下参照)	半音上げ	―――
+	プラス	半音上げ	―――
-	マイナス	半音下げ	―――
&	アンド	音をつなげる	―――
<	小なり	オクターブを下げる	―――
>	大なり	オクターブを上げる	―――
V	ブイ	ボリューム変更	10
Q	キュー	発生時間割合変更	8 or 7
{	ナミカッコヒラキ	連符開始	―――
}	ナミカッコトジ	連符終了	―――
:	コロン	直接長さ指定	―――
$	ドル	Panpot指定	64
N	エヌ	直接音階番号指定	―――

命令	読み	音階
c	シー	ド
d	ディー	レ
e	イー	ミ
f	エフ	ファ
g	ジー	ソ
a	エー	ラ
b	ビー	シ

※アルファベットは大文字・小文字関係なく使用できる

注）読みは正しくない場合があります

*）正式にはナンバーと呼ぶようです（シャープでも良い気もするが）

■各命令と意味(番号はプログラム上で実際に使用)

命令０番　//［ダブルスラッシュ］出現以降改行までコメント

↓行開始
例) ch1 { ″cde″ }　// ドレミの音発生(改行)
↑　　↑　　↑　↑この部分は実際には改行コード
①　　②　　③
①から命令を解析しはじめ②で//を発見した場合③まで解析用ポインタを進める。

命令１番　ch　［シーエイチ］チャンネル番号指定（オープン前に指定）

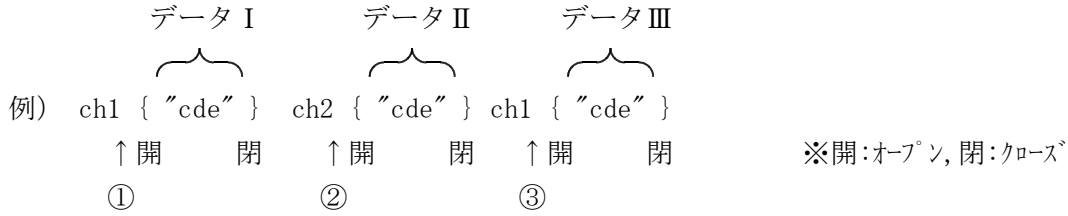
例)　ch1{″cde″}
↑↑↑
①②③

①から命令を解析しはじめ②ch文字発見直後の番号③をチャンネルオープン用の数値と認識する。数値は１～９までで９はリズムチャンネルになる。これはMIDI ch10番を指すことになるが、MT-32を使用する場合に対応するため MIDI ch2～ch10までの9チャンネルを使用するためである。MT-32はch1にデータを出力しても音がでないのでMIDI ch2からということにした。

mdcomp(ch)	MIDI(ch)
ch1	ch2
ch2	ch3
:	
ch9	ch10

命令２番　{　［ナミカッコヒラキ］チャンネルオープン

ch命令(命令１番)によって指定したチャンネルをオープンする。一度オープンしクローズ（命令３・４番）したチャンネルを再オープンした場合は続けてそこに書き込む。



①でチャンネル１を指定後②でチャンネル２を指定しているが再び③でチャンネル１を指定している。上記の例は、

ch1 = cdecde(Ⅰ+Ⅲ)　ch2 = cde(Ⅱ)

と意味解析する。

命令３番　″　［ダブルクォーテーション］　データ開始または終了

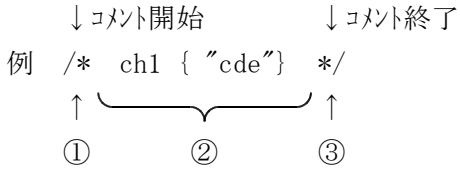
｛｝（命令２番・４番)によってオープン・クローズするチャンネルのデータ領域を指す。ch(命令１番)および{（命令２番）がすでに実行している必要がある。

※命令２番を参照のこと

命令４番　｝［ナミカッコトジ］　チャンネルクローズ

{（命令２番）でオープンしたチャンネルをクローズする。

命令５番　　/*　［スラッシュアステリスク］　コメント開始



①のコメント開始を発見後ポインタを進め、③のコメント終了を発見までは解析対象としない(②は無視する)。

注）コメントの入れ子はできない
例えば　　/* abc /* abcde　*/　fgh */　は
　　/*abc/*abcde */　までをコメントと見なし
　　fgh */　を解析しようとする.

命令６番　　*/　［アステリスクスラッシュ］　　コメント終了

コメント開始の/*(命令５番)から*/までをコメントとし解析対象から除く。

※命令５番の例参照

命令７番　op　［オーピー］　オプション指定 ※v0.00.08以降

パラメータを渡すことにより各種設定を切り替える

```
op（ AUTOLR ）
↑↑　↑　　↑
①②　③　　④
```

①でop命令を発見し②の ’(’ から④の ’)’ の中③を設定を変更する命令と認識する。設定を変更する命令は以下のように定義している。

名称	意味	変更するｽｲｯﾁ
AUTOLR	自動左右配分とする	f_autolr(trueへ)
MANULR	手動左右配分とする	f_autolr(falseへ)
Q7DEF	Q7を標準とする(88VA互換)	f_q7def(trueへ)
Q8DEF	Q8を標準とする	f_q7def(falseへ)

③には ，(カンマ)で区切ることにより複数同時に設定を切り替えられる。ただし、対照的に働く設定は最後に出現したものを有効にする。

例えば、

```
op( Q7DEF , Q8DEF)
```

のようにQ7DEF,Q8DEFを同時に指定したときは Q8DEFが有効となる.

また、命令は大文字・小文字区別しない。

※AUTOLRとAutoLRなどのように大文字・小文字が混在してもかまわない.

命令８番　cr　［シーアール］　著作権文字列指定 ※v0.00.09以降

SMFに著作権表示用文字列を組み込みます(最大63Byte mdcomp32固有リミッタ)

例）　cr（ ″(C)2013 wnc develop″）　//必ず 文字列を ″ で囲む必要がある

命令９番　sn　［エスエヌ］　シーケンス名組み込み ※v0.00.10以降

説明：SMFにシーケンス名文字列を組み込みます(最大127Byte mdcomp32固有リミッタ)

例）　sn（ ″The Original Music 2002″）　//必ず 文字列を ″ で囲む必要がある

■データ命令

命令　o　［オー］　　オクターブ番号を指定

例）　o4cdeo5cde

　オクターブ４のドレミとオクターブ５のドレミと解析する。コンパイラ初期値は４なので、o命令がはじめて出現（各チャンネルごとに）した場合まではo4とする。

　指定できる数は０～７までで以下の命令が使用できる。
　o0　o1　o2　o3　o4　o5　o6　o7

oは大文字・小文字関係なく使用できるため以下も有効

　00　01　02　03　04　05　06　07

使用例

　　ch1{"cdefgabo5cdefgabo6cde"}

命令　L ［エル］　無指定時に使用する長さ

例）L16cdeL8def
16分音符cdeと8分音符defと解析する

指定可能な値は1, 2, 4, 8, 16, 32, 64, 128で, 点は1, 2, 4, 8, 16, 32, 64に付加できる。

以下は使用可能な例である。

L1.　L1　L2.　L2　L4.　L4　L8.　L8　L16.
L16　L32.　L32　L64.　L64　L128

命令　R ［アール］ 休符　指定音符分無音にする

例）　CDERCDER32
ドレミ休符ドレミ休符（３２分）
命令Lで指定した値が使用される

指定可能な値は命令Lと同じで 1, 2, 4, 8, 16, 32, 64, 128で点は1, 2, 4, 8, 16, 32, 64に付加できる。
省略時は命令Lで指定している長さを使用する。

例)L8　R16 R　R4
↑　↑　↑　↑
①　②　③　④
上記の例では、①L命令により省略時の長さを指定し②16分音符、③８分音符(Lで指定の値)④四分音符となる。

命令　＠　［アットマーク］　　楽器の変更

例）　@20 CDE
　　↑ ⌣
　　①　②

①で楽器を２０番のものに変更する。(MIDI音源の仕様書を参照)
②では楽器番号２０番の楽器でドレミを発声。
※次に＠命令が出現するまで楽器番号は保持
　各チャンネル毎に別々な楽器番号が指定可能

@0　@1　@2　....(省略)....@99 @100 ...(省略)... @126 @127

指定できる番号は0～127までの128種類
MIDIの仕様(GM)にあわせてある。

命令　Ｔ　［ティー］　　テンポ変更

例）　T200 CDE
　　↑　↑
　　①　②

①でテンポ２００に変更し②ではテンポ２００のCDEを発声
　※次にＴ命令が出現するまで保持

T10 T11 ....(省略).... T249　T250

指定できる数値は 10～250 までである。
T120は、１分間に四分音符を１２０発声するほどの速さ
♩=120と同等

注）命令Tは複数存在しても良いが同じch内で使用すること(SMF出力時)
　複数のchに存在した場合解析の順番はch1から出現順に時間を計算して最初のトラックに書き込む。設計上、ch1すべて計算後ch2・ch3を追加で挿入する形にしているため

命令　＃　［ナンバー］　　半音上げ

例）C DE C# DE C DE
　　↑　↑　　↑
　　①　②　　①
①のＣは通常のドで②はド＃になる。

音階記号　　cdefgab　に使用できるが b#は同じオクターブのCと同等になる
(N88BASICの仕様にあわせてある)

音符指定も可能（以下に例）

　　Fと同等になる
　　⌒
C#16　　C#64　　E#8
16分音符ド＃　64分音符　8分音符E#(ファと同等)

命令　＋　［プラス］　　半音上げ

＃と同じ機能であるため説明は省略

例）　C+DEC+16DE
　　↑　↑
　　①　②

①ド＃　　②１６分音符ド＃

命令　－　［マイナス］　　半音下げ

例）　　C D EC D- EC D E
　　↑　↑　　↑
　　①　②　　①

①のDは通常のレで、②はレ♭(フラット)となる

音階記号 cdefgab に使用できるが
　c-は、同じオクターブのBと同等になる
(N88BASICの仕様に合わせてある)

音符指定も可能（以下に例）
　　Eと同等になる
　　↓
　D-16　D-64　F-8
16分音符レ♭ 64分音符 8分音符ファ♭(ミと同等)

命令　＆　［アンド］　　音と音をつなげる

例）　　C8&C16
　　↑　↑
　　①　②

①のドと②のドの長さ（①＋②）をあわせた長さでドを発声する。&(アンド)命令は複数連結に使用できるが単体では使用不可。以下に複数&の例

例１）C16&C32&C64　例２）C16&D32

例１の場合ドの音を複数連結だが、例２の場合D32は無効でC16&C32と同等になる。

命令　＜　［小なり］　　オクターブを１つ下げる

ｵｸﾀｰﾌﾞを5にする
↓
例）o5 C ＜ C ＜ C
↑　↑　↑
①　②　③

①のドはオクターブ５であるが②ではオクターブ４、③ではオクターブ３となる。以下の場合も有効。

o5 c＜＜＜c
↑　↑
①　②

①ではオクターブ５のドであるが②ではオクターブ２になる

＜ 命令は、o命令（オクターブ変更）のデータ保存場所に保管しているオクターブ値を１マイナスするためo命令で最後に指定した（ ＜ 命令出現手前のこと）オクターブからのマイナスとなる。
＜ 命令をいくつならべてもオクターブ０までしか下がらない。

使用場所としては音の位置関係を相対的に表したい場合にもちいることが望ましい。
（音楽ソースの先頭だけo命令により固定すれば容易に高低を変更できる）

命令　＞［大なり］　オクターブを１つ上げる

例）　o2cde>def>cde
　　↑　↑　↑
　　①　②　③

①のドレミはオクターブ２であるが②はオクターブ３のレミファである。③ではオクターブ４のドレミとなる。以下の場合も有効

o3　c>>>c
　↑　↑
　①　②

①ではオクターブ３のドであるが、②ではオクターブ６である。

＞命令は＜命令（オクターブを１つ下げる）と使用することにより音階位置を相対的にあらわすことが可能になる。
＞命令をいくつ並べてもオクターブ７までしか上がらない。
＜命令と同様にo命令(オクターブ変更)の値保管場所に入っているオクターブ数に１プラスするためo命令で最後に指定した(＞ 命令直前のこと)オクターブに加算する。

命令　V ［ブイ］　音量変更

例）　V15 CDE
　　↑　↑
　　①　②

①で音量を１５にする（各チャンネル別）②のドレミは音量１５のドレミとする。

使用例）　V10 CDE V15 CDE
　　↑　↑　↑　↑
　　①　②　③　④

①で音量を10にする。②は音量10のドレミ、③で音量を15に変更する。④は音量15のドレミとなる。

※指定できる数値は０～１６まで（16はBASICでは無）

命令　Q ［キュー］　発生時間の割合指定

例）　Q8 C8D16 Q7 C8D16
　　↑　↑　↑
　　①　②　③　④

①で割合を8(8/8=100%)とする。②では8分音符の時間を全てドを発声、16分音符の時間全てレの音を出す。③では割合を7(7/8)とする。

④では８分音符×7/8時間分ドを発声し、８分音符×1/8時間分無音とする。続いて16分音符×7/8時間分レを発声し、16分音符×1/8時間分無音とする。

Q	1	2	3	4	5	6	7	8
発生時間	12.5%	25%	37.5%	50%	62.5%	75%	87.5%	100%
無音時間	87.5%	75%	62.5%	50%	37.5%	25%	12.5%	0%

なお、Qに0は指定できない。

命令　{ }　[ナミカッコヒラキ][ナミカッコトジ]　連符指定

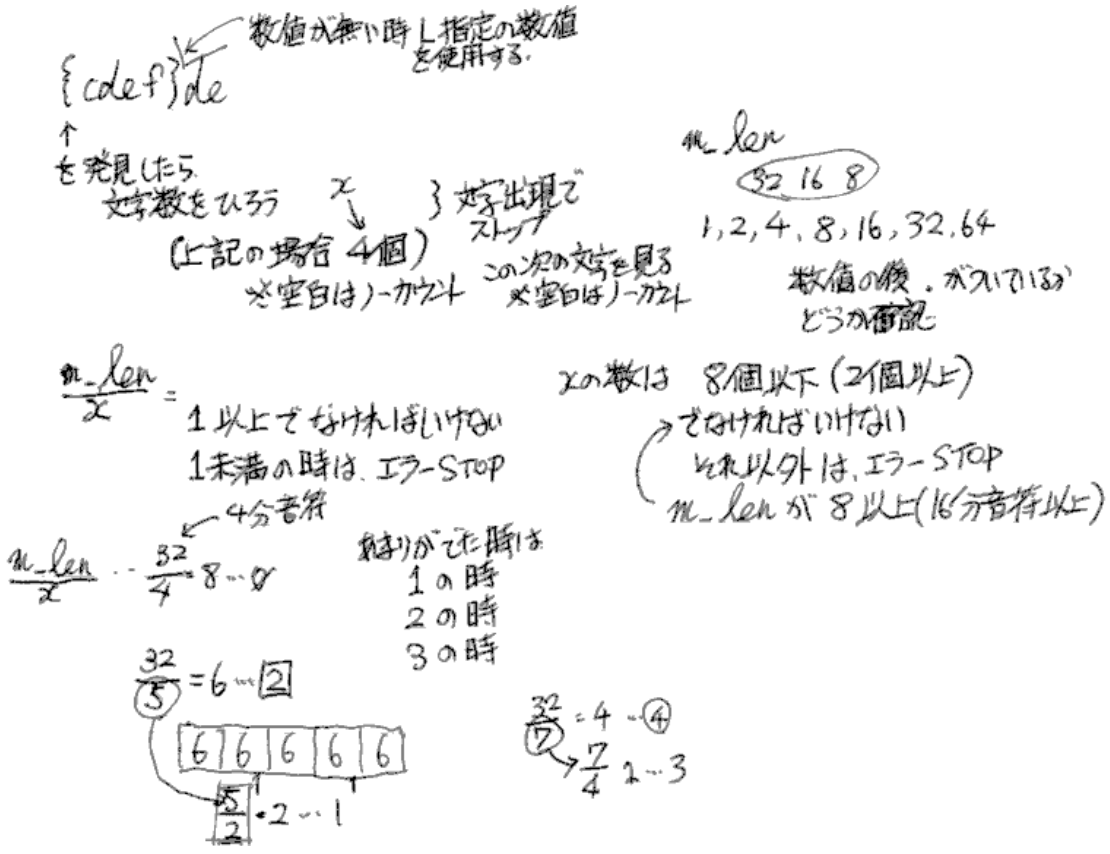
例）　{CDE}8

①　②

①で指定した音階を②で指定した音符の長さに入れて発生する。

連符として指定できる音階数は３２音まで、短い音符の中に多くの音階を指定すると発声しない音がでてくる。

2001.3.23pmでの連符（案）写　※案のため実装と一部異なる（連符最大数など）



命令　：　［コロン］　　長さ直接指定

例）　C:100

ドの長さを100として発声（四分音符=32である）

注）この命令は連符機能が無い時に使用していた。v0.00.05で連符機能を付加したため本来の目的は失われている。

指定できる値は１～２５５までで、長さ調整に使用する。

命令　$　ドル　左右音の配分(panpot)指定

例）　$127

値によって命令出現後の音の配分を変更する。

左------中央------右
0　　64　　127

※ステレオスピーカーとした場合

■音階関係命令

命令　C　　シー　　ドの音

例）　C8　C#16　C:100　C+　C-
　　↑　↑　↑　↑　↑
　　この部分に音階コード

ドの音を発声する。音階コードの後に音符指定などができる。

命令　D　　ディー　　レの音
命令　E　　イー　　ミの音
命令　F　　エフ　　ファの音
命令　G　　ジー　　ソの音
命令　A　　エー　　ラの音
命令　B　　ビー　　シの音

例は命令Cと同様なので省略。
大文字・小文字関係なく共通である。

命令　N　　エヌ　　直音指定

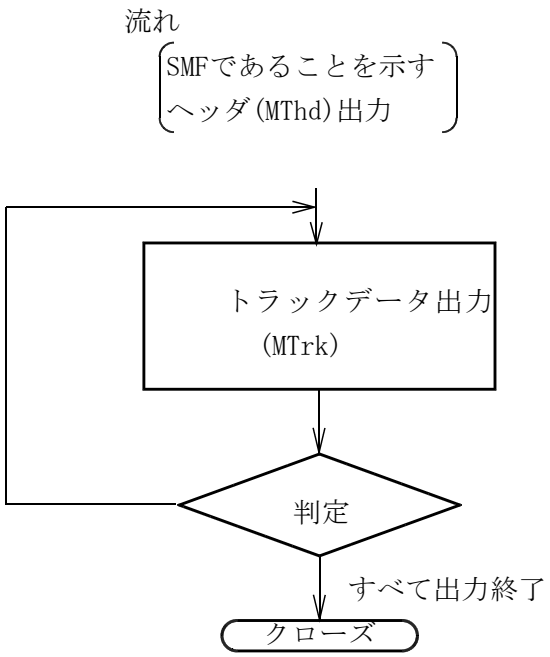
例）　N49
　　↑
　　①

①の49はオクターブ４のドである（定義・音階コード表参照）

数値は０～９６までで０は無音

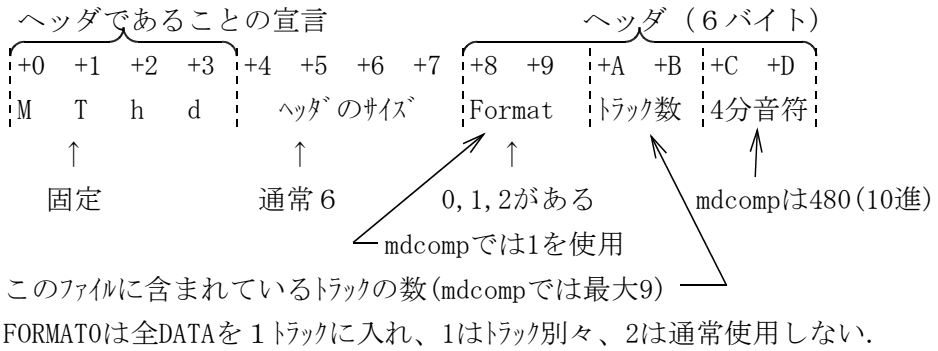
■SMF作成機能について

mdcompは独自の関数からSMFを作成できる。以下にmdcompで作成するSMFについて記す。



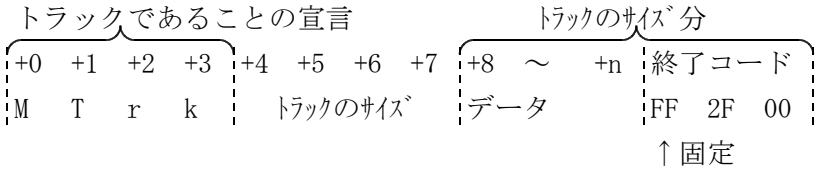
トラックデータ出力は、データ出力後サイズ計算を行いトラック開始直後にあるサイズ保管場所にシークし書き直す。サイズ書き込み後はファイル最後へ再シークする。

■ヘッダ(MThd)

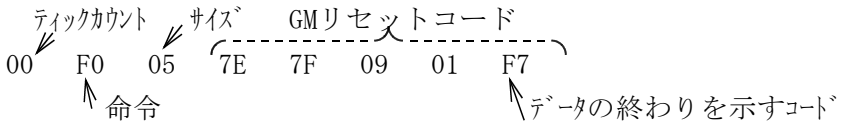


※FORMAT0は全DATAを１ﾄﾗｯｸに入れ、1はﾄﾗｯｸ別々、2は通常使用しない.

■トラック(MTrk)

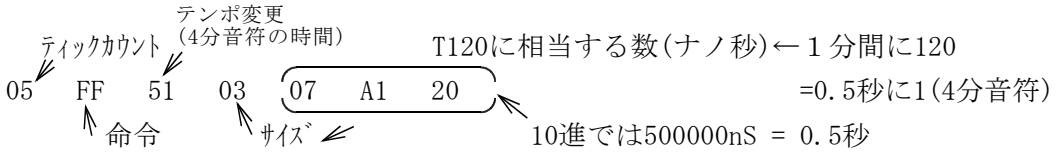


mdcompでは一番はじめのトラックにGMシステム・オンを入れる



また、ローランドのVSC-88についている説明書（オンライン）からGSリセットを実行後50ms以上の時間をあけなければならないとのこと。 注)GMシステム・オン時は不明です

さらに、GMシステム・オン後にはテンポも120に相当するスピードにするため以下のコードを実行する



テンポを計算する方法

$$\frac{T}{60}\times 1000000$$

例えば

T=120の時

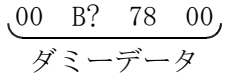
$$\frac{120}{60}\times 1000000$$

$= 0.5 \times 1000000$

$= 500000 = 07A120(16)$

■トラック

二番目以降に出現したトラックにはダミーデータを先頭に入れる

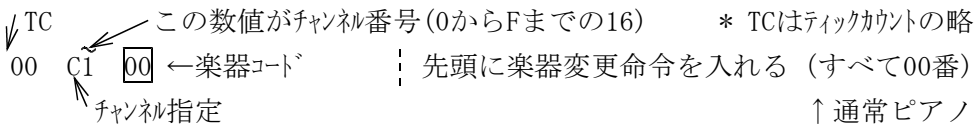


?にはチャンネル

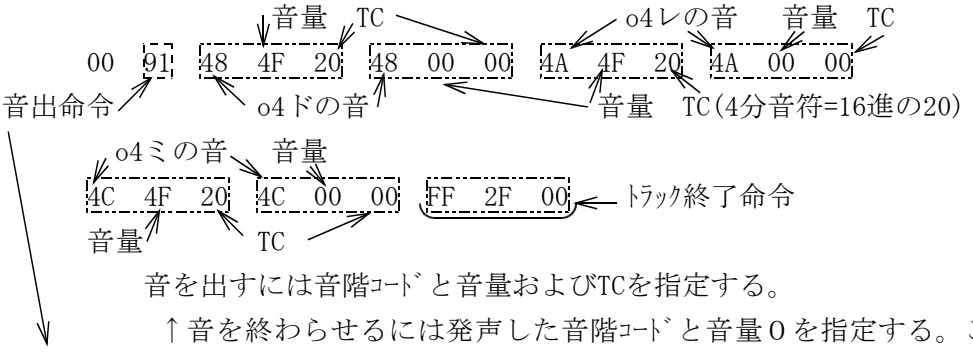
最初のトラック

初期化命令（GMシステム・オン、テンポ120)が完了したら

音階コードを入れる



音階ドレミを発声する時の例(４分音符)



音を出すには音階ｺｰﾄﾞと音量およびTCを指定する。

↑音を終わらせるには発声した音階ｺｰﾄﾞと音量０を指定する。これで１音。

音を出せという命令は9?となっており?にはチャンネル番号が入る。命令がつながる場合は、9?を先頭に１つおき、音階コードを連続してもよい。なおC?命令により楽器番号変更時はもう一度9?命令を出さないと音がそこで停止する。

■ティックカウント

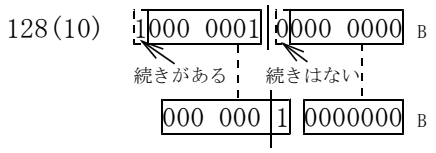
通常時間の流れを示すもので４分音符がティックカウント480と定義し、T=120=4分音符は0.5秒とすればティックカウント１は

$$\frac{0.5秒}{480} \times 1 \fallingdotseq 0.0010417秒となる$$

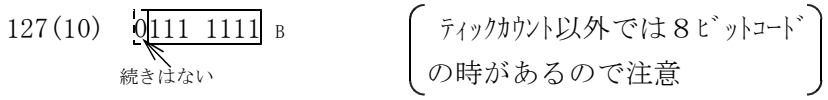
ティックカウントは7ビットコードで上位から保存する。
例えば、128(10)と127(10)の２つを７ビットコードで示すと

	8ﾋﾞｯﾄ ↓	7ﾋﾞｯﾄ	7ﾋﾞｯﾄ
128(10)	1000 0000B	0000001	0000000B
127(10)	0111 1111B		1111111B

であるが、SMFでは7ビットコードの最上位に8ビット目の続有無ビットを付加し8ビットとしている上記の128(10)をSMFの7ビットコードであらわすと



となるためティックカウントが128以上になるときは注意が必要(16384以上で同じようなことがまたおきる)



■技術情報

FMCOMPアルゴリズム(DOS版/16bitおよび32bit)

全面的にFMCOMPのソースコードを利用している
FMCOMPの仕様書を参照(YM-2203FM音源ドライバのファイル)

EDITキャンセラ

行番号指定でアルファベットや数字以外の文字混入をさけるために数字以外はキャンセルするようにした。

行番号指定アルゴリズム

memoコンポーネントの現在のカーソル位置を変更させるアルゴリズムを考案した。後にRichEditに変更

分母・余り交換アルゴリズム

連符で余りのティックカウントをうまく分配するアルゴリズムを考案した（連符参照）

注）RichEdit機能は1.00で削除しました。また、FM音源ドライバは非公開です。

DEC2ASCについて

10進数から５桁のアスキーコードに変換し指定番地に順に格納する。mdcompで使用した関数は16ビット値専用である.

※Special版のため99999までに改良してある

```
void __fastcall TForm1::Dec2Asc16(char *p,int i)
{
        unsigned int a,b,c;
        // special custom
        if(i>= 100000)
        {       int work1;
                work1 = i;
                work1 = work1 / 100000 * 100000;
                i = i-work1;  //line No. loop
        }
        a = (unsigned int)i;  //special custom
        c = 10000;   //max 99999
        for( i=0; i<5 ; i++)  // i< 5の5は最大桁数を示す
        {       b = (unsigned int)(a/c);
                a = (unsigned int)(a-c*b);
                b = (unsigned int)(b + '0');
                *p = (char)b;
                p++;
                c = (unsigned)(c/10);
        }
}
```

RD1LINEエミュレーションについて

ファイルから入力する関数をコンポーネントから入力する関数に書き換えた。

```
int __fastcall TForm1::rd1line()
{
        char *p;
        int c;
        p = (char *)rdwork;  /* work area addr */
        *p = 0;   /* 一番はじめを初期化する(誤動作防止) */
        int MaxLine = WncRichEdit1 -> Lines -> Count;  //行数を求める
        AnsiString AsWork1;  //work area
        if (RdFileLineCnt < MaxLine)
        {
                AsWork1 = WncRichEdit1->Lines->Strings[RdFileLineCnt];
                c = strlen(strncpy(p,AsWork1.c_str();LINEMAX-1));
                p[c] = 0x0d;  //改行コードを強制的に入力
                c++;          //改行コードも読み込んだ扱い
        }
        else
        {
                c = 0;
        }
        RdFileLineCnt++;  //次の行にする
        rd_linecount++;  //読み込み行数inc
        return(c);   /* no error */
}
//rdwork[]にﾃﾞｰﾀを保管、戻り値は保管したバイト数
```

EDITキャンセラ

数値以外の文字は入力していないとFormにおもわせる

```
void __fastcall TGotoLine::EditKeyPress(TObject *Sender,char &Key)
{
        if(Key >= '0' && Key <= '9') return;

        if(Key == VK_ESCAPE)return;
        if(Key == VK_BACK) return;
        if(Key == VK_DELETE) return;

        Key = NULL;  //なかったことにする
}
```

行番号指定アルゴリズム

RichEditの現在カーソルがある位置を指定位置へ

```
void __fastcall TForm1::N6Click(TObject *Sender)
{
        GotoLine->Edit1->SelectAll();
        int mrsult = GotoLine->ShowModal();
        WncRichEdit1->SetFocus();
        int jumpbyte = GotoLine -> Tag;
        if(jumpbyte >= 0 && mrsult == mrOk)
        {       WncRichEdit1->SelStart = jumpbyte;
        }
        else if(jumpbyte < 0 && mrsult == mrOk)
        {       int MaxLine = WncRichEdit1 ->Lines->Count;
                AnsiString Me1 = "指定した行番号はない(最大 ";
                Me1 += IntToStr(MaxLine +1)+"行)";
                Application->MessageBox(Me1.c_str(),NULL,MB_ICONSTOP);
        }
}
void __fastcall TGotoLine::BitBtn1Click(TObject *Sender)
{
        int i1;
        try
        {       if(Edit1->Text.Length()>0)
                {       i1 = StrToInt(Edit1->Text);}
                else{   ModalResult = mbCancel;}
        }
        catch(const Exception &E)
        {  Application->MessageBox("数値を入力してください",NULL,MB_ICONSTOP);}
        int jumpbyte; //何バイト先か格納
        int toY;      //goto飛び先格納用
        toY = i1;
        if(toY > 0)toY--;
        jumpbyte = Form1->WncRichEdit1->Perform(EM_LINEINDEX,toY,0);
        GotoLine->Tag=jumpbyte;   //飛ばすべきバイト数を格納(midmainで受取)
}
```

この部分で飛び先バイト数を得ている

偽RichEditコンポーネント

スクロールバーをリアルタイムに監視するため作成（WncRichEditとした）.
RichEditコンポーネントから派生したためRichEditの機能を全て含む。

コンポーネントのヘッダでメッセージを受け取る定義（プロテクト部で定義）

```
BEGIN_MESSAGE_MAP
        VCL_MESSAGE_HANDLER(WM_PAINT,TMessage,WMSize);
        VCL_MESSAGE_HANDLER(WM_SETCURSOR,TMessage,WMSize);
        VCL_MESSAGE_HANDLER(WM_KEYDOWN,TMessage,WMSize);
        VCL_MESSAGE_HANDLER(WM_KEYUP,TMessage,WMSize);
END_MESSAGE_MAP(TRichEdit);

void __fastcall TWncRichEdit::WMSize(Messages::TMessage &Message)
{
        HWND hand1;
        hand1 = GetParent(TWncRichEdit::Handle);

        SendMessage(hand1,WM_PAINT,NULL,NULL);
        TRichEdit::Dispatch(&Message);
}
```

注）RichEdit系は移植の都合上v1.00で削除しています

■エラー番号とメッセージ（v1.01）

1　解読できない命令があります
10　&は連続で並べられません
11　&を単独では利用できません
12　命令 N? の範囲は0～96
13　命令 O? の範囲は0～7
14　命令 L? で不可数値を使用
15　命令 @? の範囲は0～127
16　命令 V? の範囲は0～16
17　命令 T? の範囲は10～250
18　命令 :　の範囲は1～255
19　命令 Q? の範囲は1～8
20　命令 {} 連符指定に誤りがある
21　命令 $? の範囲は 0～127
22　op は ')' で終わる必要があります
23　op で 該当する制御命令はありません
24　cr は ')' で終わる必要があります
25　文字列は63Byteまで(超過違反)
26　sn は ')' で終わる必要があります
27　文字列は127Byteまで(超過違反)
999 chをオープンしていません

他　　その他のエラー発生

■将来実現したい機能

著作権設定命令(搭載済)

指定ﾁｬﾝﾈﾙのコンパイルを行わない命令

コンパイルしないﾁｬﾝﾈﾙ

案1)　nc { ch1,ch2,ch4}

↑

コンパイルを行わない指定

案2)　nc = 1,2,4

案3)　nc(1,2,4)

案4)　nc {1,2,4 }

SMFに含めるコメント命令(搭載済)

注) mdcomp32の開発は完了しており追加は予定していません

長さ途中経過出力命令

チャンネル
↙
案1)　ot { 1,2,4 }
↑
長さ出力命令

案2)　ot(1,2,4)

なお、途中経過はメッセージ枠に出力する

案1)　-> ch1(ot1:32 TrackLength)
-> ch2(ot1:64 TrackLength)
-> ch4(ot1:32 TrackLength)
↑otの数値は命令出現ごとにあがる.

案2)　-> ch1:32,ch2:64,ch4:32 TrackLength

案2-2) -> ot1(ch1:32,ch2:64,ch4:32)Track Length

コンパイルの途中終了

案1)　ce

案2)　!!

２文字にしているのはコンパイラのリミットを現在２文字に固定しているため将来多文字命令を使用する場合は以下も案としたい。

案3)　end

案4)　compstop

注）mdcomp32の開発は完了しており追加は予定していません

■v0.00.05以降で追加した機能

v0.00.05b　例外発生で起動していたのを変更し例外発生なしに起動
v0.00.05c　関連づけ起動に対応
v0.00.06　Panpot命令に対応($命令追加)
v0.00.07　自動左右配分機能追加
v0.00.08　ソースファイルからQ_DEFとAUTOLRを変更できる(op命令追加)

v0.00.07で追加した自動左右配分について

チェックボタンを押すと機能が作動する。
ch1:64(操作せず)さわらず
ch2:MIN_PAN (0)
ch3:MAX_PAN (127)
ch4:PAN_DEF / 3 (64/3)
ch5:PAN_DEF / 3 + PAN_DEF (64/3+64)
ch6:PAN_DEF / 3 * 2 (64/3*2)
ch7:PAN_DEF / 3 * 2 + PAN_DEF(64/3*2+64)
ch8:さわらず
ch9:さわらず

v0.00.08で追加した命令について

op(スイッチ１ , ...)
スイッチにはAUTOLR,MANULR,Q7DEF,Q8DEFが使用できる。
, (カンマ)で区切ればいくつでも指定できる。

v0.00.09,v0.00.10で追加した命令について

cr("著作権文字")
SMFに著作権表示文字列を組み込みます(最大63Byte mdcomp固有制限)

sn("文字列")
SMFにシーケンス名文字列を組み込みます(最大127Byte mdcomp固有制限)

注) v1.00,v1.01に命令の追加はありません

■mdcomp32.cfgの構造

v1.01での構造を示す.名称はmdcomp32.cfgで、サイズは 512 バイト.

+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+a	+b	+c	+d	+e	+f
M	D	C	O	M	P	0	0	0	CR	LF	EOF	(RESERVED)			

+10	+11	+12	+13	+14	+15	+16	+17	+18	+19	+1a	+1b	+1c	+1d	+1e	+1f
FL1	FL2	(フラグ用予約領域)													

+20	～	+2f
(予約領域)		

+30	～	+1ff
(予約領域)		(予約領域)

FL1の内容

+20 b7 Q7DEF
b6 文字 大
b5 文字 中
b4 文字 小
b3 最大化
b2 番号表示
b1 演奏LOOP
b0 自動保存

FL2の内容

+21 b7
b6
b5
b4
b3
b2
b1
} 予約
b0 AUTOLR

■fmcompデータフォーマット(FMC version 001)

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+a	+b	+c	+d	+e	+f
+0	'F'	'M'	'P'	'0'	'0'	'0'	'F'	'M'	'C'	'0'	'0'	'1'	ch*2	予約	開始位置	
+10	CH1		CH2		CH3		CH4		CH5		CH6		CH7		CH8	
+20	CH9		(CH10)		(CH11)		(CH12)		(CH13)		(CH14)		(CH15)		(CH16)	
+30	データ															

+0 ～ +5　　FMP000 文字(プレイヤー用識別文字)※ユーザーが変更しても良い

+6 ～ +b　　FMC001 文字(フォーマット識別文字)※各バージョンに互換性は無い

+c　　チャンネル×2byte(CH16として通常0x20が入る)

+d　　予約領域

+e +f　　データ開始位置(CH1のオフセット格納位置、通常0x10が入る)

+10 +11　　CH1データの開始オフセット(通常0x10を引いた値0x20が入る)

注意書きがない場合、複数バイトにまたがる数値はリトルエンディアンで格納

また、各値は8bitの場合0～255 , 16bitは0～65535までの符号無値

CH1のデータが先頭+30のところから始まっている場合、+10に0x20 , +11に00が入る。

■格納データ

データはアドレスの小さい側に1byteに機能・音階コード、2byte目に長さ・数値が入る。

例えば、04C4D8R4の音階(オクターブ4のド♩・レ♪・休符♩)時は49,32,51,16,0,32が入る

@音階コード表

	o0	o1	o2	o3	o4	o5	o6	o7
C	1	13	25	37	49	61	73	85
C#	2	14	26	38	50	62	74	86
D	3	15	27	39	51	63	75	87
D#	4	16	28	40	52	64	76	88
E	5	17	29	41	53	65	77	89
F	6	18	30	42	54	66	78	90
F#	7	19	31	43	55	67	79	91
G	8	20	32	44	56	68	80	92
G#	9	21	33	45	57	69	81	93
A	10	22	34	46	58	70	82	94
A#	11	23	35	47	59	71	83	95
B	12	24	36	48	60	72	84	96

※0番は無音コード

@長さ表

長さ	定義値	長さ	定義値
1.	192	16.	12
1	128	16	8
2.	96	32.	6
2	64	32	4
4.	48	64.	3
4	32	64	2
8.	24	128	1
8	16		

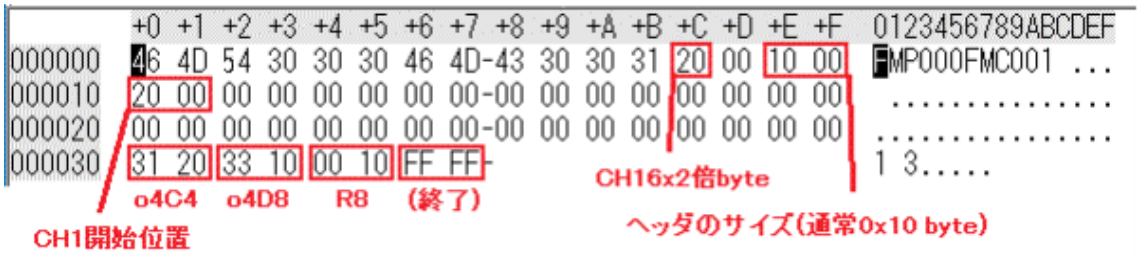
各チャンネルの終了は 255,255のペアでチャンネルデータの終了とする。

@機能コード v1.01

機能番	値	意味
255	255	終了コード
254	楽器No.	楽器変更 0～127(初期値=0)
253	音量	音量変更 0 ～ 16 (初期値=10)
252	テンポ	テンポ変更 10～250 (初期値=120)
251	0	&命令
250	割合	Q命令 1～8(初期値=8または7)
249	割合	音の左右振り分け(0～127)

0～96までは音階、97～248までは現在未使用(機能追加時は248から小へ)

mdcomp32から出力したデータをバイナリエディタで見た場合。
(ch1 { "o4c4d8r8" })



0x31 = 49
0x20 = 32
0x33 = 51
0x10 = 16
0xff = 255

+c には 0x20が入っているが +10から+2fまでのCH1～CH16までの16チャンネル分場所を用意している (実際に使用しているのはCH1～CH9まで)

各チャンネルをすべて合わせたバイトは64kバイトまでとする。
理論上65535を越えた演奏データを作成できるが、符号なし16bit値を考慮して65535を越えないようにデータを作成するのが望ましい。

□このテキストについて

著作権およびその他の権利はwnc / t.w.にありますが、自由に配布または複写および使用・利用できます。この技術解説書は無制限に複製・配布可能です（ご連絡は不要です)。引用する場合は mdcomp32(c)wnc / t.w. を使用箇所にいれていただければ引用量にかかわらずご連絡は不要です。いかなる損害や動作・修正を保証しません。すべて使用者の責任で使用してください。

Web頁(wnc inu goya)　http://hp.vector.co.jp/authors/VA055892/

都合により約束の内容が変更になる場合があります。また、予告なく掲載を中止または終了することがあります。

■著作・制作

2001/09/26(水),2013/02/24(日),2017/10/5(木)