m17n ライブラリ

1.5.5

作成： Doxygen 1.5.6

Thu Oct 15 14:12:49 2009

Contents

Chapter 1

m17n ライブラリ

1.1 m17n ライブラリとは?

m17n **ライブラリ** は C 言語用の多言語文書処理ライブラリです。

- 自由公開ソフトウェアです。
- GNU/Linux と Unix のアプリケーションやライブラリから利用できます。
- アプリケーションやライブラリのさまざまな側面で、多言語化を実現し ます。

"m17n" とは "multilingualization" の省略形です。
m17n ライブラリは多言語を扱うため、以下の機能を提供します。

- *M-text*: 多言語テキスト用のデータ構造。基本的には文字列であるが、 テキストプロパティと呼ばれる属性が付いており、C の文字列の代わりになる よう設計されている。m17n ライブラリで最も重要なオブジェクト。
- M-text を作ったり取り扱ったりするための関数。
- M-text と既存のフォーマットでコード化された文字列との間の変換を行 う関数。
- 巨大な文字空間。 Unicode 文字すべてとそれ以上の数の非 Unicode 文 字を含むことができる。
- **文字テーブル**: 文字毎の情報を効率的に保持するデータ構造。
- M-text をウィンドウシステム上で入力／表示する関数。

1.2 利用方法

<m17n.h> をプログラムに include し、-lm17n で m17n ライブラリ とリンクしてください。 **はじめに** (p. 5) を参照。

1.3 外部ライブラリ／データ

m17n ライブラリは以下の外部ライブラリを利用しています。必須ではありま せんが、m17n ライブラリの幾つかの関数はこれらに依存しています。

- m17n-db – http://www.m17n.org/m17n-lib-ja/download/m17n-db-1.5.0.tar.gz
  m17n ライブラリに種々の情報を提供します。
- libxml2 – http://xmlsoft.org/
  関数 **mtext_serialize()** (p. 56) と **mtext_deserialize()** (p. 57) が使います。 libxml2 が利用できない時には、これらの関数は NULL を返します。
- fribidi – http://fribidi.sourceforge.net/
  BIDI 処理に使います。利用できない時は、m17n ライブラリの表示エン ジンは Arabic や Hebrew などのスクリプトを正しく処理できません。
- freetype – http://www.freetype.org/
  ローカルフォントの処理に使います。
- fontconfig – http://nexp.cs.pdx.edu/fontconfig/
  Xft と共に、ローカルフォントの検索に使います。
- xft – http://freedesktop.org/Software/Xft
  fontconfig と共に X サーバの XRender 拡張を利用してテキストをロー カルフォントで表示するために使います。
- GD テキストをローカルフォントで bitmap/pixmap 上に表示するのに使いま す。
- libotf – http://www.m17n.org/libotf/
  freetype と共に OpenType フォントの処理に使います。
- anthy – http://anthy.sourceforge.jp/
  日本語入力メソッド ja-anthy.mim が使います。
- wordcut – http://thaiwordseg.sourceforge.net/
  プログラム例 example/linebreak.c 中でタイ語の語の境界を見つけるた めに使っています。

1.4 連絡先:

独立行政法人 産業技術総合研究所

情報技術研究部門

グローバル IT セキュリティグループ

Web: http://www.m17n.org/m17n-lib-ja/

バグレポート: m17n-lib-bug-XXXX@m17n.org (XXXX の部分には現在の年（西暦）を 4 桁で入れてください。)

メイリングリスト: http://www.m17n.org/m17n-lib-ja/mailinglist.htm

1.5 謝辞

Special thanks to:

- Dimitri van Heesch <dimitri@stack.nl>
  Author of Doxygen <http://www.stack.nl/~dimitri/doxygen/>. Without this tool, it would have been impossible to create this documentation.

- Information-technology Promotion Agency (IPA), Japan

  Writing this documentation was partially funded by Information-technology Promotion Agency (IPA) <http://www.ipa.go.jp/about/english/index.html> in fiscal year 2001.

Chapter 2

モジュール

2.1　はじめに

m17n ライブラリ イントロダクション.

マクロ定義

- #define **M17NLIB_MAJOR_VERSION**
- #define **M17NLIB_MINOR_VERSION**
- #define **M17NLIB_PATCH_LEVEL**
- #define **M17NLIB_VERSION_NAME**
- #define **M17N_INIT**()

  m17n **ライブラリを初期化する**.

- #define **M17N_FINI**()

  m17n **ライブラリを終了する**.

列挙型

- enum **M17NStatus** {
  M17N_NOT_INITIALIZED,
  M17N_CORE_INITIALIZED,
  M17N_SHELL_INITIALIZED,
  M17N_GUI_INITIALIZED }

  m17n **ライブラリの状態を示す列挙型**.

関数

- enum **M17NStatus m17n_status** (void)

  m17n **ライブラリのどの部分が初期化されたか報告する**.

2.1.1 説明

m17n ライブラリ イントロダクション.

***API* のレベル**

m17n ライブラリの API は以下の４種に分類されている。

1. コア API

   M-text を扱うための基本的なモジュールを提供する。 この API を利用するためには、アプリケーションプログラムは <m17n-core.h> を include し、 -lm17n-core でリンクされなくてはならない。

2. シェル API

   文字プロパティ、文字集合操作、コード変換等のためのモジュールを提供する。 これらのモジュールは、データベースから必要に応じて多様なデータをロードする。 この API を利用するためには、アプリケーションプログラムは <m17n.h> を include し、 -lm17n-core -lm17n でリンクされなくてはならない。

   この API を使用すれば、コア API も自動的に使用可能となる。

3. FLT API

   文字列表示に **フォントレイアウトテーブル** (p. 207) を用いるモジュールを提供する。この API を利用するためには、アプリケーションプログラムは <m17n.h> を include し、 -lm17n-core -lm17n-flt でリンクされなくてはならない。

   この API を使用すれば、コア API も自動的に使用可能となる。

4. GUI API

   グラフィックデバイス上で M-text を表示したり入力したりするための GUI モジュールを提供する。この API 自体はグラフィックデバイスとは独立であるが、 多くの関数は特定のグラフィックデバイス用に作成された MFrame を引数に取る。 現時点でサポートされているグラフィックデバイスは、ヌルデバイス、X ウィンドウシステム、および GD ライブラリのイメージデータ (gdImagePtr) である。

   ヌルデバイスのフレーム上では表示も入力もできない。ただし **mdraw_glyph_list()** (p. 144) などの関数は使用可能である。

   X ウィンドウシステムのフレーム上ではすべての GUI API が使用できる。

   GD ライブラリのフレーム上では、描画用の API はすべて使用できるが、入力はできない。

   この API を使用するためには、アプリケーションプログラムは <m17n-gui.h> を include し、-lm17n-core -lm17n -lm17n-gui でリンクされなくてはならない。

   この API を使用すれば、コア API、シェル API、および FLT API も自動的に使用可能となる。

5. その他の API

   エラー処理、デバッグ用のその他の関数を提供する。この API はそれだけでは使用できず、上記の他の API と共に使う。利用するためには、上記のいずれかの include ファイルに加えて、<m17n-misc.h> を include しなくてはならない。

m17n-config(1) (p. 193) 節も参照。

環境変数

m17n ライブラリは以下の環境変数を参照する。

- `M17NDIR`

  m17n データベースのデータを格納したディレクトリの名前。詳細は **データベース** (p. 59) 参照。

- `MDEBUG_XXX`

  "MDEBUG_" で始まる名前を持つ環境変数はデバッグ情報の出力を制御する。 詳細は **デバッグサポート** (p. 153) 参照。

API の命名規則

m17n ライブラリは、関数、変数、マクロ、型を export する。それらは 'm' または 'M' のあとにオブジェクト名 ("symbol"、"plist" など) またはモジュール名 (draw, input など) を続けたものである。 M-text オブジェクトの名前は "mmtext" ではなくて "mtext" で始まることに注意。

- 関数 – mobject() または mobject_xxx()

  'm' のあとに小文字でオブジェクト名が続く。単語間は '_' で区切られる。たとえば、msymbol(), **mtext_ref_char()** (p. 38), **mdraw_text()** (p. 141) など。

- シンボルでない変数 – mobject, または mobject_xxx

  関数と同じ命名規則に従う。たとえば mface_large など。

- シンボル変数 – Mname

  MSymbol 型変数は、'M' の後に名前が続く。単語間は '_' で区切られる。たとえば Mlanguage (名前は "language"), Miso_2022 (名前は"iso-2022") など。

- マクロ – MOBJECT_XXX

  'M' の後に大文字でオブジェクト名が続く。単語間は '_' で区切られる。

- タイプ – MObject または MObjectXxx

  'M' の後に大文字で始まるオブジェクト名が続く。単語は連続して書かれ、 '_' は用いられない。たとえば **MConverter** (p. 160), **MInputDriver** (p. 186) など。

2.1.2 マクロ定義

2.1.2.1 #define M17NLIB_MAJOR_VERSION

マクロ **M17NLIB_MAJOR_VERSION** (p. 7) は m17n ライブラリのメジャーバージョン番号を与える.

2.1.2.2 #define M17NLIB_MINOR_VERSION

マクロ **M17NLIB_MINOR_VERSION** (p. 7) は m17n ライブラリのマイナーバージョン番号を与える.

2.1.2.3 #define M17NLIB_PATCH_LEVEL

マクロ **M17NLIB_PATCH_LEVEL** (p. 7) は m17n ライブラリのパッチレベル番号を与える.

2.1.2.4 #define M17NLIB_VERSION_NAME

マクロ **M17NLIB_VERSION_NAME** (p. 7) は m17n ライブラリのバージョン名を文字列として与える.

2.1.2.5 #define M17N_INIT()

m17n ライブラリを初期化する.

マクロ **M17N_INIT()** (p. 7) は m17n ライブラリを初期化する。m17n の関数を利用する前に、このマクロをまず呼ばなくてはならない。

このマクロを複数回呼んでも安全であるが、その場合メモリを解放するためにマクロ **M17N_FINI()** (p. 8) を同じ回数呼ぶ必要がある。

外部変数 **merror_code** (p. 152) は、初期化が成功すれば 0 に、そうでなければ -1 に設定される。

参照:
M17N_FINI() (p. 8), **m17n_status()** (p. 8)

2.1.2.6 #define M17N_FINI()

m17n ライブラリを終了する.

マクロ **M17N_FINI()** (p. 8) は m17n ライブラリを終了する。m17n ライブラリが使った全てのメモリ領域は解放される。一度このマクロが呼ばれたら、マクロ **M17N_INIT()** (p. 7) が再度呼ばれるまで m17n 関数は使うべきでない。

マクロ **M17N_INIT()** (p. 7) が N 回呼ばれていた場合には、このマクロが N 回呼ばれて初めてメモリが解放される。

参照:
M17N_INIT() (p. 7), **m17n_status()** (p. 8)

2.1.3 列挙型

2.1.3.1 enum M17NStatus

m17n ライブラリの状態を示す列挙型.

列挙型 **M17NStatus** (p. 8) は関数 **m17n_status()** (p. 8) の戻り値として用いられる。

列挙型の値:
M17N_NOT_INITIALIZED
M17N_CORE_INITIALIZED
M17N_SHELL_INITIALIZED
M17N_GUI_INITIALIZED

2.1.4 関数

2.1.4.1 enum M17NStatus m17n_status (void)

m17n ライブラリのどの部分が初期化されたか報告する.

関数 **m17n_status()** (p. 8) は m17n ライブラリのどの部分が初期化されたかに応じて、以下の値のいずれかを返す。

M17N_NOT_INITIALIZED (p. 8), **M17N_CORE_INITIALIZED** (p. 8), **M17N_SHELL_INITIALIZED** (p. 8), **M17N_GUI_INITIALIZED** (p. 8)

2.2 コア API

libm17n-core.so が提供する API

モジュール

- **管理下オブジェクト**
  参照回数で管理されるオブジェクト
- **シンボル**
  シンボルオブジェクトとそれに関する *API.*
- **プロパティリスト**
  プロパティリストオブジェクトとそれに関する *API.*
- **文字**
  文字オブジェクトとそれに関する *API.*
- **文字テーブル**
  文字テーブルとそれに関する *API.*
- **M-text**
  M-text **オブジェクトとそれに関する** *API.*
- **テキストプロパティ**
  テキストプロパティを操作するための関数.
- **データベース**
  m17n **データベースにとそれに関する** *API.*

マクロ定義

- #define **M17N_FUNC**(func) ((**M17NFunc**) (func))
  汎関数型へのラッパ.

型定義

- typedef void(∗ **M17NFunc**)(void)
  汎関数型.

2.2.1 説明

libm17n-core.so が提供する API

2.2.2 マクロ定義

2.2.2.1 #define M17N_FUNC(func) ((M17NFunc) (func))

汎関数型へのラッパ.

マクロ **M17N_FUNC()** (p. 10) は関数を **M17NFunc** (p. 10) 型へキャストする。

2.2.3 型定義

2.2.3.1 typedef void(∗ M17NFunc)(void)

汎関数型.

M17NFunc (p. 10) は汎関数型であり、関数ポインタを **MSymbol** (p. 14) プロパティや **MPlist** (p. 19) の値として設定する際用いる。

参照:

msymbol_put_func() (p. 16), **msymbol_get_func()** (p. 16), **mplist_put_func()** (p. 21), **mplist_get_func()** (p. 21).

2.3 管理下オブジェクト

参照回数で管理されるオブジェクト

データ構造

- struct **M17NObjectHead**
  管理下オブジェクトの最初のメンバ.

関数

- void * **m17n_object** (int size, void(*freer)(void *))
  管理下オブジェクトを割り当てる.
- int **m17n_object_ref** (void *object)
  管理下オブジェクトの参照数を *1* 増やす.
- int **m17n_object_unref** (void *object)
  管理下オブジェクトの参照数を *1* 減らす.

2.3.1 説明

参照回数で管理されるオブジェクト

管理下オブジェクトとは参照数によって管理されているオブジェクトである.

m17n オブジェクトのある型のものは、参照数によって管理されている。 それらのオブジェクトは **管理下オブジェクト** と呼ばれる。生成された時点での参照数は 1 に初期化されている。関数 **m17n_object_ref()** (p. 12) は管理下オブジェクトの参照数を 1 増やし、関数 m17n_object_unref() は 1 減らす。参照数が 0 になった管理下オブジェクトは自動的に解放される。

キーが管理キーであるプロパティは、値として管理下オブジェクトだけを取る。 関数 **msymbol_put()** (p. 15) や **mplist_put()** (p. 20) などはそれらのプロパティを特別扱いする。

定義済み管理下オブジェクトタイプの他に、ユーザは必要な管理下オブジェクトタイプを自分で定義することができる。詳細は **m17n_object()** (p. 11) の説明を参照。

2.3.2 関数

2.3.2.1 void* m17n_object (int *size*, void(*)(void *) *freer*)

管理下オブジェクトを割り当てる.

関数 **m17n_object()** (p. 11) は **size** バイトの新しい管理下オブジェクトを割り当て、その参照数を 1 とする。 **freer** は参照数が 0 になった際にそのオブジェクトを解放するために用いられる関数である。**freer** が NULL ならば、オブジェクトは関数 free() によって解放される。

割り当てられたオブジェクト冒頭のバイトは、**M17NObjectHead** (p. 157) が占める。この領域は m17n ライブラリが使用するので、アプリケーションプログラムは触れてはならない。

戻り値:
この関数は新しく割り当てられたオブジェクトを返す。

エラー:
この関数は失敗しない。

例：

```
typedef struct
{
  M17NObjectHead head;
  int mem1;
  char *mem2;
} MYStruct;

void
my_freer (void *obj)
{
  free (((MYStruct *) obj)->mem2);
  free (obj);
}

void
my_func (MText *mt, MSymbol key, int num, char *str)
{
  MYStruct *st = m17n_object (sizeof (MYStruct), my_freer);

  st->mem1 = num;
  st->mem2 = strdup (str);
  /* KEY must be a managing key.   */
  mtext_put_prop (mt, 0, mtext_len (mt), key, st);
  /* This sets the reference count of ST back to 1.  */
  m17n_object_unref (st);
}
```

2.3.2.2 int m17n_object_ref (void * *object*)

管理下オブジェクトの参照数を 1 増やす.

関数 **m17n_object_ref()** (p. 12) は **object** で指される管理下オブジェクトの参照数を 1 増やす。

戻り値:
この関数は、増やした参照数が 16 ビットの符号無し整数値 (すなわち 0x10000 未満) におさまれば、それを返す。そうでなければ -1 を返す。

エラー:
この関数は失敗しない。

2.3.2.3 int m17n_object_unref (void * *object*)

管理下オブジェクトの参照数を 1 減らす.

関数 **m17n_object_unref()** (p. 12) は **object** で指される管理下オブジェクトの参照数を 1 減らす。参照数が 0 になれば、オブジェクトは解放関数によって解放される。

戻り値:
この関数は、減らした参照数が 16 ビットの符号無し整数値 (すなわち 0x10000 未満) におさまれば、それを返す。そうでなければ -1 を返す。つまり、0 が返って来た場合は **object** は解放されている。

エラー:
この関数は失敗しない。

2.4 シンボル

シンボルオブジェクトとそれに関する API.

型定義

- typedef struct MSymbolStruct * **MSymbol**
  シンボルの型宣言.

関数

- **MSymbol msymbol** (const char *name)
  シンボルを得る.
- **MSymbol msymbol_as_managing_key** (const char *name)
  管理キーを作る.
- int **msymbol_is_managing_key** (**MSymbol** symbol)
- **MSymbol msymbol_exist** (const char *name)
  指定された名前を持つシンボルを探す.
- char * **msymbol_name** (**MSymbol** symbol)
  シンボルの名前を得る.
- int **msymbol_put** (**MSymbol** symbol, **MSymbol** key, void *val)
  シンボルプロパティに値を設定する.
- void * **msymbol_get** (**MSymbol** symbol, **MSymbol** key)
  シンボルプロパティの値を得る.
- int **msymbol_put_func** (**MSymbol** symbol, **MSymbol** key, **M17NFunc** func)
  シンボルプロパティの値 *(***関数ポインタ***)* **を設定する**.
- **M17NFunc msymbol_get_func** (**MSymbol** symbol, **MSymbol** key)
  シンボルプロパティの値 *(***関数ポインタ***)* **を得る**.

変数

- **MSymbol Mnil**
  "nil" **を名前として持つシンボル**.
- **MSymbol Mt**
  "t" **を名前として持つシンボル**.
- **MSymbol Mstring**
  "string" **を名前として持つシンボル**.
- **MSymbol Msymbol**
  "symbol" **を名前として持つシンボル**.

2.4.1 説明

シンボルオブジェクトとそれに関する API.

m17n ライブラリは一意に決まる識別子として **シンボル** と呼ぶオブジェクトを用いる。シンボルは X ライブラリのアトムと似ているが、 0 個以上の **シンボルプロパティ** を持つことができる。シンボルプロパティは **キー** と **値** からなる。キーはそれ自体シンボルであり、値は `(void *)` 型にキャストできるものなら何でもよい。「シンボル S が持つシンボルプロパティのうちキーが K のもの」を簡単に「S の K プロパティ」と呼ぶことがある。

シンボルの用途は主に以下の 3 通りである。

- シンボルプロパティおよび他のプロパティのキーを表す。
- 文字セット、コード系、フォントセットなどの各種オブジェクトを表す。
- m17n ライブラリ関数の引数となり、関数の挙動を制御する。

管理キー と呼ばれる特別なシンボルがあり、管理キーをキーとして持つプロパティの値は **管理下オブジェクト** でなくてはならない。詳細は **管理下オブジェクト** (p. 11) 参照。

2.4.2 型定義

2.4.2.1 typedef struct MSymbolStruct* MSymbol

シンボルの型宣言.

MSymbol (p. 14) は **シンボル** (symbol) オブジェクトの型である。 内部構造はアプリケーションプログラムからは見えない。

2.4.3 関数

2.4.3.1 MSymbol msymbol (const char * *name*)

シンボルを得る.

関数 **msymbol()** (p. 14) は **name** という名前を持つ正規化されたシンボルを返す。そのようなシンボルが存在しない場合には、生成する。生成されたシンボルは管理キーではない。

空白文字二つで始まるシンボルは m17n ライブラリ用であり、内部的にのみ用いられる。

戻り値:
この関数は見つけたか生成したかしたシンボルを返す。

エラー:
この関数は決して失敗しない。

参照:
msymbol_as_managing_key() (p. 14), **msymbol_name()** (p. 15), **msymbol_exist()** (p. 15)

2.4.3.2 MSymbol msymbol_as_managing_key (const char * *name*)

管理キーを作る.

関数 **msymbol_as_managing_key()** (p. 14) は名前 **name** を持つ新しく作られた管理キーを返す。すでに名前 **name** を持つシンボルがあれば、 **Mnil** (p. 16) を返す。

空白文字二つで始まるシンボルは m17n ライブラリ用であり、内部的にのみ用いられる。

戻り値:
処理に成功すれば、この関数は生成したシンボルを返す。そうでなければ **Mnil** (p. 16) を返す。

エラー:
MERROR_SYMBOL

参照:
msymbol() (p. 14), **msymbol_exist()** (p. 15)

2.4.3.3 int msymbol_is_managing_key (MSymbol *symbol*)

2.4.3.4 MSymbol msymbol_exist (const char * *name*)

指定された名前を持つシンボルを探す.

関数 **msymbol_exist()** (p. 15) は **name** という名前を持つシンボルを探す。

戻り値:
もしそのようなシンボルが存在するならばそのシンボルを返す。そうでなければ、定義済みシンボル **Mnil** (p. 16) を返す。

エラー:
この関数は決して失敗しない。

参照:
msymbol_name() (p. 15), **msymbol()** (p. 14)

2.4.3.5 char* msymbol_name (MSymbol *symbol*)

シンボルの名前を得る.

関数 **msymbol_name()** (p. 15) は指定されたシンボル **symbol** の名前を含む文字列へのポインタを返す。

エラー:
この関数は決して失敗しない。

参照:
msymbol() (p. 14), **msymbol_exist()** (p. 15)

2.4.3.6 int msymbol_put (MSymbol *symbol*, MSymbol *key*, void * *val*)

シンボルプロパティに値を設定する.

関数 **msymbol_put()** (p. 15) は、シンボル **symbol** 中でキーが **key** であるシンボルプロパティの値を **val** に設定する。そのシンボルプロパティにすでに値があれば上書きする。 **symbol**, **key** とも **Mnil** (p. 16) であってはならない。

key が管理キーならば、**val** は管理下オブジェクトでなくてはならない。この場合、古い値の参照数は NULL でなければ 1 減らされ、**val** の参照数は 1 増やされる。

戻り値:
処理が成功すれば、msymbol_put() は 0 を返す。そうでなければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_SYMBOL

参照:
msymbol_get() (p. 16)

2.4.3.7 void* msymbol_get (MSymbol *symbol*, MSymbol *key*)

シンボルプロパティの値を得る.

関数 **msymbol_get()** (p. 16) は、シンボル **symbol** が持つシンボルプロパティのうち、キーが **key** であるものを探す。もし該当するシンボルプロパティが存在すれば、それの値を返す。そうでなければ NULL を返す。

戻り値:
エラーが検出された場合、msymbol_get() は NULL を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_SYMBOL

参照:
msymbol_put() (p. 15)

2.4.3.8 int msymbol_put_func (MSymbol *symbol*, MSymbol *key*, M17NFunc *func*)

シンボルプロパティの値 (関数ポインタ) を設定する.

関数 **msymbol_put_func()** (p. 16) は、関数 **msymbol_put()** (p. 15) と同様に、シンボル **symbol** のキーが **key** であるシンボルプロパティの値を設定する。但し その値は関数ポインタ **func** である。

参照:
msymbol_put() (p. 15), **M17N_FUNC()** (p. 10)

2.4.3.9 M17NFunc msymbol_get_func (MSymbol *symbol*, MSymbol *key*)

シンボルプロパティの値 (関数ポインタ) を得る.

関数 **msymbol_get_func()** (p. 16) は、関数 **msymbol_get()** (p. 16) と同様に、シンボル **symbol** が持つシンボルプロパティのうち、キーが **key** であるものを得る。但し その値は関数ポインタをである。

参照:
msymbol_get() (p. 16)

2.4.4 変数

2.4.4.1 MSymbol Mnil

"nil" を名前として持つシンボル.

シンボル **Mnil** (p. 16) は "nil" という名前を持ち、一般に「偽」または「否定」を意味する。 "int" に変換された場合、値は 0 である。 **Mnil** (p. 16) 自身はいかなるシンボルプロパティも持たない。

2.4.4.2 MSymbol Mt

"t" を名前として持つシンボル.

シンボル **Mt** (p. 17) は `"t"` という名前を持ち、一般に「真」または「肯定」を意味する。

2.4.4.3 MSymbol Mstring

"string" を名前として持つシンボル.

シンボル **Mstring** (p. 17) は `"string"` という名前を持ち、関数 **mchar_define_property()** (p. 25) などの引数として用いられる。

2.4.4.4 MSymbol Msymbol

"symbol" を名前として持つシンボル.

定義済みシンボル **Msymbol** (p. 17) は `"symbol"` という名前を持ち、関数 **mchar_define_property()** (p. 25) などの引数として使われる。

2.5 プロパティリスト

プロパティリストオブジェクトとそれに関する API.

型定義

- typedef struct **MPlist MPlist**
  プロパティリスト・オブジェクトの型宣言.

関数

- **MPlist** ∗ **mplist_deserialize** (**MText** ∗mt)
  M-text **をデシリアライズしてプロパティリストを作る**.
- **MPlist** ∗ **mplist** (void)
  プロパティリストオブジェクトを作る.
- **MPlist** ∗ **mplist_copy** (**MPlist** ∗plist)
  プロパティリストをコピーする.
- **MPlist** ∗ **mplist_put** (**MPlist** ∗plist, **MSymbol** key, void ∗val)
  プロパティリスト中のプロパティの値を設定する.
- void ∗ **mplist_get** (**MPlist** ∗plist, **MSymbol** key)
  プロパティリスト中のプロパティの値を得る.
- **MPlist** ∗ **mplist_put_func** (**MPlist** ∗plist, **MSymbol** key, **M17NFunc** func)
  プロパティリスト中のプロパティに関数ポインタである値を設定する.
- **M17NFunc mplist_get_func** (**MPlist** ∗plist, **MSymbol** key)
  プロパティリストからプロパティの関数ポインタである値を得る.
- **MPlist** ∗ **mplist_add** (**MPlist** ∗plist, **MSymbol** key, void ∗val)
  プロパティリスト末尾にプロパティを追加する.
- **MPlist** ∗ **mplist_push** (**MPlist** ∗plist, **MSymbol** key, void ∗val)
  プロパティリストの先頭にプロパティを挿入する.
- void ∗ **mplist_pop** (**MPlist** ∗plist)
  プロパティリストの先頭からプロパティを削除する.
- **MPlist** ∗ **mplist_find_by_key** (**MPlist** ∗plist, **MSymbol** key)
  プロパティリスト中から指定のキーを持つプロパティを探す.
- **MPlist** ∗ **mplist_find_by_value** (**MPlist** ∗plist, void ∗val)
  プロパティリスト中から指定の値を持つプロパティを探す.
- **MPlist** ∗ **mplist_next** (**MPlist** ∗plist)
  プロパティリストの次の部分リストを返す.

- **MPlist** ∗ **mplist_set** (**MPlist** ∗plist, **MSymbol** key, void ∗val)
  プロパティリストの最初のプロパティを設定する.

- int **mplist_length** (**MPlist** ∗plist)
  プロパティリストの長さを返す.

- **MSymbol mplist_key** (**MPlist** ∗plist)
  プロパティリスト中の最初のプロパティのキーを返す.

- void ∗ **mplist_value** (**MPlist** ∗plist)
  プロパティリスト中の最初のプロパティの値を返す.

変数

- **MSymbol Minteger**
  "integer" **を名前として持つシンボル**.

- **MSymbol Mplist**
  "plist" **を名前として持つシンボル**.

- **MSymbol Mtext**
  "mtext" **を名前として持つシンボル**.

2.5.1 説明

プロパティリストオブジェクトとそれに関する API.

プロパティリスト (または *plist*) は 0 個以上のプロパティのリストである。プロパティは **キー** と **値** からなる。キーはシンボルであり、値は `(void *)` にキャストできるものならば何でも良い。

あるプロパティのキーが **管理キー** ならば、その **値** は **管理下 オブジェクト** である。プロパティリスト自体も管理下オブジェクトである。

2.5.2 型定義

2.5.2.1 typedef struct MPlist MPlist

プロパティリスト・オブジェクトの型宣言.

MPlist (p. 19) は **プロパティリスト** (Property list) オブジェクトの型である。 内部構造はアプリケーションプログラムからは見えない。

2.5.3 関数

2.5.3.1 MPlist ∗ mplist_deserialize (MText ∗ *mt*)

M-text をデシリアライズしてプロパティリストを作る.

関数 **mplist_deserialize()** (p. 19) は M-text **mt** を解析してプロパティリストを返す。

mt のシンタックスは以下の通り。

MT ::= ’(’ ELEMENT ∗ ’)’

ELEMENT ::= SYMBOL | INTEGER | M-TEXT | PLIST

SYMBOL ::= アスキー文字列

INTEGER ::= ’-’ ? [’0’ | .. | ’9’]+ | ’0x’ [’0’ | .. | ’9’ | ’A’ | .. | ’F’ | ’a’ | .. | ’f’]+

M-TEXT ::= ’"’ character-sequence ’"’

`ELEMENT` の各選択肢はキー：`Msymbol`, `Minteger`, `Mtext`, `Mplist` のいずれかを割り当てられている。

アスキー文字列内では、バックスラッシュ (\) がエスケープ文字として用いられる。たとえば `abc\ def` は 4 文字目が空白文字であり長さが 7 である持つ名前を持つシンボルを生成する。

2.5.3.2 MPlist∗ mplist (void)

プロパティリストオブジェクトを作る.

関数 **mplist()** (p. 20) は長さ 0 のプロパティリストオブジェクトを新しく作って返す。

戻り値:
この関数は新しく作られたプロパティリストオブジェクトを返す。

エラー:
この関数は決して失敗しない。

2.5.3.3 MPlist∗ mplist_copy (MPlist ∗ *plist*)

プロパティリストをコピーする.

関数 **mplist_copy()** (p. 20) はプロパティリスト **plist** をコピーする。コピーのすべての値はコピー元 **plist** の値と同じである。

戻り値:
この関数は新しく作られた、**plist** のコピーであるプロパティリストを返す。

エラー:
この関数は決して失敗しない。

2.5.3.4 MPlist∗ mplist_put (MPlist ∗ *plist*, MSymbol *key*, void ∗ *val*)

プロパティリスト中のプロパティの値を設定する.

関数 **mplist_put()** (p. 20) はプロパティリスト **plist** を始めから探して、キーが **key** であるプロパティを見つける。見つかれば、その値を **value** に変更する。見つからなければ、キーが **key** で値が **value** である新しいプロパティが **plist** の末尾に追加される。**key** と **val** に対する制限については、mplist_add() の説明を参照。

key が管理キーならば、 **val** は管理下オブジェクトでなくてはならない。この場合、古い値の参照数は `NULL` でなければ 1 減らされ、**val** の参照数は 1 増やされる。

戻り値:
処理が成功すれば **mplist_put()** (p. 20) は変更されたか追加された要素から始まる **plist** の部分リストを返す。そうでなければ `NULL` を返す。

2.5.3.5 void* mplist_get (MPlist * *plist*, MSymbol *key*)

プロパティリスト中のプロパティの値を得る.

関数 **mplist_get()** (p. 21) は、プロパティリスト **plist** を始めから探して、キー が **key** であるプロパティを見つける。見つかれば、その値を `(void *)` 型で返す。見つからなければ `NULL` を返す。

`NULL` が返った際には二つの可能性がある: 上記のようにプロパティが見つからなかった場合と、プロパティが見つかり、その値が `NULL` である場合である。これらを区別する必要がある場合には関数 **mplist_find_by_key()** (p. 22) を使うこと。

参照:
mplist_find_by_key() (p. 22)

2.5.3.6 MPlist* mplist_put_func (MPlist * *plist*, MSymbol *key*, M17NFunc *func*)

プロパティリスト中のプロパティに関数ポインタである値を設定する.

関数 **mplist_put_func()** (p. 21) は関数 **mplist_put()** (p. 20) 同様、プロパティリスト **plist** 中でキーが **key** であるプロパティに値を設定する。但しその値は関数ポインタ **func** である。**key** は管理キーであってはならない。

参照:
mplist_put() (p. 20), **M17N_FUNC()** (p. 10)

2.5.3.7 M17NFunc mplist_get_func (MPlist * *plist*, MSymbol *key*)

プロパティリストからプロパティの関数ポインタである値を得る.

関数 **mplist_get_func()** (p. 21) は関数 **mplist_get()** (p. 21) と同様に、プロパティリ スト **plist** 中でキーが **key** であるプロパティの値、但し関数ポインタ、 を得る。

参照:
mplist_get() (p. 21)

2.5.3.8 MPlist* mplist_add (MPlist * *plist*, MSymbol *key*, void * *val*)

プロパティリスト末尾にプロパティを追加する.

関数 **mplist_add()** (p. 21) は、プロパティリスト **plist** の末尾にキーが **key** で値が **val** であるプロパティを追加する。**key** は、`Mnil` 以外の任意のシンボルでよい。

key が管理キーならば、**val** は管理下オブジェクトでなくてはならない。この場合、 **val** の参照数は 1 増やされる。

戻り値:
処理が成功すれば **mplist_add()** (p. 21) は追加された要素から始まる **plist** の部分リストを返す。そうでなければ `NULL` を返す。

2.5.3.9 MPlist* mplist_push (MPlist * *plist*, MSymbol *key*, void * *val*)

プロパティリストの先頭にプロパティを挿入する.

関数 **mplist_push()** (p. 22) はプロパティリスト **plist** の先頭にキーが **key** で値が **val** であるオブジェクトを挿入する。

key が管理キーならば、**val** は管理下オブジェクトでなくてはならない。この場合、 **val** の参照数は 1 増やされる。

戻り値:
処理が成功すればこの関数は **plist** を返し、そうでなければ NULL を返す。

2.5.3.10 void* mplist_pop (MPlist * *plist*)

プロパティリストの先頭からプロパティを削除する.

関数 **mplist_pop()** (p. 22) はプロパティリスト **plist** の先頭のプロパティを削 除する。結果として、元の 2 番目のキーと値が先頭のキーと値になる。

戻り値:
処理に成功すれば、この関数は削除されたプロパティの値を返す。そうでなければ NULL を返す。

2.5.3.11 MPlist* mplist_find_by_key (MPlist * *plist*, MSymbol *key*)

プロパティリスト中から指定のキーを持つプロパティを探す.

関数 **mplist_find_by_key()** (p. 22) はプロパティリスト **plist** を始めから探 して、キーが **key** であるプロパティを見つける。見つかれば、そのプロパティから始まる **plist** の部分リストを返す。そうでなければ NULL を返す。

key が Mnil ならば、**plist** の最後の要素から始まる部分リストを返す。

2.5.3.12 MPlist* mplist_find_by_value (MPlist * *plist*, void * *val*)

プロパティリスト中から指定の値を持つプロパティを探す.

関数 **mplist_find_by_value()** (p. 22) はプロパティリスト **plist** を始めから探して、値が **val** であるプロパティを見つける。見つかれば、そのプロパティから始まる **plist** の部分リストを返す。そうでなければ NULL を返す。

2.5.3.13 MPlist* mplist_next (MPlist * *plist*)

プロパティリストの次の部分リストを返す.

関数 **mplist_next()** (p. 22) はプロパティリスト **plist** の 2 番目の要素から始まる部分リストへのポインタを返す。**plist** の長さが 0 ならば NULL を返す。

2.5.3.14 MPlist* mplist_set (MPlist * *plist*, MSymbol *key*, void * *val*)

プロパティリストの最初のプロパティを設定する.

関数 **mplist_set()** (p. 22) はプロパティリスト **plist** の最初のプロパティのキーと値をそれぞれ **key** と **value** に設定する。 **key** と **val** に対する制限については、mplist_add() の説明を参照。

戻り値:
処理に成功すれば **mplist_set()** (p. 22) は **plist** を返す。そうでなければ NULL を返す。

2.5.3.15 int mplist_length (MPlist * *plist*)

プロパティリストの長さを返す.

関数 **mplist_length()** (p. 23) はプロパティリスト **plist** 中のプロパティの数を返す。

2.5.3.16 MSymbol mplist_key (MPlist * *plist*)

プロパティリスト中の最初のプロパティのキーを返す.

関数 **mplist_key()** (p. 23) は、プロパティリスト **plist** 中の最初のプロパティのキーを返す。**plist** の長さが 0 ならば、 Mnil を返す。

2.5.3.17 void* mplist_value (MPlist * *plist*)

プロパティリスト中の最初のプロパティの値を返す.

関数 **mplist_value()** (p. 23) は、プロパティリスト **plist** 中の最初のプロパティの値を返す。 **plist** の長さが 0 ならば、 Mnil を返す。

2.5.4 変数

2.5.4.1 MSymbol Minteger

"integer" を名前として持つシンボル.

シンボル Minteger は "integer" という名前を持つ。キーが Minteger であるプロパティの値は整数値でなくてはならない。

2.5.4.2 MSymbol Mplist

"plist" を名前として持つシンボル.

シンボル Mplist は "plist" という名前を持つ。これは管理キーである。キーが Mplist であるプロパティの値は plist でなくてはならない。

2.5.4.3 MSymbol Mtext

"mtext" を名前として持つシンボル.

シンボル Mtext は "mtext" という名前を持つ管理キーである。キーが Mtext であるプロパティの値は M-text でなくてはならない。

2.6 文字

文字オブジェクトとそれに関する API.

変数: 文字プロパティのキー

これらのシンボルは文字プロパティのキーとして使われる。

- **MSymbol Mscript**
  スクリプトを表わすキー.
- **MSymbol Mname**
  名前を表わすキー.
- **MSymbol Mcategory**
  一般カテゴリを表わすキー.
- **MSymbol Mcombining_class**
  標準結合クラスを表わすキー.
- **MSymbol Mbidi_category**
  双方向カテゴリを表わすキー.
- **MSymbol Msimple_case_folding**
  対応する小文字一文字を表わすキー.
- **MSymbol Mcomplicated_case_folding**
  対応する小文字の列を表わすキー.
- **MSymbol Mcased**
  Case **処理に用いられる値のキー**.
- **MSymbol Msoft_dotted**
  Case **処理に用いられる値のキー**.
- **MSymbol Mcase_mapping**
  Case **処理に用いられる値のキー**.
- **MSymbol Mblock**
  スクリプトブロック名を表すキー.

マクロ定義

- #define **MCHAR_MAX**
  文字コードの最大値.

関数

- **MSymbol mchar_define_property** (const char ∗name, **MSymbol** type)
  文字プロパティを定義する.
- void ∗ **mchar_get_prop** (int c, **MSymbol** key)
  文字プロパティの値を得る.
- int **mchar_put_prop** (int c, **MSymbol** key, void ∗val)
  文字プロパティの値を設定する.
- **MCharTable** ∗ **mchar_get_prop_table** (**MSymbol** key, **MSymbol** ∗type)
  文字プロパティの文字テーブルを得る.

2.6.1 説明

文字オブジェクトとそれに関する API.

m17n ライブラリは **文字** を文字コード（整数）で表現する。 最小の文字コードは `0` であり、最大の文字コードはマクロ **MCHAR_MAX** (p. 25) によって定義されている。**MCHAR_MAX** (p. 25) は `0x3FFFFF`（22 ビット） 以上であることが保証されている。

`0` から `0x10FFFF` までの文字は、それと同じ値を持つ Unicode の文字に割り当てられている。

各文字は **文字プロパティ** と呼ぶプロパティを 0 個以上持つことができる。 文字プロパティは **キー** と **値** からなる。 キーはシンボルであり、値は `(void *)` 型にキャストできるものなら何でもよい。「文字 C の文字プロパティのうちキーが K であるもの」を簡単に「文字 C の K プロパティ」と呼ぶことがある。

2.6.2 マクロ定義

2.6.2.1 #define MCHAR_MAX

文字コードの最大値.

マクロ **MCHAR_MAX** (p. 25) は文字コードの最大値を表す。

2.6.3 関数

2.6.3.1 MSymbol mchar_define_property (const char ∗ *name*, MSymbol *type*)

文字プロパティを定義する.

関数 **mchar_define_property()** (p. 25) は、 <**Mchar_table** (p. 32), **type**, **sym** > というタグを持ったデータベースを m17n 言語情報ベースから探す。 ここで **sym** は **name** という名前のシンボルである。**type** は **Mstring** (p. 17), **Mtext** (p. 23), **Msymbol** (p. 17), **Minteger** (p. 23), **Mplist** (p. 23) のいずれかでなければならない。

戻り値:
処理に成功すれば **mchar_define_property()** (p. 25) は **sym** を返す。 失敗した場合は **Mnil** (p. 16) を返す。

エラー:
`MERROR_DB`

参照:
mchar_get_prop() (p. 26), **mchar_put_prop()** (p. 26)

2.6.3.2 void* mchar_get_prop (int *c*, MSymbol *key*)

文字プロパティの値を得る.

関数 **mchar_get_prop()** (p. 26) は、文字 **c** の文字プロパティのうちキーが **key** であるものを探す。

戻り値:
処理が成功すれば **mchar_get_prop()** (p. 26) は見つかったプロパティの値を返す。 失敗した場合は NULL を返す。

エラー:
MERROR_SYMBOL, MERROR_DB

参照:
mchar_define_property() (p. 25), **mchar_put_prop()** (p. 26)

2.6.3.3 int mchar_put_prop (int *c*, MSymbol *key*, void * *val*)

文字プロパティの値を設定する.

関数 **mchar_put_prop()** (p. 26) は、文字 **c** の文字プロパティのうちキーが **key** であるものを探し、その値として **val** を設定する。

戻り値:
処理が成功すれば **mchar_put_prop()** (p. 26) は 0 を返す。失敗した場合は-1 を返す。

エラー:
MERROR_SYMBOL, MERROR_DB

参照:
mchar_define_property() (p. 25), **mchar_get_prop()** (p. 26)

2.6.3.4 MCharTable* mchar_get_prop_table (MSymbol *key*, MSymbol * *type*)

文字プロパティの文字テーブルを得る.

関数 **mchar_get_prop_table()** (p. 26) は、キーが **key** である文字プロパティ を含む文字テーブルを返す。もし **type** が NULL でなければ、 **type** で 指される場所にその文字のプロパティを格納する。文字プロパティの種類 に関しては **mchar_define_property()** (p. 25) を見よ。

戻り値:
もし **key** が正当な文字プロパティのキーであれば、文字テーブルが返さ れる。そうでない場合は NULL が返される。

2.6.4 変数

2.6.4.1 MSymbol Mscript

スクリプトを表わすキー.

シンボル **Mscript** (p. 26) は "script" という名前を持ち、文字プロパティのキーとして使われる。 このプロパティの値は、この文字の属するスクリプトを表わすシンボルである。

スクリプトを表わすシンボルの名前は、*Unicode Technical Report #24* にリストされているもののいずれかである。

2.6.4.2 MSymbol Mname

名前を表わすキー.

シンボル **Mname** (p. 27) は "name" という名前を持ち、文字プロパティのキーとして使われる。 このプロパティの値はその文字の名前を表わす C の文字列である。

2.6.4.3 MSymbol Mcategory

一般カテゴリを表わすキー.

シンボル **Mcategory** (p. 27) は "category" という名前を持ち、文字プロパティのキーとして使われる。 このプロパティの値は、対応する **一般カテゴリ** を表わすシンボルである。

一般カテゴリを表わすシンボルの名前は、*General Category* の省略形として Unicode に定義されているものである。

2.6.4.4 MSymbol Mcombining_class

標準結合クラスを表わすキー.

シンボル **Mcombining_class** (p. 27) は "combining-class" という名前を持ち、文字プロパティのキーとして使われる。 このプロパティの値は、対応する **標準結合クラス** を表わす整数である。

標準結合クラスを表わす整数の意味は、Unicode に定義されているものと同じである。

2.6.4.5 MSymbol Mbidi_category

双方向カテゴリを表わすキー.

シンボル **Mbidi_category** (p. 27) は "bidi-category" という名前を持ち、文字プロパティのキーとして使われる。 このプロパティの値は、対応する **双方向カテゴリ** を表わすシンボルである。

双方向カテゴリを表わすシンボルの名前は、*Bidirectional Category* の型として Unicode に定義されているものである。

2.6.4.6 MSymbol Msimple_case_folding

対応する小文字一文字を表わすキー.

シンボル **Msimple_case_folding** (p. 27) は "simple-case-folding" という名前を持ち、文字プロパティのキーとして使われる。 このプロパティの値は、対応する小文字一文字であり、大文字／小文字の区別を無視した文字列比較の際に使われる。

複雑な比較方法を必要とする文字であった場合 （別の一文字と対応付けることによって比較できない場合)、このプロパティの値は 0xFFFF になる。この場合その文字は、**Mcomplicated_case_folding** (p. 27) というキーの文字プロパティを持つ。

2.6.4.7 MSymbol Mcomplicated_case_folding

対応する小文字の列を表わすキー.

シンボル **Mcomplicated_case_folding** (p. 27) は "complicated-case-folding" という名前を持ち、文字プロパティのキーとして使われる。 このプロパティの値は、対応する小文字列からなる M-text であり、大文字／小文字の区別を無視した文字列比較の際に使 われる。

2.6.4.8 MSymbol Mcased

Case 処理に用いられる値のキー.

シンボル **Mcased** (p. 28) は、"cased" という名前を持ち、文字プロパ ティのキーとして使われる。このプロパティの値は整数値 1, 2, 3 のいず れかであり、それぞれ "cased", "case-ignorable", その両方を意味する。 詳細については、the Unicode Standard 5.0 (Section 3.13 Default Case Algorithm) 参照。

2.6.4.9 MSymbol Msoft_dotted

Case 処理に用いられる値のキー.

シンボル **Msoft_dotted** (p. 28) は、"soft-dotted" という名前を持ち、 文字プロパティのキーとして使われる。このプロパティの値は、文字が "Soft_Dotted"プロパティを持つ場合には **Mt** (p. 17), そうでなければ **Mnil** (p. 16) で ある。 詳細については、the Unicode Standard 5.0 (Section 3.13 Default Case Algorithm) 参照。

2.6.4.10 MSymbol Mcase_mapping

Case 処理に用いられる値のキー.

シンボル **Mcase_mapping** (p. 28) は、"case-mapping" という名前をも ち、文字プロパティのキーとして使われる。このプロパティの値は、３つ の M-text、すなわちその文字の lower, title, と upper からなる plist である。 詳細については、the Unicode Standard 5.0 (Section 3.13 Default Case Algorithm) 参照。

2.6.4.11 MSymbol Mblock

スクリプトブロック名を表すキー.

シンボル **Mblock** (p. 28) は、"block" という名前をもち、文字プロパ ティのキーとして使われる。このプロパティの値は、その文字のスクリプ トブロック名を表すシンボルである。

2.7 文字テーブル

文字テーブルとそれに関する API.

型定義

- typedef struct **MCharTable MCharTable**
  文字テーブルの型宣言.

関数

- **MCharTable** ∗ **mchartable** (**MSymbol** key, void ∗default_value)
  新しい文字テーブルを作る.
- int **mchartable_min_char** (**MCharTable** ∗table)
- int **mchartable_max_char** (**MCharTable** ∗table)
- void ∗ **mchartable_lookup** (**MCharTable** ∗table, int c)
  文字テーブル中で文字に割り当てられた値を返す.
- int **mchartable_set** (**MCharTable** ∗table, int c, void ∗val)
  文字テーブル中での文字の値を設定する.
- int **mchartable_set_range** (**MCharTable** ∗table, int from, int to, void ∗val)
  指定範囲の文字に値を設定する.
- void **mchartable_range** (**MCharTable** ∗table, int ∗from, int ∗to)
  値がデフォルトと異なる文字を探す.
- int **mchartable_map** (**MCharTable** ∗table, void ∗ignore, void(∗func)(int, int, void ∗, void ∗), void ∗func_arg)
  文字テーブル中の文字に対して指定の関数を呼ぶ.

変数

- **MSymbol Mchar_table**
  "char-table" **という名前を持つシンボル**.

2.7.1 説明

文字テーブルとそれに関する API.

m17n ライブラリが扱う文字の空間は広大であるため、文字毎の情報を単純な配列に格納しようとすると、その配列は巨大になりすぎ、非実用的である。 しかし通常必要となる文字についての情報は、ある特定の範囲の文字にのみ付いていることが多い。 全文字に関して情報がある場合にも、連続した文字コードを持つ文字は同じ情報を持つことが多い。

このような傾向を利用して文字とその付加情報を効率的に格納するために、 m17n ライブラリは **文字テーブル** (chartable) と呼ぶオブジェクトを用いる。 文字テーブルは配列ではないが、アプリケーションプログ

ラムは文字テーブルを配列の一種として扱うことができる。 ある文字についての特定の情報は、その情報を持つ文字テーブルをその文字のコードで引くこと で得られる。

文字テーブルは管理下オブジェクトである。

2.7.2 型定義

2.7.2.1 typedef struct MCharTable MCharTable

文字テーブルの型宣言.

MCharTable (p. 30) は **文字テーブル** (chartable) オブジェクトの型である。 内部構造はアプリケーションプログラムからは見えない。

2.7.3 関数

2.7.3.1 MCharTable* mchartable (MSymbol *key*, void * *default_value*)

新しい文字テーブルを作る.

関数 **mchartable()** (p. 30) はキーが **key** で要素のデフォルト値が **default_value** である新しい文字テーブルを作る。もし **key** が管理キーであれば、このテーブルの要素は（デフォルト値を含めて）管理下オブジェクトか NULL のいずれかである。

戻り値:

処理が成功すれば **mchartable()** (p. 30) は作成された文字テーブルへのポインタを返す。 失敗した場合は `NULL` を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

2.7.3.2 int mchartable_min_char (MCharTable * *table*)

2.7.3.3 int mchartable_max_char (MCharTable * *table*)

2.7.3.4 void* mchartable_lookup (MCharTable * *table*, int *c*)

文字テーブル中で文字に割り当てられた値を返す.

関数 **mchartable_lookup()** (p. 30) は文字テーブル **table** 中で文字 **c** に割り当てられた値を返す。**c** に対する明示的な値がなければ、**table** のデフォルト値を返す。**c** が妥当な文字でなければ、mchartable_lookup() は `NULL` を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:

`MERROR_CHAR`

参照:

mchartable_set() (p. 30)

2.7.3.5 int mchartable_set (MCharTable * *table*, int *c*, void * *val*)

文字テーブル中での文字の値を設定する.

関数 **mchartable_set()** (p. 30) は、文字テーブル **table** 中の文字 **c** に値 **val** を割り当てる。

戻り値:
処理が成功すれば、mchartable_set() は 0 を返す。そうでなければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_CHAR

参照:
mchartable_lookup() (p. 30), **mchartable_set_range()** (p. 31)

2.7.3.6 int mchartable_set_range (MCharTable * *table*, int *from*, int *to*, void * *val*)

指定範囲の文字に値を設定する.

関数 **mchartable_set_range()** (p. 31) は、文字テーブル **table** 中の **from** から **to** まで（両端を含む）の文字に、値として **val** を設定する。

戻り値:
処理が成功すれば **mchartable_set_range()** (p. 31) は 0 を返す。そうでなければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。**from** が **to** より大きいときには、**mchartable_set_range()** (p. 31) は何もせず、エラーも起こさない。

エラー:
MERROR_CHAR

参照:
mchartable_set() (p. 30)

2.7.3.7 void mchartable_range (MCharTable * *table*, int * *from*, int * *to*)

値がデフォルトと異なる文字を探す.

関数 **mchartable_range()** (p. 31) は文字テーブル **table** 中で、**table** のデフォルト値以外の値を持つ最初と最後の文字を探し、それぞれを **from** と **to** に設定する。すべての文字が値としてデフォルト値をとっている場合には **from** と **to** を -1 に設定する。

2.7.3.8 int mchartable_map (MCharTable * *table*, void * *ignore*, void(*)(int, int, void *, void *) *func*, void * *func_arg*)

文字テーブル中の文字に対して指定の関数を呼ぶ.

関数 **mchartable_map()** (p. 31) は、文字テーブル **table** 中の文字に対して関数 **func** を呼ぶ。ただし **table** 中でも値が **ignore** である文字については関数呼び出しを行なわない。**ignore** と文字の値の比較は == で行なうので、文字列リテラルやポインタを使う際には注意を要する。

mchartable_map() (p. 31) は、一文字ごとに **func** を呼ぶのではなく、関数呼び出しの回数を最適化しようとする。 すなわち、連続した文字が同じ値を持っていた場合には、その文字のまとまり全体について一度の関数呼び出 ししか行なわない。

文字のまとまりの大きさにかかわらず、**func** は **from**, **to**, **val**, **arg** の４引数で呼ばれる。**from** と **to**（両端を含む）は **val** を値として持つ文字の範囲を示し、**arg** は **func_arg** そのものである。

戻り値:
この関数は常に 0 を返す。

2.7.4 変数

2.7.4.1 MSymbol Mchar_table

"char-table" という名前を持つシンボル.

シンボル `Mchar_table` は名前 `"char-table"` を持つ。

2.8 M-text

M-text オブジェクトとそれに関する API.

変数: UTF-16 と UTF-32 のデフォルトのエンディアン

- enum **MTextFormat MTEXT_FORMAT_UTF_16**
  値が *MTEXT_FORMAT_UTF_16LE* **か** *MTEXT_FORMAT_UTF_16BE* **である変数**

- const int **MTEXT_FORMAT_UTF_32**
  値が *MTEXT_FORMAT_UTF_32LE* **か** *MTEXT_FORMAT_UTF_32BE* **である変数**

型定義

- typedef struct **MText MText**
  MText **の型宣言**.

列挙型

- enum **MTextFormat** {
  MTEXT_FORMAT_US_ASCII,
  MTEXT_FORMAT_UTF_8,
  MTEXT_FORMAT_UTF_16LE,
  MTEXT_FORMAT_UTF_16BE,
  MTEXT_FORMAT_UTF_32LE,
  MTEXT_FORMAT_UTF_32BE,
  MTEXT_FORMAT_MAX }
  M-text **のフォーマットを指定する列挙型**.

- enum **MTextLineBreakOption** {
  MTEXT_LBO_SP_CM = 1,
  MTEXT_LBO_KOREAN_SP = 2,
  MTEXT_LBO_AI_AS_ID = 4,
  MTEXT_LBO_MAX }

関数

- int **mtext_line_break** (**MText** ∗mt, int pos, int option, int ∗after)
- **MText** ∗ **mtext** ()
  新しい *M-text* **を割り当てる**.

- **MText** ∗ **mtext_from_data** (const void ∗data, int nitems, enum **MTextFormat** format)
  指定のデータを元に新しい *M-text* **を割り当てる**.

- void ∗ **mtext_data** (**MText** ∗mt, enum **MTextFormat** ∗fmt, int ∗nunits, int ∗pos_idx, int ∗unit_idx)

- int **mtext_len** (**MText** ∗mt)
  M-text 中の文字の数.
- int **mtext_ref_char** (**MText** ∗mt, int pos)
  M-text 中の指定された位置の文字を返す.
- int **mtext_set_char** (**MText** ∗mt, int pos, int c)
  M-text に一文字を設定する.
- **MText** ∗ **mtext_cat_char** (**MText** ∗mt, int c)
  M-text に一文字追加する.
- **MText** ∗ **mtext_dup** (**MText** ∗mt)
  M-text のコピーを作る.
- **MText** ∗ **mtext_cat** (**MText** ∗mt1, **MText** ∗mt2)
  2 個の *M-text* を連結する.
- **MText** ∗ **mtext_ncat** (**MText** ∗mt1, **MText** ∗mt2, int n)
  M-text の一部を別の *M-text* に付加する.
- **MText** ∗ **mtext_cpy** (**MText** ∗mt1, **MText** ∗mt2)
  M-text を別の *M-text* にコピーする.
- **MText** ∗ **mtext_ncpy** (**MText** ∗mt1, **MText** ∗mt2, int n)
  M-text に含まれる最初の何文字かをコピーする.
- **MText** ∗ **mtext_duplicate** (**MText** ∗mt, int from, int to)
  既存の *M-text* の一部から新しい *M-text* をつくる.
- **MText** ∗ **mtext_copy** (**MText** ∗mt1, int pos, **MText** ∗mt2, int from, int to)
  M-text に指定範囲の文字をコピーする.
- int **mtext_del** (**MText** ∗mt, int from, int to)
  指定範囲の文字を破壊的に取り除く.
- int **mtext_ins** (**MText** ∗mt1, int pos, **MText** ∗mt2)
  M-text を別の *M-text* に挿入する.
- int **mtext_insert** (**MText** ∗mt1, int pos, **MText** ∗mt2, int from, int to)
  M-text の一部を別の *M-text* に挿入する.
- int **mtext_ins_char** (**MText** ∗mt, int pos, int c, int n)
  M-text に文字を挿入する.
- int **mtext_replace** (**MText** ∗mt1, int from1, int to1, **MText** ∗mt2, int from2, int to2)
  M-text の一部を別の *M-text* の一部で置換する.
- int **mtext_character** (**MText** ∗mt, int from, int to, int c)
  M-text 中で文字を探す.
- int **mtext_chr** (**MText** ∗mt, int c)

M-text 中で指定された文字が最初に現れる位置を返す.

- int **mtext_rchr** (**MText** ∗mt, int c)
  M-text 中で指定された文字が最後に現れる位置を返す.

- int **mtext_cmp** (**MText** ∗mt1, **MText** ∗mt2)
  二つの *M-text* を文字単位で比較する.

- int **mtext_ncmp** (**MText** ∗mt1, **MText** ∗mt2, int n)
  二つの *M-text* の先頭部分を文字単位で比較する.

- int **mtext_compare** (**MText** ∗mt1, int from1, int to1, **MText** ∗mt2, int from2, int to2)
  二つの *M-text* の指定した領域同士を比較する.

- int **mtext_spn** (**MText** ∗mt, **MText** ∗accept)
  ある集合の文字を *M-text* の中で探す.

- int **mtext_cspn** (**MText** ∗mt, **MText** ∗reject)
  ある集合に属さない文字を *M-text* の中で探す.

- int **mtext_pbrk** (**MText** ∗mt, **MText** ∗accept)
  ある集合に属す文字を *M-text* の中から探す.

- **MText** ∗ **mtext_tok** (**MText** ∗mt, **MText** ∗delim, int ∗pos)
  M-text 中のトークンを探す.

- int **mtext_text** (**MText** ∗mt1, int pos, **MText** ∗mt2)
  M-text 中で別の *M-text* を探す.

- int **mtext_search** (**MText** ∗mt1, int from, int to, **MText** ∗mt2)
  M-text 中の特定の領域で別の *M-text* を探す.

- int **mtext_casecmp** (**MText** ∗mt1, **MText** ∗mt2)
  二つの *M-text* を大文字／小文字の区別を無視して比較する.

- int **mtext_ncasecmp** (**MText** ∗mt1, **MText** ∗mt2, int n)
  二つの *M-text* の先頭部分を大文字／小文字の区別を無視して比較する.

- int **mtext_case_compare** (**MText** ∗mt1, int from1, int to1, **MText** ∗mt2, int from2, int to2)
  二つの *M-text* の指定した領域を、大文字／小文字の区別を無視して比較する.

- int **mtext_lowercase** (**MText** ∗mt)
  M-text を小文字にする.

- int **mtext_titlecase** (**MText** ∗mt)
  M-text をタイトルケースにする.

- int **mtext_uppercase** (**MText** ∗mt)
  M-text を大文字にする.

変数

- **MSymbol Mlanguage**

2.8.1 説明

M-text オブジェクトとそれに関する API.

m17n ライブラリは、 C-string（`char *` や `unsigned char *`）ではなく *M-text* と呼ぶオブジェクトでテキストを表現する。 M-text は長さ 0 以上の文字列であり、種々の文字ソース（たとえば C-string、ファイル、文字コード等）から作成できる。

M-text には、C-string にない以下の特徴がある。

- M-text は非常に多くの種類の文字を、同時に、混在させて、同等に扱うことができる。 Unicode の全ての文字はもちろん、より多くの文字までも扱うことができる。 これは多言語テキストを扱う上では必須の機能である。

- M-text 内の各文字は、**テキストプロパティ** と呼ばれるプロパティを持ち、 テキストプロパティによって、テキストの各部位に関する様々な情報を M-text 内に保持することが可能になる。 そのため、それらの情報をアプリケーションプログラム内で統一的に扱うことが可能になる。 また、M-text 自体が豊富な情報を持つため、アプリケーションプログラム中の各関数を簡素化することができる。

さらに m17n ライブラリは、 C-string を操作するために提供される種々の関数と同等のものを M-text を操作するためにサポートしている。

2.8.2 型定義

2.8.2.1 typedef struct MText MText

MText の型宣言.

Mtext (p. 23) は *M-text* オブジェクトの型である。 内部構造はアプリケーションプログラムからは見えない。

2.8.3 列挙型

2.8.3.1 enum MTextFormat

M-text のフォーマットを指定する列挙型.

列挙型 **MTextFormat** (p. 36) は関数 **mtext_from_data()** (p. 37) の引数として用いられ、 M-text を生成する元となるデータのフォーマットを指定する。

列挙型の値:

MTEXT_FORMAT_US_ASCII US-ASCII encoding

MTEXT_FORMAT_UTF_8 UTF-8 encoding

MTEXT_FORMAT_UTF_16LE UTF-16LE encoding

MTEXT_FORMAT_UTF_16BE UTF-16BE encoding

MTEXT_FORMAT_UTF_32LE UTF-32LE encoding

MTEXT_FORMAT_UTF_32BE UTF-32BE encoding

MTEXT_FORMAT_MAX

2.8.3.2 enum MTextLineBreakOption

列挙型の値:

MTEXT_LBO_SP_CM
MTEXT_LBO_KOREAN_SP
MTEXT_LBO_AI_AS_ID
MTEXT_LBO_MAX

2.8.4 関数

2.8.4.1 int mtext_line_break (MText * *mt*, int *pos*, int *option*, int * *after*)

2.8.4.2 MText* mtext ()

新しい M-text を割り当てる.

関数 **mtext()** (p. 37) は、長さ 0 の新しい M-text を割り当て、それへのポインタを返す。割り当てられた M-text は、関数 **m17n_object_unref()** (p. 12) によってユーザが明示的に行なわない限り、解放されない。

参照:

m17n_object_unref() (p. 12)

2.8.4.3 MText* mtext_from_data (const void * *data*, int *nitems*, enum MTextFormat *format*)

指定のデータを元に新しい M-text を割り当てる.

関数 **mtext_from_data()** (p. 37) は、要素数 **nitems** の配列 **data** で指定された文字列を持つ新しい M-text を割り当てる。**format** は **data** のフォーマットを示す。

format が **MTEXT_FORMAT_US_ASCII** (p. 36) か **MTEXT_FORMAT_UTF_8** (p. 36) ならば、 **data** の内容は `unsigned char` 型であり、**nitems** はバイト単位で表されている。

format が **MTEXT_FORMAT_UTF_16LE** (p. 36) か **MTEXT_FORMAT_UTF_16BE** (p. 36) ならば、**data** の内容は `unsigned short` 型であり、**nitems** は unsigned short 単位である。

format が **MTEXT_FORMAT_UTF_32LE** (p. 36) か **MTEXT_FORMAT_UTF_32BE** (p. 36) ならば、**data** の内容は `unsigned` 型であり、**nitems** は unsigned 単位である。

割り当てられた M-text の文字列は変更できない。**data** の内容は M-text が有効な間は変更してはならない。

割り当てられた M-text は、関数 **m17n_object_unref()** (p. 12) によってユーザが明示的に行なわない限り、解放されない。その場合でも **data** は解放されない。

戻り値:

処理が成功すれば、mtext_from_data() は割り当てられた M-text へのポインタを返す。そうでなければ `NULL` を返し外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:

`MERROR_MTEXT`

2.8.4.4 void* mtext_data (MText * *mt*, enum MTextFormat * *fmt*, int * *nunits*, int * *pos_idx*, int * *unit_idx*)

2.8.4.5 int mtext_len (MText * *mt*)

M-text 中の文字の数.

関数 **mtext_len()** (p. 37) は M-text **mt** 中の文字の数を返す。

2.8.4.6 int mtext_ref_char (MText * *mt*, int *pos*)

M-text 中の指定された位置の文字を返す.

関数 **mtext_ref_char()** (p. 38) は、M-text **mt** の位置 **pos** の文字を返す。エラーが検出された場合は -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:

MERROR_RANGE

2.8.4.7 int mtext_set_char (MText * *mt*, int *pos*, int *c*)

M-text に一文字を設定する.

関数 **mtext_set_char()** (p. 38) は、テキストプロパティ無しの文字 **c** を M-text **mt** の位置 **pos** に設定する。

戻り値:

処理に成功すれば **mtext_set_char()** (p. 38) は 0 を返す。失敗すれば -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:

MERROR_RANGE

2.8.4.8 MText* mtext_cat_char (MText * *mt*, int *c*)

M-text に一文字追加する.

関数 **mtext_cat_char()** (p. 38) は、テキストプロパティ無しの文字 **c** を M-text **mt** の末尾に追加する。

戻り値:

この関数は変更された M-text **mt** へのポインタを返す。**c** が正しい文字でない場合には NULL を返す。

参照:

mtext_cat() (p. 39), **mtext_ncat()** (p. 39)

2.8.4.9 MText* mtext_dup (MText * *mt*)

M-text のコピーを作る.

関数 **mtext_dup()** (p. 38) は、M-text **mt** のコピーを作る。**mt** のテキストプロパティはすべて継承される。

戻り値:

この関数は作られたコピーへのポインタを返す。

参照:

mtext_duplicate() (p. 40)

2.8.4.10 MText* mtext_cat (MText * *mt1*, MText * *mt2*)

2 個の M-text を連結する.

関数 **mtext_cat()** (p. 39) は、 M-text **mt2** を M-text **mt1** の末尾に付け加える。**mt2** のテキストプロパティはすべて継承される。**mt2** は変更されない。

戻り値:
この関数は変更された M-text **mt1** へのポインタを返す。

参照:
mtext_ncat() (p. 39), **mtext_cat_char()** (p. 38)

2.8.4.11 MText* mtext_ncat (MText * *mt1*, MText * *mt2*, int *n*)

M-text の一部を別の M-text に付加する.

関数 **mtext_ncat()** (p. 39) は、M-text **mt2** のはじめの **n** 文字を M-text **mt1** の末尾に付け加える。**mt2** のテキストプロパティはすべて継承される。**mt2** の長さが **n** 以下ならば、**mt2** のすべての文字が付加される。**mt2** は変更されない。

戻り値:
処理が成功した場合、mtext_ncat() は変更された M-text **mt1** へのポインタを返す。エラーが検出された場合は `NULL` を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
`MERROR_RANGE`

参照:
mtext_cat() (p. 39), **mtext_cat_char()** (p. 38)

2.8.4.12 MText* mtext_cpy (MText * *mt1*, MText * *mt2*)

M-text を別の M-text にコピーする.

関数 **mtext_cpy()** (p. 39) は M-text **mt2** を M-text **mt1** に上書きコピーする。 **mt2** のテキストプロパティはすべて継承される。**mt1** の長さは必要に応じて伸ばされる。**mt2** は変更されない。

戻り値:
この関数は変更された M-text **mt1** へのポインタを返す。

参照:
mtext_ncpy() (p. 39), **mtext_copy()** (p. 40)

2.8.4.13 MText* mtext_ncpy (MText * *mt1*, MText * *mt2*, int *n*)

M-text に含まれる最初の何文字かをコピーする.

関数 **mtext_ncpy()** (p. 39) は、M-text **mt2** の最初の **n** 文字を M-text **mt1** に上書きコピーする。**mt2** のテキストプロパティはすべて継承される。もし **mt2** の長さが **n** よりも小さければ **mt2** のすべての文字をコピーする。**mt1** の長さは必要に応じて伸ばされる。**mt2** は変更されない。

戻り値:
処理が成功した場合、mtext_ncpy() は変更された M-text **mt1** へのポインタを返す。エラーが検出された場合は NULL を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_RANGE

参照:
mtext_cpy() (p. 39), **mtext_copy()** (p. 40)

2.8.4.14 MText* mtext_duplicate (MText * *mt*, int *from*, int *to*)

既存の M-text の一部から新しい M-text をつくる.

関数 **mtext_duplicate()** (p. 40) は、M-text **mt** の **from** （**from** 自体も含む）から **to** （**to** 自体は含まない）までの部分のコピーを作る。このとき **mt** のテキストプロパティはすべて継承される。**mt** そのものは変更されない。

戻り値:
処理が成功すれば、mtext_duplicate() は作られた M-text へのポインタを返す。エラーが検出された場合は NULL を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_RANGE

参照:
mtext_dup() (p. 38)

2.8.4.15 MText* mtext_copy (MText * *mt1*, int *pos*, MText * *mt2*, int *from*, int *to*)

M-text に指定範囲の文字をコピーする.

関数 **mtext_copy()** (p. 40) は、 M-text **mt2** の **from** （**from** 自体も含む）から **to** （**to** 自体は含まない）までの範囲のテキストを M-text **mt1** の位置 **pos** から上書きコピーする。**mt2** のテキストプロパティはすべて継承される。**mt1** の長さは必要に応じて伸ばされる。**mt2** は変更されない。

戻り値:
処理が成功した場合、mtext_copy() は変更された **mt1** へのポインタを返す。そうでなければ NULL を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_RANGE

参照:
mtext_cpy() (p. 39), **mtext_ncpy()** (p. 39)

2.8.4.16 int mtext_del (MText * *mt*, int *from*, int *to*)

指定範囲の文字を破壊的に取り除く.

関数 **mtext_del()** (p. 40) は、M-text **mt** の **from** （**from** 自体も含む）から **to** （**to** 自体は含まない）までの文字を破壊的に取り除く。結果的に **mt** は長さが (**to** − **from**) だけ縮むことになる。

戻り値:
処理が成功すれば **mtext_del()** (p. 40) は 0 を返す。そうでなければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_RANGE

参照:
mtext_ins() (p. 41)

2.8.4.17 int mtext_ins (MText * *mt1*, int *pos*, MText * *mt2*)

M-text を別の M-text に挿入する.

関数 **mtext_ins()** (p. 41) は M-text **mt1** の **pos** の位置に別の M-text **mt2** を挿入する。この結果 **mt1** の長さは **mt2** の長さ分だけ増える。挿入の際、**mt2** のテキストプロパティはすべて継承される。**mt2** そのものは変更されない。

戻り値:
処理が成功すれば **mtext_ins()** (p. 41) は 0 を返す。そうでなければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_RANGE , MERROR_MTEXT

参照:
mtext_del() (p. 40) , **mtext_insert()** (p. 41)

2.8.4.18 int mtext_insert (MText * *mt1*, int *pos*, MText * *mt2*, int *from*, int *to*)

M-text の一部を別の M-text に挿入する.

関数 **mtext_insert()** (p. 41) は M-text **mt1** 中の **pos** の位置に、別の M-text **mt2** の **from** （**from** 自体も含む）から **to** （**to** 自体は含ま ない）までの文字を挿入する。結果的に **mt1** は長さが (**to** - **from**) だけ伸びる。挿入の際、 **mt2** 中のテキストプロパティはすべて継承され る。

戻り値:
処理が成功すれば、mtext_insert() は 0 を返す。そうでなければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_MTEXT , MERROR_RANGE

参照:
mtext_ins() (p. 41)

2.8.4.19 int mtext_ins_char (MText * *mt*, int *pos*, int *c*, int *n*)

M-text に文字を挿入する.

関数 **mtext_ins_char()** (p. 41) は M-text **mt** の **pos** の位置に文字 **c** のコピーを **n** 個挿入する。この結果 **mt1** の長さは **n** だけ増える。

戻り値:
処理が成功すれば **mtext_ins_char()** (p. 41) は 0 を返す。そうでなければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_RANGE

参照:
mtext_ins, **mtext_del()** (p. 40)

2.8.4.20 int mtext_replace (MText * *mt1*, int *from1*, int *to1*, MText * *mt2*, int *from2*, int *to2*)

M-text の一部を別の M-text の一部で置換する.

関数 **mtext_replace()** (p. 42) は、 M-text **mt1** の **from1** （**from1** 自体も含 む）から **to1** （**to1** 自体は含まない）までを、 M-text **mt2** の **from2** （**from2** 自体も含む）から **to2** （**to2** 自体は含まない）で置 き換える。新しく挿入された部分は、置き換える前のテキストプロパティ すべてを継承する。

戻り値:
処理が成功すれば、 **mtext_replace()** (p. 42) は 0 を返す。そうでな ければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_MTEXT , MERROR_RANGE

参照:
mtext_insert() (p. 41)

2.8.4.21 int mtext_character (MText * *mt*, int *from*, int *to*, int *c*)

M-text 中で文字を探す.

関数 **mtext_character()** (p. 42) は M-text **mt** 中で文字 **c** を探す。もし **from** が **to** より小さければ、探索は位置 **from** から末尾方向へ、最大 (**to** - 1) まで進む。そうでなければ位置 (**from** - 1) から先頭方向へ、最大 **to** まで進む。位置の指定に誤りがある場合は、**from** と **to** の両方に 0 が指定されたものとみなす。

戻り値:
もし **c** が見つかれば、mtext_character() はその最初の出現位置を返す。見つからなかった場合は外部変数 **merror_code** (p. 152) を変更せずに -1 を返す。エラーが検出された場合は -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

参照:
mtext_chr() (p. 42), **mtext_rchr()** (p. 43)

2.8.4.22 int mtext_chr (MText * *mt*, int *c*)

M-text 中で指定された文字が最初に現れる位置を返す.

関数 **mtext_chr()** (p. 42) は M-text **mt** 中で文字 **c** を探す。探索は **mt** の先頭から末尾方向に進む。

戻り値:
もし **c** が見つかれば、mtext_chr() はその出現位置を返す。見つからなかった場合は -1 を返す。

エラー:
MERROR_RANGE

参照:
mtext_rchr() (p. 43), **mtext_character()** (p. 42)

2.8.4.23 int mtext_rchr (MText * *mt*, int *c*)

M-text 中で指定された文字が最後に現れる位置を返す.

関数 **mtext_rchr()** (p. 43) は M-text **mt** 中で文字 **c** を探す。探索は **mt** の最後から先頭方向へと後向きに進む。

戻り値:
もし **c** が見つかれば、mtext_rchr() はその出現位置を返す。見つからなかった場合は -1 を返す。

エラー:
MERROR_RANGE

参照:
mtext_chr() (p. 42), **mtext_character()** (p. 42)

2.8.4.24 int mtext_cmp (MText * *mt1*, MText * *mt2*)

二つの M-text を文字単位で比較する.

関数 **mtext_cmp()** (p. 43) は、 M-text **mt1** と **mt2** を文字単位で比較する。

戻り値:
この関数は、**mt1** と **mt2** が等しければ 0、**mt1** が **mt2** より大きければ 1、**mt1** が **mt2** より小さければ -1 を返す。比較は文字コードに基づく。

参照:
mtext_ncmp() (p. 43), **mtext_casecmp()** (p. 45), **mtext_ncasecmp()** (p. 45), **mtext_compare()** (p. 44), **mtext_case_compare()** (p. 46)

2.8.4.25 int mtext_ncmp (MText * *mt1*, MText * *mt2*, int *n*)

二つの M-text の先頭部分を文字単位で比較する.

関数 **mtext_ncmp()** (p. 43) は、関数 **mtext_cmp()** (p. 43) 同様の M-text 同士の比較を先頭から最大 **n** 文字までに関して行なう。

戻り値:
この関数は、**mt1** と **mt2** が等しければ 0、**mt1** が **mt2** より大きければ 1、**mt1** が **mt2** より小さければ -1 を返す。

参照:
mtext_cmp() (p. 43), **mtext_casecmp()** (p. 45), **mtext_ncasecmp()** (p. 45) **mtext_compare()** (p. 44), **mtext_case_compare()** (p. 46)

2.8.4.26 int mtext_compare (MText * *mt1*, int *from1*, int *to1*, MText * *mt2*, int *from2*, int *to2*)

二つの M-text の指定した領域同士を比較する.

関数 **mtext_compare()** (p. 44) は二つの M-text **mt1** と **mt2** を文字単位で比較する。比較の対象は **mt1** のうち **from1** から **to1** までと、**mt2** のうち **from2** から **to2** までである。**from1** と **from2** は含まれ、**to1** と **to2** は含まれない。**from1** と **to1**（あるいは **from2** と **to2**）が等しい場合は長さゼロの M-text を意味する。範囲指定に誤りがある場合は、**from1** と **to1**（あるいは **from2** と **to2**）両方に 0 が指定されたものとみなす。

戻り値:
この関数は、**mt1** と **mt2** が等しければ 0、**mt1** が **mt2** より大きければ 1 、**mt1** が **mt2** より小さければ -1 を返す。比較は文字コードに基づく。

参照:
mtext_cmp() (p. 43), **mtext_ncmp()** (p. 43), **mtext_casecmp()** (p. 45), **mtext_ncasecmp()** (p. 45), **mtext_case_compare()** (p. 46)

2.8.4.27 int mtext_spn (MText * *mt*, MText * *accept*)

ある集合の文字を M-text の中で探す.

関数 **mtext_spn()** (p. 44) は、M-text **mt1** の先頭から M-text **mt2** に含まれる文字だけでできている部分の長さを返す。

参照:
mtext_cspn() (p. 44)

2.8.4.28 int mtext_cspn (MText * *mt*, MText * *reject*)

ある集合に属さない文字を M-text の中で探す.

関数 **mtext_cspn()** (p. 44) は、M-text **mt1** の先頭部分で M-text **mt2** に含まれない文字だけでできている部分の長さを返す。

参照:
mtext_spn() (p. 44)

2.8.4.29 int mtext_pbrk (MText * *mt*, MText * *accept*)

ある集合に属す文字を M-text の中から探す.

関数 **mtext_pbrk()** (p. 44) は、M-text **mt1** 中で M-text **mt2** の文字のどれかが最初に現れる位置を調べる。

戻り値:
見つかった文字の、**mt1** 内における出現位置を返す。もしそのような文字がなければ -1 を返す。

2.8.4.30 MText* mtext_tok (MText * *mt*, MText * *delim*, int * *pos*)

M-text 中のトークンを探す.

関数 **mtext_tok()** (p. 44) は、M-text **mt** の中で位置 **pos** 以降最初に現れるトークンを探す。ここでトークンとは M-text **delim** の中に現われない文字だけからなる部分文字列である。**pos** の型が `int` ではなくて `int` へのポインタであることに注意。

戻り値:

もしトークンが見つかれば **mtext_tok()** (p. 44) はそのトークンに相当する部分の **mt** をコピーし、そのコピーへのポインタを返す。この場合、**pos** は見つかったトークンの終端にセットされる。トークンが見つからなかった場合は外部変数 **merror_code** (p. 152) を変えずに NULL を返す。エラーが検出された場合は NULL を返し、変部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:

MERROR_RANGE

2.8.4.31 int mtext_text (MText * *mt1*, int *pos*, MText * *mt2*)

M-text 中で別の M-text を探す.

関数 **mtext_text()** (p. 45) は、M-text **mt1** 中で位置 **pos** 以降に現われる M-text **mt2** の最初の位置を調べる。テキストプロパティの違いは無視される。

戻り値:

mt1 中に **mt2** が見つかれば、mtext_text() はその最初の出現位置を返す。見つからない場合は -1 を返す。もし **mt2** が空ならば 0 を返す。

2.8.4.32 int mtext_search (MText * *mt1*, int *from*, int *to*, MText * *mt2*)

M-text 中の特定の領域で別の M-text を探す.

関数 **mtext_search()** (p. 45) は、M-text **mt1** 中の **from** から **to** までの間の領域で M-text **mt2** が最初に現われる位置を調べる。テキストプロパティの違いは無視される。もし **from** が **to** より小さければ探索は位置 **from** から末尾方向へ、そうでなければ **to** から先頭方向へ進む。

戻り値:

mt1 中に **mt2** が見つかれば、mtext_search() はその最初の出現位置を返す。見つからない場合は -1 を返す。もし **mt2** が空ならば 0 を返す。

2.8.4.33 int mtext_casecmp (MText * *mt1*, MText * *mt2*)

二つの M-text を大文字／小文字の区別を無視して比較する.

関数 **mtext_casecmp()** (p. 45) は、関数 **mtext_cmp()** (p. 43) 同様の M-text 同士の比較を、大文字／小文字の区別を無視して行なう。

戻り値:

この関数は、**mt1** と **mt2** が等しければ 0、**mt1** が **mt2** より大きければ 1、**mt1** が **mt2** より小さければ -1 を返す。

参照:

mtext_cmp() (p. 43), **mtext_ncmp()** (p. 43), **mtext_ncasecmp()** (p. 45) **mtext_compare()** (p. 44), **mtext_case_compare()** (p. 46)

2.8.4.34 int mtext_ncasecmp (MText * *mt1*, MText * *mt2*, int *n*)

二つの M-text の先頭部分を大文字／小文字の区別を無視して比較する.

関数 **mtext_ncasecmp()** (p. 45) は、関数 **mtext_casecmp()** (p. 45) 同様の M-text 同士の比較を先頭から最大 **n** 文字までに関して行なう。

戻り値:
この関数は、**mt1** と **mt2** が等しければ 0、**mt1** が **mt2** より大きければ 1、**mt1** が **mt2** より小さければ -1 を返す。

参照:
mtext_cmp() (p. 43), **mtext_casecmp()** (p. 45), **mtext_casecmp()** (p. 45) **mtext_compare()** (p. 44), **mtext_case_compare()** (p. 46)

2.8.4.35 int mtext_case_compare (MText * *mt1*, int *from1*, int *to1*, MText * *mt2*, int *from2*, int *to2*)

二つの M-text の指定した領域を、大文字／小文字の区別を無視して比較する.

関数 **mtext_compare()** (p. 44) は二つの M-text **mt1** と **mt2** を、大文字／小文字の区別を無視して文字単位で比較する。比較の対象は **mt1** の **from1** から **to1** まで、**mt2** の **from2** から **to2** までである。 **from1** と **from2** は含まれ、**to1** と **to2** は含まれない。**from1** と **to1**（あるいは **from2** と **to2**）が等しい場合は長さゼロの M-text を意味する。範囲指定に誤りがある場合は、**from1** と **to1**（あるいは **from2** と **to2**）両方に 0 が指定されたものと見なす。

戻り値:
この関数は、**mt1** と **mt2** が等しければ 0、**mt1** が **mt2** より大きければ 1、**mt1** が **mt2** より小さければ -1 を返す。比較は文字コードに基づく。

参照:
mtext_cmp() (p. 43), **mtext_ncmp()** (p. 43), **mtext_casecmp()** (p. 45), **mtext_ncasecmp()** (p. 45), **mtext_compare()** (p. 44)

2.8.4.36 int mtext_lowercase (MText * *mt*)

M-text を小文字にする.

関数 **mtext_lowercase()** (p. 46) は M-text **mt** 中の各文字を破壊的に小文字に変 換する。変換に際して隣接する文字の影響を受けることがある。**mt** にテ キストプロパティ Mlanguage が付いている場合は、それも変換に影響を 与えうる。**mt** の長さは変わることがある。小文字に変換できなかった文 字はそのまま残る。テキストプロパティはすべて継承される。

戻り値:
この関数は更新後の **mt** の長さを返す。

参照:
mtext_titlecase() (p. 46), **mtext_uppercase()** (p. 47)

2.8.4.37 int mtext_titlecase (MText * *mt*)

M-text をタイトルケースにする.

関数 **mtext_titlecase()** (p. 46) は M-text **mt** 中で cased プロパティを持つ 最初の文字をタイトルケースに、そしてそれ以降の文字を小文字に破壊的 に変換する。**mt** の長さは変わることがある。タイトルケースにに変換で きなかった場合はそのままで変わらない。テキストプロパティはすべて継 承される。

戻り値:
この関数は更新後の **mt** の長さを返す。

参照:
mtext_lowercase() (p. 46), **mtext_uppercase()** (p. 47)

2.8.4.38 int mtext_uppercase (MText * *mt*)

M-text を大文字にする.

関数 **mtext_uppercase()** (p. 47) は M-text **mt** 中の各文字を破壊的に大文字に変 換する。変換に際して隣接する文字の影響を受けることがある。**mt** にテ キストプロパティ Mlanguage が付いている場合は、それも変換に影響を 与えうる。**mt** の長さは変わることがある。大文字に変換できなかった文 字はそのまま残る。テキストプロパティはすべて継承される。

戻り値:
この関数は更新後の **mt** の長さを返す。

参照:
mtext_lowercase() (p. 46), **mtext_titlecase()** (p. 46)

2.8.5 変数

2.8.5.1 enum MTextFormat MTEXT_FORMAT_UTF_16

値が MTEXT_FORMAT_UTF_16LE か MTEXT_FORMAT_UTF_16BE である変数

大域変数 **MTEXT_FORMAT_UTF_16** (p. 47) はリトル・エンディアン・システム （ワードを LSB (Least Significant Byte) を先にして格納）上では **MTEXT_FORMAT_UTF_16LE** (p. 36) に初期化され、ビッグ・エンディアン・システム （ワードを MSB (Most Significant Byte) を先にして格納）上では **MTEXT_FORMAT_UTF_16BE** (p. 36) に初期化される。

参照:
mtext_from_data() (p. 37)

2.8.5.2 const int MTEXT_FORMAT_UTF_32

値が MTEXT_FORMAT_UTF_32LE か MTEXT_FORMAT_UTF_32BE である変数

大域変数 **MTEXT_FORMAT_UTF_32** (p. 47) はリトル・エンディアン・システム （ワードを LSB (Least Significant Byte) を先にして格納）上では **MTEXT_FORMAT_UTF_32LE** (p. 36) に初期化され、ビッグ・エンディアン・システム （ワードを MSB (Most Significant Byte) を先にして格納）上では **MTEXT_FORMAT_UTF_32BE** (p. 36) に初期化される。

参照:
mtext_from_data() (p. 37)

2.8.5.3 MSymbol Mlanguage

"language" という名前を持つシンボル.

2.9 テキストプロパティ

テキストプロパティを操作するための関数.

型定義

- typedef **MPlist** ∗(∗ **MTextPropSerializeFunc**)(void ∗val)
  シリアライザ関数の型宣言.

- typedef void ∗(∗ **MTextPropDeserializeFunc**)(**MPlist** ∗plist)
  デシリアライザ関数の型宣言.

- typedef struct **MTextProperty MTextProperty**
  テキストプロパティの型宣言.

列挙型

- enum **MTextPropertyControl** {
  MTEXTPROP_FRONT_STICKY = 0x01,
  MTEXTPROP_REAR_STICKY = 0x02,
  MTEXTPROP_VOLATILE_WEAK = 0x04,
  MTEXTPROP_VOLATILE_STRONG = 0x08,
  MTEXTPROP_NO_MERGE = 0x10,
  MTEXTPROP_CONTROL_MAX = 0x1F }
  テキストプロパティを制御するフラグビット.

関数

- void ∗ **mtext_get_prop** (**MText** ∗mt, int pos, **MSymbol** key)
  テキストプロパティの一番上の値を得る.

- int **mtext_get_prop_values** (**MText** ∗mt, int pos, **MSymbol** key, void ∗∗values, int num)
  テキストプロパティの値を複数個得る.

- int **mtext_get_prop_keys** (**MText** ∗mt, int pos, **MSymbol** ∗∗keys)
  M-text **の指定した位置のテキストプロパティのキーのリストを得る**.

- int **mtext_put_prop** (**MText** ∗mt, int from, int to, **MSymbol** key, void ∗val)
  テキストプロパティを設定する.

- int **mtext_put_prop_values** (**MText** ∗mt, int from, int to, **MSymbol** key, void ∗∗values, int num)
  同じキーのテキストプロパティを複数設定する.

- int **mtext_push_prop** (**MText** ∗mt, int from, int to, **MSymbol** key, void ∗val)
  テキストプロパティをプッシュする.

- int **mtext_pop_prop** (**MText** ∗mt, int from, int to, **MSymbol** key)

テキストプロパティをポップする.

- int **mtext_prop_range** (**MText** ∗mt, **MSymbol** key, int pos, int ∗from, int ∗to, int deeper)
  テキストプロパティが同じ値をとる範囲を調べる.

- **MTextProperty** ∗ **mtext_property** (**MSymbol** key, void ∗val, int control_bits)
  テキストプロパティを生成する.

- **MText** ∗ **mtext_property_mtext** (**MTextProperty** ∗prop)
  あるテキストプロパティを持つ *M-text* を返す.

- **MSymbol mtext_property_key** (**MTextProperty** ∗prop)
  テキストプロパティのキーを返す.

- void ∗ **mtext_property_value** (**MTextProperty** ∗prop)
  テキストプロパティの値を返す.

- int **mtext_property_start** (**MTextProperty** ∗prop)
  テキストプロパティの開始位置を返す.

- int **mtext_property_end** (**MTextProperty** ∗prop)
  テキストプロパティの終了位置を返す.

- **MTextProperty** ∗ **mtext_get_property** (**MText** ∗mt, int pos, **MSymbol** key)
  一番上のテキストプロパティを得る.

- int **mtext_get_properties** (**MText** ∗mt, int pos, **MSymbol** key, **MTextProperty** ∗∗props, int num)
  複数のテキストプロパティを得る.

- int **mtext_attach_property** (**MText** ∗mt, int from, int to, **MTextProperty** ∗prop)
  M-text にテキストプロパティを付加する.

- int **mtext_detach_property** (**MTextProperty** ∗prop)
  M-text からテキストプロパティを分離する.

- int **mtext_push_property** (**MText** ∗mt, int from, int to, **MTextProperty** ∗prop)
  M-text にテキストプロパティをプッシュする.

- **MText** ∗ **mtext_serialize** (**MText** ∗mt, int from, int to, **MPlist** ∗property_list)
  M-text 中のテキストプロパティをシリアライズする.

- **MText** ∗ **mtext_deserialize** (**MText** ∗mt)
  M-text 中のテキストプロパティをデシリアライズする.

変数

- **MSymbol Mtext_prop_serializer**
  シリアライザ関数を指定するシンボル.

- **MSymbol Mtext_prop_deserializer**
  デシリアライザ関数を指定するシンボル.

2.9.1 説明

テキストプロパティを操作するための関数.

M-text 内の各文字は、**テキストプロパティ** と呼ばれるプロパティを 持つことができる。テキストプロパティは、M-text の各部位に付加され たさまざまな情報を保持しており、アプリケーションプログラムはそれら の情報を統一的に扱うことができる。M-text 自体が豊富な情報を持つた め、アプリケーションプログラム中の関数を簡素化することができる。

テキストプロパティは **キー** と **値** からなる。キーはシンボルであ り、値は (`void *`) 型にキャストできるものなら何でもよい。 他のタイプのプロパティと異なり、一つのテキストプロパティが複数の値 を持つことが許される。「キーが K であるテキストプロパティ」のこと を簡単に「K プロパティ」と呼ぶことがある。

2.9.2 型定義

2.9.2.1 typedef MPlist*(* MTextPropSerializeFunc)(void *val)

シリアライザ関数の型宣言.

シリアライザ関数の型である。 あるシンボルのプロパティのキーが **`Mtext_prop_serializer`** (p. 57) であるとき、 値はこの型でなくてはならない。

参照:
mtext_serialize() (p. 56), **Mtext_prop_serializer** (p. 57)

2.9.2.2 typedef void*(* MTextPropDeserializeFunc)(MPlist *plist)

デシリアライザ関数の型宣言.

デシリアライザ関数の型である。 あるシンボルのプロパティのキーが **`Mtext_prop_deserializer`** (p. 58) であるとき、 値はこの型でなくてはならない。

参照:
Mtext_prop_deserialize(), Mtext_prop_deserializer

2.9.2.3 typedef struct MTextProperty MTextProperty

テキストプロパティの型宣言.

MTextProperty (p. 50) は **テキストプロパティ** オブジェクトの型である。 内部構造はアプリケーションプログラムからは見えない。

2.9.3 列挙型

2.9.3.1 enum MTextPropertyControl

テキストプロパティを制御するフラグビット.

関数 **mtext_property()** (p. 54) は以下のフラグビットの論理 OR を引数としてとることができる。 フラグビットは生成されたテキストプロパティの振舞いを制御する。 詳細は各フラグビットの説明を参照。

列挙型の値:
MTEXTPROP_FRONT_STICKY このビットが on ならば、このテキストプロパティの始まる点あるいは中間に挿入された M-text はこのテキストプロパティを継承する。

MTEXTPROP_REAR_STICKY このビットが on ならば、このテキストプロパティの終わる点あるいは中間に挿入された M-text はこのテキストプロパティを継承する。

MTEXTPROP_VOLATILE_WEAK このビットが on ならば、このテキストプロパティの範囲内のテキストが変更された場合テキストプロパティは取り除かれる。

MTEXTPROP_VOLATILE_STRONG このビットが on ならば、このテキストプロパティの範囲内のテキストあるいは別のテキストプロパティが変更された場合このテキ ストプロパティは取り除かれる。

MTEXTPROP_NO_MERGE このビットが on ならば、このテキストプロパティは他のプロパティと自動的にはマージされない。

MTEXTPROP_CONTROL_MAX

2.9.4 関数

2.9.4.1 void* mtext_get_prop (MText * *mt*, int *pos*, MSymbol *key*)

テキストプロパティの一番上の値を得る.

関数 **mtext_get_prop()** (p. 51) は、M-text **mt** 内の位置 **pos** にある文字のテ キストプロパティのうち、キーが **key** であるものを探す。

戻り値:

テキストプロパティがみつかれば、mtext_get_prop() はそのプロパティ の値を返す。値が複数存在するときは、一番上の値を返す。見つからなけ れば外部変数 **merror_code** (p. 152) を変更することなく NULL を返す。

エラーが検出された場合 **mtext_get_prop()** (p. 51) は NULL を返し、外部変 数 **merror_code** (p. 152) にエラーコードを設定する。

覚え書き:

エラーなしで NULL が返された場合には二つの可能性がある。

- **pos** の位置の文字は **key** をキーとするプロパティを持たない。
- その文字はそのようなプロパティを持ち、その値が NULL である。

この二つを区別する必要がある場合には、関数 **mtext_get_prop_values()** (p. 51) を代わりに使用すること。

エラー:

MERROR_RANGE, MERROR_SYMBOL

参照:

mtext_get_prop_values() (p. 51), **mtext_put_prop()** (p. 52), **mtext_put_prop_values()** (p. 53), **mtext_push_prop()** (p. 53), **mtext_pop_prop()** (p. 53), **mtext_prop_range()** (p. 54)

2.9.4.2 int mtext_get_prop_values (MText * *mt*, int *pos*, MSymbol *key*, void ** *values*, int *num*)

テキストプロパティの値を複数個得る.

関数 **mtext_get_prop_values()** (p. 51) は、M-text **mt** 内で **pos** という位置 にある文字のプロパティのうち、キーが **key** であるものを探す。もしそ のようなプロパティが見つかれば、それが持つ値 (複数可) を **values** の指すメモリ領域に格納する。**num** は格納する値の数の上限である。

戻り値:
処理が成功すれば、mtext_get_prop_values() は実際にメモリに格納さ れた値の数を返す。**pos** の位置の文字が **key** をキーとするプロパティ を持たなければ 0 を返す。エラーが検出された場合は -1 を返し、外部 変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_RANGE, MERROR_SYMBOL

参照:
mtext_get_prop() (p. 51), **mtext_put_prop()** (p. 52), **mtext_put_prop_values()** (p. 53), **mtext_push_prop()** (p. 53), **mtext_pop_prop()** (p. 53), **mtext_prop_range()** (p. 54)

2.9.4.3 int mtext_get_prop_keys (MText * *mt*, int *pos*, MSymbol ** *keys*)

M-text の指定した位置のテキストプロパティのキーのリストを得る.

関数 **mtext_get_prop_keys()** (p. 52) は、M-text **mt** 内で **pos** の位置にある すべてのテキストプロパティのキーを要素とする配列を作り、その配列の アドレスを ***keys** に設定する。この配列のために確保されたメモリを解 放するのはユーザの責任である。

戻り値:
処理が成功すれば **mtext_get_prop_keys()** (p. 52) は得られたリストの長さを返 す。そうでなければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを 設定する。

エラー:
MERROR_RANGE

参照:
mtext_get_prop() (p. 51), **mtext_put_prop()** (p. 52), **mtext_put_prop_values()** (p. 53), **mtext_get_prop_values()** (p. 51), **mtext_push_prop()** (p. 53), **mtext_pop_prop()** (p. 53)

2.9.4.4 int mtext_put_prop (MText * *mt*, int *from*, int *to*, MSymbol *key*, void * *val*)

テキストプロパティを設定する.

関数 **mtext_put_prop()** (p. 52) は、M-text **mt** の **from** （含まれる）から **to**（含まれない）の範囲の文字に、キーが **key** で値が **val** であるよ うなテキストプロパティを設定する。この関数によって

```
                        FROM                    TO
M-text:       |<------------|-------- MT ---------|------------>|
PROP:          <------------------ OLD_VAL --------------------->
```

は次のようになる。

```
                        FROM                    TO
M-text:       |<------------|-------- MT ---------|------------>|
PROP:          <-- OLD_VAL-><-------- VAL -------><-- OLD_VAL-->
```

戻り値:
処理が成功すれば **mtext_put_prop()** (p. 52) は 0 を返す。そうでなければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_RANGE, MERROR_SYMBOL

参照:
mtext_put_prop_values() (p. 53), **mtext_get_prop()** (p. 51), **mtext_get_prop_values()** (p. 51), **mtext_push_prop()** (p. 53), **mtext_pop_prop()** (p. 53), **mtext_prop_range()** (p. 54)

2.9.4.5 int mtext_put_prop_values (MText * *mt*, int *from*, int *to*, MSymbol *key*, void ** *values*, int *num*)

同じキーのテキストプロパティを複数設定する.

関数 **mtext_put_prop_values()** (p. 53) は、M-Text **mt** の **from** （含まれる） から **to** （含まれない）の範囲の文字に、テキストプロパティを設定す る。テキストプロパティのキーは **key** によって、値 (複数可) は **values** によって指定される。**num** は設定される値の個数である。

戻り値:

処理が成功すれば、mtext_put_prop_values() は 0 を返す。そうでなけ れば -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:

MERROR_RANGE, MERROR_SYMBOL

参照:

mtext_put_prop() (p. 52), **mtext_get_prop()** (p. 51), **mtext_get_prop_values()** (p. 51), **mtext_push_prop()** (p. 53), **mtext_pop_prop()** (p. 53), **mtext_prop_range()** (p. 54)

2.9.4.6 int mtext_push_prop (MText * *mt*, int *from*, int *to*, MSymbol *key*, void * *val*)

テキストプロパティをプッシュする.

関数 **mtext_push_prop()** (p. 53) は、キーが **key** で値が **val** であるテキスト プロパティを、M-text **mt** 中の **from** （含まれる）から **to** （含まれな い）の範囲の文字にプッシュする。この関数によって

```
                    FROM                    TO
M-text: |<------------|-------- MT ---------|------------>|
PROP   :  <------------------ OLD_VAL ---------------------->
```

は次のようになる。

```
                    FROM                    TO
M-text: |<------------|-------- MT ---------|------------>|
PROP   :  <-------------------- OLD_VAL --------------------->
PROP   :                <-------- VAL ------->
```

戻り値:

処理が成功すれば、mtext_push_prop() は 0 を返す。そうでなければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:

MERROR_RANGE, MERROR_SYMBOL

参照:

mtext_put_prop() (p. 52), **mtext_put_prop_values()** (p. 53), **mtext_get_prop()** (p. 51), **mtext_get_prop_values()** (p. 51), **mtext_pop_prop()** (p. 53), **mtext_prop_range()** (p. 54)

2.9.4.7 int mtext_pop_prop (MText * *mt*, int *from*, int *to*, MSymbol *key*)

テキストプロパティをポップする.

関数 **mtext_pop_prop()** (p. 53) は、キーが **key** であるテキストプロパティの うち一番上のものを、M-text **mt** の **from** （含まれる）から **to**（含ま れない）の範囲の文字から取り除く。

指定範囲の文字がそのようなプロパティを持たないならば、この関数は何 もしない。この関数によって、

```
                    FROM                    TO
M-text: |<------------|-------- MT ---------|------------>|
PROP  :  <------------------ OLD_VAL --------------------->
```

は以下のようになる。

```
                    FROM                    TO
M-text: |<------------|-------- MT ---------|------------>|
PROP  :  <--OLD_VAL-->|                     |<--OLD_VAL-->|
```

戻り値:

処理が成功すれば、mtext_pop_prop() は 0 を返す。そうでなければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:

MERROR_RANGE, MERROR_SYMBOL

参照:

mtext_put_prop() (p. 52), **mtext_put_prop_values()** (p. 53), **mtext_get_prop()** (p. 51), **mtext_get_prop_values()** (p. 51), **mtext_push_prop()** (p. 53), **mtext_prop_range()** (p. 54)

2.9.4.8 int mtext_prop_range (MText ∗ *mt*, MSymbol *key*, int *pos*, int ∗ *from*, int ∗ *to*, int *deeper*)

テキストプロパティが同じ値をとる範囲を調べる.

関数 **mtext_prop_range()** (p. 54) は、指定したテキストプロパティの値が同じ である連続した文字の範囲を調べる。まず M-text **mt** の **pos** の位置に ある文字のプロパティのうち、キー **key** で指定されたもの値を見つけ る。そして前後の文字も **key** のプロパティの値が同じであるかどうかを 調べる。見つけた範囲の最初と最後を、それぞれ **from** と **to** にポイン トされる変数に保存する。**from** に保存される文字の位置は見つけた範囲 に含まれるが、**to** は含まれない。(**to** の前で同じ値をとる範囲は終わ る。) この範囲指定法は、関数 **mtext_put_prop()** (p. 52) などと共通である。

deeper が 0 でなければ、**key** というキーを持つプロパティのうち一番 上のものだけでなく、スタック中のすべてのものが比較される。

from が NULL ならば、範囲の始まりは探索しない。**to** が NULL ならば、範囲の終りは探索しない。

戻り値:

処理が成功すれば、mtext_prop_range() は **key** プロパティの値の数を 返す。そうでなければ-1 を返し、外部変数 **merror_code** (p. 152) にエラーコー ドを設定する。

エラー:

MERROR_RANGE, MERROR_SYMBOL

参照:

mtext_put_prop() (p. 52), **mtext_put_prop_values()** (p. 53), **mtext_get_prop()** (p. 51), **mtext_get_prop_values()** (p. 51), **mtext_pop_prop()** (p. 53), **mtext_push_prop()** (p. 53)

2.9.4.9 MTextProperty∗ mtext_property (MSymbol *key*, void ∗ *val*, int *control_bits*)

テキストプロパティを生成する.

関数 **mtext_property()** (p. 54) は **key** をキー、**val** を値とする新しく割り当 てられたテキストプロパティを返す。生成したテキストプロパティはいか なる M-text にも付加されていない、すなわち分離して (detached) いる。

control_bits は 0 であるか enum MTextPropertyControl の論理 OR でなくてはならない。

2.9.4.10 MText* mtext_property_mtext (MTextProperty * *prop*)

あるテキストプロパティを持つ M-text を返す.

関数 **mtext_property_mtext()** (p. 55) は、テキストプロパティ **prop** が付加さ れている M-text を返す。その時点で **prop** が分離していれば NULL を 返す。

2.9.4.11 MSymbol mtext_property_key (MTextProperty * *prop*)

テキストプロパティのキーを返す.

関数 **mtext_property_key()** (p. 55) は、テキストプロパティ **prop** のキー（シ ンボル）を返す。

2.9.4.12 void* mtext_property_value (MTextProperty * *prop*)

テキストプロパティの値を返す.

関数 **mtext_property_value()** (p. 55) は、テキストプロパティ **prop** の値を返 す。

2.9.4.13 int mtext_property_start (MTextProperty * *prop*)

テキストプロパティの開始位置を返す.

関数 **mtext_property_start()** (p. 55) は、テキストプロパティ **prop** の開始位 置を返す。開始位置とは M-text 中で **prop** が始まる文字位置である。 **prop** が分離されていれば、-1 を返す。

2.9.4.14 int mtext_property_end (MTextProperty * *prop*)

テキストプロパティの終了位置を返す.

関数 **mtext_property_end()** (p. 55) は、テキストプロパティ **prop** の終了位置 を返す。終了位置とは M-text 中で **prop** が終る文字位置である。**prop** が分離されていれば、-1 を返す。

2.9.4.15 MTextProperty* mtext_get_property (MText * *mt*, int *pos*, MSymbol *key*)

一番上のテキストプロパティを得る.

関数 **mtext_get_property()** (p. 55) は M-text **mt** の位置 **pos** の文字がキー が **key** であるテキストプロパティを持つかどうかを調べる。

戻り値:

テキストプロパティが見つかれば、mtext_get_property() はそれを返す。 複数ある場合には、一番上のものを返す。見つからなければ、外部変数 **merror_code** (p. 152) を変えることなく NULL を返す。

エラーが検出された場合 **mtext_get_property()** (p. 55) は NULL を返し、外 部変数 **merror_code** (p. 152) にエラーコードを設定する。

2.9.4.16 int mtext_get_properties (MText * *mt*, int *pos*, MSymbol *key*, MTextProperty ** *props*, int *num*)

複数のテキストプロパティを得る.

関数 **mtext_get_properties()** (p. 55) は M-text **mt** の位置 **pos** の文字がキー が **key** であるテキストプロパティを持つかどうかを調べる。そのような プロパティがみつかれば、**props** が指すメモリ領域に保存する。**num** は 保存されるプロパティの数の上限である。

戻り値:
処理が成功すれば、mtext_get_properties() は実際に保存したプロパティ の数を返す。**pos** の位置の文字がキーが **key** であるプロパティを持た なければ、0 が返る。エラーが検出された場合には、**mtext_get_properties()** (p. 55) は -1 を返し、外部変数 **merror_code** (p. 152) にエラー コードを設定する。

2.9.4.17 int mtext_attach_property (MText * *mt*, int *from*, int *to*, MTextProperty * *prop*)

M-text にテキストプロパティを付加する.

関数 **mtext_attach_property()** (p. 56) は、M-text **mt** の **from** から **to** ま での領域にテキストプロパティ **prop** を付加する。もし **prop** が既に M-text に付加されていれば、**mt** に付加する前に分離される。

戻り値:
処理に成功すれば、mtext_attach_property() は 0 を返す。そうでなけ れば -1 を返して外部変数 **merror_code** (p. 152) にエラーコードを設定する。

2.9.4.18 int mtext_detach_property (MTextProperty * *prop*)

M-text からテキストプロパティを分離する.

関数 **mtext_detach_property()** (p. 56) はテキストプロパティ **prop** を分離する。

戻り値:
この関数は常に 0 を返す。

2.9.4.19 int mtext_push_property (MText * *mt*, int *from*, int *to*, MTextProperty * *prop*)

M-text にテキストプロパティをプッシュする.

関数 **mtext_push_property()** (p. 56) は、テキストプロパティ **prop** を、 M-text **mt** 中の **from** （含まれる）から **to** （含まれない）の範囲の 文字にプッシュする。

戻り値:
処理に成功すれば、mtext_push_property() は 0 を返す。そうでなけ れば -1 を返して外部変数 **merror_code** (p. 152) にエラーコードを設定する。

2.9.4.20 MText* mtext_serialize (MText * *mt*, int *from*, int *to*, MPlist * *property_list*)

M-text 中のテキストプロパティをシリアライズする.

関数 **mtext_serialize()** (p. 56) は M-text **mt** の **from** から **to** までのテキ ストをシリアライズする。シリアライズした結果は XML 形式の M-text で ある。 **property_list** はシリアライズされるテキストプロパティを限定 する。対象となるテキストプロパティは、そのキーが

- **property_list** の要素の値として現われ、かつ
- シンボルプロパティ **Mtext_prop_serializer** (p. 57) を持つ

もののみである。この条件を満たすテキストプロパティは、生成される XML 表現中で "property" 要素にシリアライズされる。

生成される XML の DTD は以下の通り:

```
<!DOCTYPE mtext [
  <!ELEMENT mtext (property*,body+)>
  <!ELEMENT property EMPTY>
  <!ELEMENT body (#PCDATA)>
  <!ATTLIST property key CDATA #REQUIRED>
  <!ATTLIST property value CDATA #REQUIRED>
  <!ATTLIST property from CDATA #REQUIRED>
  <!ATTLIST property to CDATA #REQUIRED>
  <!ATTLIST property control CDATA #REQUIRED>
 ]>
```

この関数は libxml2 ライブラリに依存する。m17n ライブラリが libxml2 無しに設定されている場合、この関数は常に失敗する。

戻り値:

処理に成功すれば、mtext_serialize() は XML 形式で M-text を返す。 そうでなければ NULL を返して外部変数 **merror_code** (p. 152) にエラーコード を設定する。

参照:

mtext_deserialize() (p. 57), **Mtext_prop_serializer** (p. 57)

2.9.4.21 MText* mtext_deserialize (MText * *mt*)

M-text 中のテキストプロパティをデシリアライズする.

関数 **mtext_deserialize()** (p. 57) は M-text **mt** をデシリアライズする。**mt** は次の DTD を持つ XML でなくてはならない。

```
<!DOCTYPE mtext [
  <!ELEMENT mtext (property*,body+)>
  <!ELEMENT property EMPTY>
  <!ELEMENT body (#PCDATA)>
  <!ATTLIST property key CDATA #REQUIRED>
  <!ATTLIST property value CDATA #REQUIRED>
  <!ATTLIST property from CDATA #REQUIRED>
  <!ATTLIST property to CDATA #REQUIRED>
  <!ATTLIST property control CDATA #REQUIRED>
 ]>
```

この関数は libxml2 ライブラリに依存する。m17n ライブラリが libxml2 無しに設定されている場合、この関数は常に失敗する。

戻り値:

処理に成功すれば、mtext_serialize() は得られた M-text を 返す。そうでなければ NULL を返して外部変数 **merror_code** (p. 152) にエラー コードを設定する。

参照:

mtext_serialize() (p. 56), **Mtext_prop_deserializer** (p. 58)

2.9.5 変数

2.9.5.1 MSymbol Mtext_prop_serializer

シリアライザ関数を指定するシンボル.

テキストプロパティをシリアライズするためには、そのテキストプロパ ティ用のシリアライザ関数を与えなくてはならない。具体的には、 **Mtext_prop_serializer** (p. 57) をキーとし、適切なシリアライズ関数へのポイ ンタを値とするシンボルプロパティを指定する。

参照:
mtext_serialize() (p. 56), **MTextPropSerializeFunc** (p. 50)

2.9.5.2 MSymbol Mtext_prop_deserializer

デシリアライザ関数を指定するシンボル.

テキストプロパティをデシリアライズするためには、そのテキストプロ パティ用のデシリアライザ関数を与えなくてはならない。具体的には、 **Mtext_prop_deserializer** (p. 58) をキーとし、適切なデシリアライズ関数への ポインタを値とするシンボルプロパティを指定する。

参照:
mtext_deserialize() (p. 57), **MTextPropSerializeFunc** (p. 50)

2.10 データベース

m17n データベースにとそれに関する API.

型定義

- typedef struct **MDatabase MDatabase**
  データベースの型宣言.

関数

- **MDatabase** ∗ **mdatabase_find** (**MSymbol** tag0, **MSymbol** tag1, **MSymbol** tag2, **MSymbol** tag3)
  データベース中のデータを探す.
- **MPlist** ∗ **mdatabase_list** (**MSymbol** tag0, **MSymbol** tag1, **MSymbol** tag2, **MSymbol** tag3)
  m17n **データベースのデータリストを返す**.
- **MDatabase** ∗ **mdatabase_define** (**MSymbol** tag0, **MSymbol** tag1, **MSymbol** tag2, **MSymbol** tag3, void ∗(∗loader)(**MSymbol** ∗, void ∗), void ∗extra_info)
  m17n **データベースのデータを定義する**.
- void ∗ **mdatabase_load** (**MDatabase** ∗mdb)
  データベースからデータをロードする.
- **MSymbol** ∗ **mdatabase_tag** (**MDatabase** ∗mdb)
  データのタグを得る.

変数

- char ∗ **mdatabase_dir**
  アプリケーション固有のデータ用ディレクトリ.

2.10.1 説明

m17n データベースにとそれに関する API.

m17n ライブラリは必要に応じて動的に *m17n* **データベース** から情報を取得する。またアプリケーションプログラムも、独自のデータを m17n データベースに追加し、それを動的に取得することができる。 アプリケーションプログラムが独自のデータを追加・取得するには、変数 **mdatabase_dir** (p. 61) にそのアプリケーション固有のディレクトリをセットし、 その中にデータを格納する。ユーザがそのデータをオーバーライトしたい ときは、環境変数 "M17NDIR" で指定されるディレクトリ（指定されていな いときは "∼/.m17n.d" というディレクトリ）に別のデータを置く。

m17n データベースには複数の多様なデータが含まれており、各データは TAG0, TAG1, TAG2, TAG3（すべてシンボル）の４つのタグによって識別される。

TAG0 によって、データベース内のデータのタイプは次のように指定される。

- TAG0 が **Mchar_table** (p. 32) であるデータは *chartable* **タイプ** と呼ばれ、各文字に関する情報を提供する。この場合 TAG1 は情報の種類を指定するシンボルであり、**Msymbol** (p. 17), **Minteger** (p. 23), **Mstring** (p. 17), **Mtext** (p. 23), **Mplist** (p. 23) のいずれかである。TAG2 と TAG3 は任意のシンボルでよい。

- TAG0 が **Mcharset** (p. 70) であるデータは *charset* **タイプ** と呼ばれ、文字セット用のデコード／エンコードマップを提供する。この場合 TAG1 は文字セットのシンボルでなければならない。TAG2 と TAG3 は任意のシンボルでよい。

- TAG0 が **Mchar_table** (p. 32) でも **Mcharset** (p. 70) でもない場合、そのデータは *plist* **タイプ** である。詳細に関しては関数 **mdatabase_load()** (p. 61) の説明を参照のこと。この場合 TAG1、TAG2、TAG3 は任意のシンボルでよい。

特定のタグを持つデータベースを <TAG0, TAG1, TAG2, TAG3> という形式で表す。

アプリケーションプログラムは、まず関数 **mdatabase_find()** (p. 60) を使ってデータベースに関する情報を保持するオブジェクト（**MDatabase** (p. 60) 型）へのポインタを得る。それに成功したら、**mdatabase_load()** (p. 61) によって実際にデータベースをロードする。構造体 **MDatabase** (p. 60) 自身がどう実装されているかは、アプリケーションプログラムからは見えない。

2.10.2 型定義

2.10.2.1 typedef struct MDatabase MDatabase

データベースの型宣言.

MDatabase (p. 60) 型はデータベースオブジェクト用の構造体である。 内部構造はアプリケーションプログラムからは見えない。

2.10.3 関数

2.10.3.1 MDatabase* mdatabase_find (MSymbol *tag0*, MSymbol *tag1*, MSymbol *tag2*, MSymbol *tag3*)

データベース中のデータを探す.

関数 **mdatabase_find()** (p. 60) は、 m17n 言語情報ベース中で **tag0** から **tag3** までのタグを持つデータを探し、それへのポインタを返す。そのようなデータがなければ NULL を返す。

2.10.3.2 MPlist* mdatabase_list (MSymbol *tag0*, MSymbol *tag1*, MSymbol *tag2*, MSymbol *tag3*)

m17n データベースのデータリストを返す.

関数 **mdatabase_list()** (p. 60) は m17n データベース中から **tag0** から **tag3** までのタグを持つデータを探し、そのリストを plist として返す。 **tagn** が **Mnil** (p. 16) であった場合には、任意のタグにマッチするワイルドカードとして取り扱われる。返される plist の各要素はキー として **Mt** (p. 17) を、値として **MDatabase** (p. 60) 型へのポインタを持つ。

2.10.3.3 MDatabase* mdatabase_define (MSymbol *tag0*, MSymbol *tag1*, MSymbol *tag2*, MSymbol *tag3*, void *(*)(MSymbol *, void *) *loader*, void * *extra_info*)

m17n データベースのデータを定義する.

関数 **mdatabase_define()** (p. 60) は **tag0** から **tag3** までのタグおよび付加情報 **extra_info** を持つデータを定義する。

loader はそのデータのロードに用いられる関数へのポインタである。この関数は **mdatabase_load()** (p. 61) から **tags** と **extra_info** という二つの引数付きで呼び出される。ここで **tags** は **tag0** から **tag3** までの配列である。

もし **loader** が NULL なら、m17n ライブラリ標準のローダが使われる。この場合には **extra_info** はデータを含むファイル名でなくてはならない。

戻り値:

処理に成功すれば **mdatabase_define()** (p. 60) は定義されたデータベースへのポインタを返す。このポインタは関数 **mdatabase_load()** (p. 61) の引数として用いることができる。そうでなければ NULL を返す。

参照:

mdatabase_load() (p. 61), **mdatabase_define()** (p. 60)

2.10.3.4 void* mdatabase_load (MDatabase * *mdb*)

データベースからデータをロードする.

関数 **mdatabase_load()** (p. 61) は **mdb** が指すデータをロードし、その中身を返す。返されるものはデータのタイプによって異なる。

データが *plist* **タイプ** ならば、 *plist* へのポインタを返す。

データが *chartable* **タイプ** ならば文字テーブルを返す。 文字テーブルのデフォルト値は、データの第 2 タグによって以下のように決まる。

- タグが **Msymbol** (p. 17) なら、デフォルト値は **Mnil** (p. 16)
- タグが **Minteger** (p. 23) なら、デフォルト値は -1
- それ以外なら、デフォルト値は NULL

データが *charset* **タイプ** ならば長さ 2 の plist を返す（キーは共に **Mt** (p. 17) ）。 最初の要素の値はコードポイントを対応する文字コードにマップする整数の配列である。 2 番目の要素の値は逆のマップをする文字テーブルである。 この文字セットは予め定義されていなければならない。

参照:

mdatabase_load() (p. 61), **mdatabase_define()** (p. 60)

2.10.3.5 MSymbol* mdatabase_tag (MDatabase * *mdb*)

データのタグを得る.

関数 **mdatabase_tag()** (p. 61) は、データ **mdb** のタグ（シンボル）の配列を返す。配列の長さは 4 である。

2.10.4 変数

2.10.4.1 char* mdatabase_dir

アプリケーション固有のデータ用ディレクトリ.

アプリケーションプログラムが、そのプログラム固有のデータや m17n データベースを上書きするデータを提供する場合には、マクロ **M17N_INIT()** (p. 7) を呼ぶ前にこの変数をデータファイルを含むディレクトリ名にセットしなくてはならない。ディレクトリには "mdb.dir" ファイルをおくことができる。その"mdb.dir" ファイルには、 **mdbDir(5)** (p. 207) で説明されているフォーマットでデータ定義のリストを記述する。

デフォルトの値は NULL である。

2.11 シェル API

libm17n.so が提供する API

モジュール

- **文字セット**
  文字セットオブジェクトとそれに関する *API.*
- **コード変換**
  コード系オブジェクトとそれに関する *API.*
- **ロケール**
  ロケールオブジェクトとそれに関する *API.*
- **入力メソッド (基本部分)**
  入力メソッド用 *API.*

2.11.1 説明

libm17n.so が提供する API

2.12 文字セット

文字セットオブジェクトとそれに関する API.

変数: 文字セットを表現する定義済みシンボル.

以下の各シンボルは、定義済み文字セットを表現する。

- **MSymbol Mcharset_ascii**

  ASCII **文字セットを表現するシンボル**.

- **MSymbol Mcharset_iso_8859_1**

  ISO/IEC 8859-1:1998 **文字セットを表現するシンボル**.

- **MSymbol Mcharset_unicode**

  Unicode **文字セットを表現するシンボル**.

- **MSymbol Mcharset_m17n**

  全文字を含む文字セットを表現するシンボル.

- **MSymbol Mcharset_binary**

  正しくデコードできない文字の文字セットを表現するシンボル.

変数: mchar_define_charset 用のパラメータ・キー

これらは、関数 **mchar_define_charset()** (p. 65) 用のパラメータ・キーとして使われるシンボルである。 詳しくはこの関数の解説を参照のこと。

- **MSymbol Mmethod**
- **MSymbol Mdimension**
- **MSymbol Mmin_range**
- **MSymbol Mmax_range**
- **MSymbol Mmin_code**
- **MSymbol Mmax_code**
- **MSymbol Mascii_compatible**
- **MSymbol Mfinal_byte**
- **MSymbol Mrevision**
- **MSymbol Mmin_char**
- **MSymbol Mmapfile**
- **MSymbol Mparents**
- **MSymbol Msubset_offset**
- **MSymbol Mdefine_coding**
- **MSymbol Maliases**

変数: 文字セットのメソッド指定に使われるシンボル

これらは、文字セットの **メソッド** を指定するための定義済みシンボルであり、文字セットの **Mmethod** パラメータの値となることができる。 この値は関数 **mchar_define_charset()** (p. 65) の引数として使われる。

メソッドとは、コードポイントと文字コードを相互変換する際の方式のことである。 詳しくは関数 **mchar_define_charset()** (p. 65) の解説を参照のこと。

- **MSymbol Moffset**
  オフセット型のメソッドを示すシンボル.
- **MSymbol Mmap**
  マップ型のメソッドを示すシンボル.
- **MSymbol Munify**
  ユニファイ型のメソッドを示すシンボル.
- **MSymbol Msubset**
  サブセット型のメソッドを示すシンボル.
- **MSymbol Msuperset**
  スーパーセット型のメソッドを示すシンボル.

マクロ定義

- #define **MCHAR_INVALID_CODE**
  無効なコードポイント.

関数

- **MSymbol mchar_define_charset** (const char ∗name, **MPlist** ∗plist)
  文字セットを定義する.
- **MSymbol mchar_resolve_charset** (**MSymbol** symbol)
  文字セット名を解決する.
- int **mchar_list_charset** (**MSymbol** ∗∗symbols)
  文字セットを表わすシンボルを列挙する.
- int **mchar_decode** (**MSymbol** charset_name, unsigned code)
  コードポイントをデコードする.
- unsigned **mchar_encode** (**MSymbol** charset_name, int c)
  文字コードをエンコードする.
- int **mchar_map_charset** (**MSymbol** charset_name, void(∗func)(int from, int to, void ∗arg), void ∗func_arg)
  指定した文字セットのすべての文字に対して関数を呼ぶ.

変数

- **MSymbol Mcharset**

  シンボル `Mcharset`.

2.12.1 説明

文字セットオブジェクトとそれに関する API.

m17n ライブラリは、符号化文字集合 (CCS) を **文字セット** と呼ぶオブジェクトで表現する。 m17n ライブラリは多くの符号化文字集合をあらかじめサポートしているし、アプリケーションプログラムが独自に文字セットを追加することも可能である。 一つの文字は複数の文字セットに属してもよい。

m17n ライブラリは、以下の概念を区別している:

- **コードポイント** とは、CCS がその中の個々の文字に対して定義する数値である。 コードポイントは連続しているとは限らない。コードポイントは `unsigned` 型によって表される。無効なコードポイントはマクロ `MCHAR_INVALID_CODE` で表される。

- **文字インデックス** とは、CCS 内で各文字に割り当てられる正規化されたインデックスである。 文字インデックスが N の文字は、CCS 中の全文字をコードポイント順に並べたときに N 番目に現われる。 CCS 中の文字インデックスは連続しており、0 から始まる。

- **文字コード** とは、m17n ライブラリ内における文字の内部表現であり、21 ビット以上の長さを持つ符合付き整数である。

各文字セットオブジェクトは、その文字セットに属する文字のコードポイントと文字コードとの間の変換を規定する。 コードポイントから文字コードへの変換を **デコード** と呼び、文字コードからコードポイントへの変換を **エンコード** と呼ぶ。

2.12.2 マクロ定義

2.12.2.1 #define MCHAR_INVALID_CODE

無効なコードポイント.

マクロ **MCHAR_INVALID_CODE** (p. 65) は無効なコードポイントを示す。

2.12.3 関数

2.12.3.1 MSymbol mchar_define_charset (const char * *name*, MPlist * *plist*)

文字セットを定義する.

関数 **mchar_define_charset()** (p. 65) は新しい文字セットを定義し、それを **name** という名前を持つシンボル経由でアクセスできるようにする。 **plist** は定義される文字セットのパラメータを以下のように指定する。

- キーが **Mmethod** で値がシンボルの時

  値は、**Moffset** (p. 69), **Mmap** (p. 69) (デフォルト値), **Munify** (p. 69), **Msubset** (p. 70), **Msuperset** (p. 70) のいずれかであり、文字セットのコードポイントをデコード／エンコードする際のメソッドを指定する。

- キーが **Mdimension** で値が整数値の時

  値は、1 (デフォルト値), 2, 3, 4 のいずれかであり、文字セットのコードポイントの次元である。

- キーが **Mmin_range** で値が非負整数値の時

  値はコードポイントの最小の値である。すなわち、この値の N 番目のバイトはこの文字セットのコードポイントの N 番目のバイトの最小のものとなる。 デフォルト値は 0 。

- キーが **Mmax_range** で値が非負整数値の時

  値はコードポイントの最大の値である。すなわち、この値の N 番目のバイトはこの文字セットのコードポイントの N 番目のバイトの最大のものとなる。 デフォルト値は、コードポイントの次元が 1, 2, 3, 4 の時、それぞれ 0xFF, 0xFFFF, 0xFFFFFF, 0xFFFFFFFF 。

- キーが **Mmin_code** で値が非負整数値の時

  値はこの文字セットの最小のコードポイントである。デフォルト値は **Mmin_range** の値。

- キーが **Mmax_code** で値が非負整数値の時

  値はこの文字セットの最大のコードポイントである。デフォルト値は **Mmax_range** の値。

- キーが **Mascii_compatible** で値がシンボルの時

  値はこの文字セットが ASCII 互換であるかどうかを示す。デフォルト値の **Mnil** (p. 16) であれば互換ではなく、それ以外の場合は互換である。

- キーが **Mfinal_byte** で値が整数値の時

  値はこの文字セットの The International Registry に登録されている **終端バイト** であり、0 (デフォルト値) であるか 32..127 である。0 は登録されていないことを意味する。

- キーが **Mrevision** で値が整数値の時

  値は The International Registry に登録されている *revision number* であり、0..127 である。 文字セットが登録されていない場合にはこの値は無視される。 0 は revision number が存在しないことを意味する。

- キーが **Mmin_char** で値が整数値の時

  値はこの文字セットの最小の文字コードである。デフォルト値は 0 。

- キーが **Mmapfile** で値が M-text の時

  メソッドが **Mmap** (p. 69) か **Munify** (p. 69) の時、関数 **mdatabase_define()** (p. 60) をこの値を引数 **extra_info** として呼ぶことによって、マッピングに関するデータが m17n データベースに追加される。 すなわち、この値はデータファイルの名前である。

  そうでなければ、このパラメータは無視される。

- キーが **Mparents** で値が plist の時

  メソッドが **Msubset** (p. 70) ならば、値は長さ 1 の plist であり、その値はこの文字セットの上位集合となる文字セットを示すシンボルである。

  メソッドが **Msuperset** (p. 70) ならば、値は長さ 8 以下の plist であり、それらの値はこの文字セットの下位集合である文字セットを示すシンボルである。

  そうでなければ、このパラメータは無視される。

- キーが **Mdefine_coding** で値がシンボルの時

  文字セットの次元が 1 ならば、値が **Mnil** (p. 16) 以外の場合に **Mcharset** (p. 70) 型 で同じ名前を持つコード系を定義する。

  そうでなければ、このパラメータは無視される。

戻り値:

処理が成功すれば、mchar_define_charset() は **name** という名前のシンボルを返す。そうでなければ **Mnil** (p. 16) を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_CHARSET

2.12.3.2 MSymbol mchar_resolve_charset (MSymbol *symbol*)

文字セット名を解決する.

関数 **mchar_resolve_charset()** (p. 67) は **symbol** が文字セットを示していればそれを返す。

そうでなければ、**symbol** を文字セット名として正規化し、それが文字セットを示していていれば正規化したものを返す。 そうでなければ、**Mnil** (p. 16) を返す。

2.12.3.3 int mchar_list_charset (MSymbol ** *symbols*)

文字セットを表わすシンボルを列挙する.

関数 mchar_list_charsets() は、文字セットを示すシンボルを並べた配列を作り、**symbols** でポイントされた場所にこの配列へのポインタを置き、配列の長さを返す。

2.12.3.4 int mchar_decode (MSymbol *charset_name*, unsigned *code*)

コードポイントをデコードする.

関数 **mchar_decode()** (p. 67) は、シンボル **charset_name** で示される文字セット内の **code** というコードポイントをデコードして文字コードを得る。

戻り値:
デコードが成功すれば、mchar_decode() はデコードされた文字コードを返す。 そうでなければ -1 を返す。

参照:
mchar_encode() (p. 67)

2.12.3.5 unsigned mchar_encode (MSymbol *charset_name*, int *c*)

文字コードをエンコードする.

関数 **mchar_encode()** (p. 67) は、文字コード **c** をエンコードしてシンボル **charset_name** で示される文字セット内におけるコードポイントを得る。

戻り値:
エンコードが成功すれば、mchar_encode() はエンードされたコードポイントを返す。 そうでなければ **MCHAR_INVALID_CODE** (p. 65) を返す。

参照:
mchar_decode() (p. 67)

2.12.3.6 int mchar_map_charset (MSymbol *charset_name*, void(*)(int from, int to, void *arg) *func*, void * *func_arg*)

指定した文字セットのすべての文字に対して関数を呼ぶ.

関数 mcharset_map_chars() は **charset_name** という名前を持つ文字セット中のすべての文字に対して **func** を呼ぶ。 呼び出しは一文字毎ではなく、連続した文字のまとまり単位で行なわれる。

関数 **func** には **from**, **to**, **arg** の 3 引数が渡される。**from** と **to** は **charset** 中の文字コードの範囲を指定する。**arg** は **func_arg** と同じである。

戻り値:

処理に成功すれば mcharset_map_chars() は 0 を返す。 そうでなければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:

`MERROR_CHARSET`

2.12.4 変数

2.12.4.1 MSymbol Mcharset_ascii

ASCII 文字セットを表現するシンボル.

シンボル **Mcharset_ascii** (p. 68) は `"ascii"` という名前を持ち、 ISO 646, USA Version X3.4-1968 (ISO-IR-6) 文字セットを表現する。

2.12.4.2 MSymbol Mcharset_iso_8859_1

ISO/IEC 8859-1:1998 文字セットを表現するシンボル.

シンボル **Mcharset_iso_8859_1** (p. 68) は `"iso-8859-1"` という名前を持ち、ISO/IEC 8859-1:1998 文字セットを表現する。

2.12.4.3 MSymbol Mcharset_unicode

Unicode 文字セットを表現するシンボル.

シンボル **Mcharset_unicode** (p. 68) は `"unicode"` という名前を持ち、Unicode 文字セットを表現する。

2.12.4.4 MSymbol Mcharset_m17n

全文字を含む文字セットを表現するシンボル.

シンボル **Mcharset_m17n** (p. 68) は `"m17n"` という名前を持ち、 m17n ライブラリが扱う全ての文字を含む文字セットを表現する。

2.12.4.5 MSymbol Mcharset_binary

正しくデコードできない文字の文字セットを表現するシンボル.

シンボル **Mcharset_binary** (p. 68) は `"binary"` という名前を持ち、偽の (fake) 文字セットを表現する。デコード関数は、M-text のテキストプロパティとして、無効なバイト（シークエンス）に遭遇した位置を付加する。

詳細は **コード変換** (p. 71) 参照のこと。

2.12.4.6 MSymbol Mmethod

2.12.4.7 MSymbol Mdimension

2.12.4.8 MSymbol Mmin_range

2.12.4.9 MSymbol Mmax_range

2.12.4.10 MSymbol Mmin_code

2.12.4.11 MSymbol Mmax_code

2.12.4.12 MSymbol Mascii_compatible

2.12.4.13 MSymbol Mfinal_byte

2.12.4.14 MSymbol Mrevision

2.12.4.15 MSymbol Mmin_char

2.12.4.16 MSymbol Mmapfile

2.12.4.17 MSymbol Mparents

2.12.4.18 MSymbol Msubset_offset

2.12.4.19 MSymbol Mdefine_coding

2.12.4.20 MSymbol Maliases

2.12.4.21 MSymbol Moffset

オフセット型のメソッドを示すシンボル.

シンボル **Moffset** (p. 69) は "offset" という名前を持ち、文字セットの **Mmethod** パラメータの値として用いられた場合には、コードポイントと文字セットの文字コードの間の変換が以下の式に従って行われることを意味する。

```
文字コード = コードポイント - MIN-CODE + MIN-CHAR
```

ここで、MIN-CODE は文字セットの **Mmin_code** パラメータの値であり、MIN-CHAR は **Mmin_char** パラメータの値である。

2.12.4.22 MSymbol Mmap

マップ型のメソッドを示すシンボル.

シンボル **Mmap** (p. 69) は "map" という名前を持ち、文字セットの **Mmethod** パラメータの値として用いられた場合には、コードポイントと文字セットの文字コードの間の変換がマップを参照することによって行われることを意味する。 マップは **Mmapfile** パラメータとして与えなければならない。

2.12.4.23 MSymbol Munify

ユニファイ型のメソッドを示すシンボル.

シンボル **Munify** (p. 69) は "unify" という名前を持ち、文字セットの **Mmethod** パラメータの値として用いられた場合には、コードポイントと文字セットの文字コードの間の変換が、マップの参照とオフセットの組み合わせによって行われることを意味する。 マップは **Mmapfile** パラメータとして与えなければならない。 この種の各文字セットには、全文字に対して連続するコードスペースがそれぞれ割り当てられる。

コードポイントがマップに含まれていれば、変換はマップ参照によって行われる。 そうでなければ、以下の式に従う。

```
CHARACTER-CODE = CODE-POINT - MIN-CODE + LOWEST-CHAR-CODE
```

ここで、MIN-CODE は文字セットの **Mmin_code** パラメータの値であり、 LOWEST-CHAR-CODE は割り当てられたコードスペースの最も小さい文字コードである。

2.12.4.24 MSymbol Msubset

サブセット型のメソッドを示すシンボル.

シンボル **Msubset** (p. 70) は "subset" という名前を持ち、文字セットの **Mmethod** パラメータの値として用いられた場合には、この文字セットが別の文字セット（親文字セット）の部分集合であることを意味する。 親文字セットは **Mparents** パラメータによって与えられなくてはならない。 コードポイントと文字セットの文字コードの間の変換は、概念的には以下の式に従う。

```
CHARACTER-CODE = PARENT-CODE (CODE-POINT) + SUBSET-OFFSET
```

ここで PARENT-CODE は CODE-POINT の親文字セット中での文字コードを返す擬関数であり、 SUBSET-OFFSET は **Msubset_offset** パラメータで与えられる値である。

2.12.4.25 MSymbol Msuperset

スーパーセット型のメソッドを示すシンボル.

シンボル **Msuperset** (p. 70) は "superset" という名前を持ち、文字セットの **Mmethod** パラメータの値として用いられた場合には、この文字セットが別の文字セット（親文字セット）の上位集合であることを意味する。 親文字セットは **Mparents** パラメータによって与えられなくてはならない。

2.12.4.26 MSymbol Mcharset

シンボル Mcharset.

デコードされた M-text は、キーが Mcharset であるようなテキストプロパティを持つ。 シンボル Mcharset は "charset" という名前を持つ。

2.13 コード変換

コード系オブジェクトとそれに関する API.

データ構造

- struct **MConverter**
  コード変換に用いられる構造体.
- struct **MCodingInfoISO2022**
  MCODING_TYPE_ISO_2022 (p. 76) **タイプのコード系で必要な付加情報用構造体**.
- struct **MCodingInfoUTF**
  MCODING_TYPE_UTF (p. 75) **タイプのコード系で必要な付加情報用の構造体**.

変数: 定義済みコード系を指定するためのシンボル

- **MSymbol Mcoding_us_ascii**
  US-ASCII **コード系のシンボル**.
- **MSymbol Mcoding_iso_8859_1**
  ISO-8859-1 **コード系のシンボル**.
- **MSymbol Mcoding_utf_8**
  UTF-8 **コード系のシンボル**.
- **MSymbol Mcoding_utf_8_full**
  UTF-8-FULL **コード系のシンボル**.
- **MSymbol Mcoding_utf_16**
  UTF-16 **コード系のシンボル**.
- **MSymbol Mcoding_utf_16be**
  UTF-16BE **コード系のシンボル**.
- **MSymbol Mcoding_utf_16le**
  UTF-16LE **コード系のシンボル**.
- **MSymbol Mcoding_utf_32**
  UTF-32 **コード系のシンボル**.
- **MSymbol Mcoding_utf_32be**
  UTF-32BE **コード系のシンボル**.
- **MSymbol Mcoding_utf_32le**
  UTF-32LE **コード系のシンボル**.
- **MSymbol Mcoding_sjis**
  SJIS **コード系のシンボル**.

変数: mconv_define_coding() 用パラメータキー

- **MSymbol Mtype**
- **MSymbol Mcharsets**
- **MSymbol Mflags**
- **MSymbol Mdesignation**
- **MSymbol Minvocation**
- **MSymbol Mcode_unit**
- **MSymbol Mbom**
- **MSymbol Mlittle_endian**

変数： コード系のタイプを示すシンボル.

- **MSymbol Mutf**
- **MSymbol Miso_2022**

変数： パラメータ Mflags の値となり得るシンボル.

関数 **mconv_define_coding()** (p. 76) の引数として用いられるコード系のパラメータ **Mflags** の値となり得るシンボル。(詳細は **mconv_define_coding()** (p. 76) 参照)。

- **MSymbol Mreset_at_eol**
- **MSymbol Mreset_at_cntl**
- **MSymbol Meight_bit**
- **MSymbol Mlong_form**
- **MSymbol Mdesignation_g0**
- **MSymbol Mdesignation_g1**
- **MSymbol Mdesignation_ctext**
- **MSymbol Mdesignation_ctext_ext**
- **MSymbol Mlocking_shift**
- **MSymbol Msingle_shift**
- **MSymbol Msingle_shift_7**
- **MSymbol Meuc_tw_shift**
- **MSymbol Miso_6429**
- **MSymbol Mrevision_number**
- **MSymbol Mfull_support**

変数: その他

ほかの変数。

- **MSymbol Mmaybe**

  *"maybe"*という名前を持つシンボル.

- **MSymbol Mcoding**

  シンボル `Mcoding`.

列挙型

- enum **MConversionResult** {
  MCONVERSION_RESULT_SUCCESS,
  MCONVERSION_RESULT_INVALID_BYTE,
  MCONVERSION_RESULT_INVALID_CHAR,
  MCONVERSION_RESULT_INSUFFICIENT_SRC,
  MCONVERSION_RESULT_INSUFFICIENT_DST,
  MCONVERSION_RESULT_IO_ERROR }
  コード変換の結果を示すコード.

- enum **MCodingType** {
  MCODING_TYPE_CHARSET,
  MCODING_TYPE_UTF,
  MCODING_TYPE_ISO_2022,
  MCODING_TYPE_MISC }
  コード系のタイプ.

- enum **MCodingFlagISO2022** {
  MCODING_ISO_RESET_AT_EOL = 0x1,
  MCODING_ISO_RESET_AT_CNTL = 0x2,
  MCODING_ISO_EIGHT_BIT = 0x4,
  MCODING_ISO_LONG_FORM = 0x8,
  MCODING_ISO_DESIGNATION_G0 = 0x10,
  MCODING_ISO_DESIGNATION_G1 = 0x20,
  MCODING_ISO_DESIGNATION_CTEXT = 0x40,
  MCODING_ISO_DESIGNATION_CTEXT_EXT = 0x80,
  MCODING_ISO_LOCKING_SHIFT = 0x100,
  MCODING_ISO_SINGLE_SHIFT = 0x200,
  MCODING_ISO_SINGLE_SHIFT_7 = 0x400,
  MCODING_ISO_EUC_TW_SHIFT = 0x800,
  MCODING_ISO_ISO6429 = 0x1000,
  MCODING_ISO_REVISION_NUMBER = 0x2000,
  MCODING_ISO_FULL_SUPPORT = 0x3000,
  MCODING_ISO_FLAG_MAX }
  MCODING_TYPE_ISO_2022 **タイプのコード系の詳細を表わすビットマスク**.

関数

- **MSymbol mconv_define_coding** (const char ∗name, **MPlist** ∗plist, int(∗resetter)(**MConverter** ∗), int(∗decoder)(const unsigned char ∗, int, **MText** ∗, **MConverter** ∗), int(∗encoder)(**MText** ∗, int, int, unsigned char ∗, int, **MConverter** ∗), void ∗extra_info)
  コード系を定義する.

- **MSymbol mconv_resolve_coding** (**MSymbol** symbol)

コード系の名前を解決する.

- int **mconv_list_codings** (**MSymbol** ∗∗symbols)
  コード系を表わすシンボルを列挙する.

- **MConverter** ∗ **mconv_buffer_converter** (**MSymbol** name, const unsigned char ∗buf, int n)
  バッファに結び付けられたコードコンバータを作る.

- **MConverter** ∗ **mconv_stream_converter** (**MSymbol** name, FILE ∗fp)
  ストリームに結び付けられたコードコンバータを作る.

- int **mconv_reset_converter** (**MConverter** ∗converter)
  コードコンバータをリセットする.

- void **mconv_free_converter** (**MConverter** ∗converter)
  コードコンバータを解放する.

- **MConverter** ∗ **mconv_rebind_buffer** (**MConverter** ∗converter, const unsigned char ∗buf, int n)
  コードコンバータにバッファ領域を結び付ける.

- **MConverter** ∗ **mconv_rebind_stream** (**MConverter** ∗converter, FILE ∗fp)
  コードコンバータにストリームを結び付ける.

- **MText** ∗ **mconv_decode** (**MConverter** ∗converter, **MText** ∗mt)
  バイト列を *M-text* にデコードする.

- **MText** ∗ **mconv_decode_buffer** (**MSymbol** name, const unsigned char ∗buf, int n)
  コード系に基づいてバッファ領域をデコードする.

- **MText** ∗ **mconv_decode_stream** (**MSymbol** name, FILE ∗fp)
  コード系に基づいてストリーム入力をデコードする.

- int **mconv_encode** (**MConverter** ∗converter, **MText** ∗mt)
  M-text をバイト列にエンコードする.

- int **mconv_encode_range** (**MConverter** ∗converter, **MText** ∗mt, int from, int to)
  M-text の一部をバイト列にエンコードする.

- int **mconv_encode_buffer** (**MSymbol** name, **MText** ∗mt, unsigned char ∗buf, int n)
  M-text をエンコードしてバッファ領域に書き込む.

- int **mconv_encode_stream** (**MSymbol** name, **MText** ∗mt, FILE ∗fp)
  M-text をエンコードしてストリームに書き込む.

- int **mconv_getc** (**MConverter** ∗converter)
  コードコンバータ経由で一文字を読みこむ.

- int **mconv_ungetc** (**MConverter** ∗converter, int c)
  コードコンバータに一文字戻す.

- int **mconv_putc** (**MConverter** ∗converter, int c)
  コードコンバータを経由して一文字書き出す.

- **MText** ∗ **mconv_gets** (**MConverter** ∗converter, **MText** ∗mt)

  コードコンバータを使って一行読み込む.

2.13.1 説明

コード系オブジェクトとそれに関する API.

m17n ライブラリは、符号化文字集合 (coded character set; CCS) の文字符合化方式 (character encoding scheme; CES) を **コード系** と呼ぶオブジェクトで表現する。 アプリケーションプログラムは独自にコード系を追加することもできる。

コードポイントから文字コードへの変換を **エンコード** と呼び、文字コードからコードポイントへの変換を **デコード** と呼ぶ。

アプリケーションプログラムは、指定されたコード系でバイト列をデコードすることによって M-text を得ることができる。また逆に、指定されたコード系で M-text をエンコードしすることによってバイト列を得ることができる。

2.13.2 列挙型

2.13.2.1 enum MConversionResult

コード変換の結果を示すコード.

これらの値のうち一つが `MConverter->result` に設定される。

列挙型の値:

MCONVERSION_RESULT_SUCCESS コード変換は成功.

MCONVERSION_RESULT_INVALID_BYTE デコード時、ソースに不正なバイトが含まれている.

MCONVERSION_RESULT_INVALID_CHAR エンコード時、指定のコード系でエンコードできない文字がソースに含まれている.

MCONVERSION_RESULT_INSUFFICIENT_SRC デコード時、不完全なバイト列でソースが終わっている。

MCONVERSION_RESULT_INSUFFICIENT_DST エンコード時、結果を格納する領域が短かすぎる。

MCONVERSION_RESULT_IO_ERROR コード変換中に I/O エラーが起こった。

2.13.2.2 enum MCodingType

コード系のタイプ.

列挙型の値:

MCODING_TYPE_CHARSET このタイプのコード系は文字セットを直接サポートする。 各文字セットの次元とは、その文字セットで一文字を表現するために必要なバイト数であり、バイト列は文字のコードポイントを直接表す。 m17n ライブラリはこのタイプ用のデフォルトのエンコード／デコードルーティンを提供する。

MCODING_TYPE_UTF このタイプのコード系は、UTF 系 (UTF-8, UTF-16, UTF-32) のバイト列をサポートする。 m17n ライブラリはこのタイプ用のデフォルトのエンコード／デコードルーティンを提供する。

MCODING_TYPE_ISO_2022 このタイプのコード系は、ISO-2022 系のバイト列をサポートする。 各コード系の構造の詳細は **MCodingInfoISO2022** (p. 158) で指定される。 m17n ライブラリはこのタイプ用のデフォルトのエンコード／デコードルーティンを提供する。

MCODING_TYPE_MISC このタイプのコード系は、その他の構造のバイト列のためのものである。 m17n ライブラリはこのタイプ用のエンコード／デコードルーティンを提供しないので、アプリケーションプログラム側で準備する必要がある。

2.13.2.3 enum MCodingFlagISO2022

MCODING_TYPE_ISO_2022 タイプのコード系の詳細を表わすビットマスク.

列挙型の値:

MCODING_ISO_RESET_AT_EOL エンコードの際、行末で呼び出し (invocation) と指示 (designation) の状態を初期値に戻す。

MCODING_ISO_RESET_AT_CNTL エンコードの際、すべての制御文字の前で、呼び出し (invocation) と指示 (designation) の状態を初期値に戻す。

MCODING_ISO_EIGHT_BIT 図形文字集合の右側を使う。

MCODING_ISO_LONG_FORM JISX0208-1978, GB2312, JISX0208-1983 の文字集合に対する指示シークエンスとして、非標準の 4 バイト形式を用いる。

MCODING_ISO_DESIGNATION_G0 エンコードの際、特に指定されない限り、文字集合を G0 に指示する。

MCODING_ISO_DESIGNATION_G1 エンコードの際、特に指定されない限り、ASCII 以外の文字集合を G1 に指示する。

MCODING_ISO_DESIGNATION_CTEXT エンコードの際、特に指定されない限り、94 文字集合を G0 に、96 文字集合を G1 に指示する。

MCODING_ISO_DESIGNATION_CTEXT_EXT エンコードの際、ISO-2022 に合致しない文字集合を ESC % / ... でエンコードする。サポートされていない Unicode 文字は ESC % G ... ESC % @ でエンコードする。 デコードの際、これらのエスケープ・シーケンスを解釈する。

MCODING_ISO_LOCKING_SHIFT ロッキングシフトを使う。

MCODING_ISO_SINGLE_SHIFT シングルシフト (SS2 (0x8E or ESC N), SS3 (0x8F or ESC O)) を使う。

MCODING_ISO_SINGLE_SHIFT_7 7 ビットシングルシフト 2 (SS2 (0x19)) を使う。

MCODING_ISO_EUC_TW_SHIFT EUC-TW 風の特別なシフトを使う。

MCODING_ISO_ISO6429 ISO-6429 のエスケープシークエンスで方向を指示する。未実装。

MCODING_ISO_REVISION_NUMBER エンコードの際、文字セットに revision number があればそれを表わすエスケープシークエンスを生成する。

MCODING_ISO_FULL_SUPPORT ISO-2022 の全文字集合をサポートする。

MCODING_ISO_FLAG_MAX

2.13.3 関数

2.13.3.1 MSymbol mconv_define_coding (const char * *name*, MPlist * *plist*, int(*)(MConverter *) *resetter*, int(*)(const unsigned char *, int, MText *, MConverter *) *decoder*, int(*)(MText *, int, int, unsigned char *, int, MConverter *) *encoder*, void * *extra_info*)

コード系を定義する.

関数 **mconv_define_coding()** (p. 76) は、新しいコード系を定義し、それを **name** という名前のシンボル経由でアクセスできるようにする。 **plist** では定義するコード系のパラメータを以下のように指定する。

- キーが `Mtype` で値がシンボルの時

  値はコード系のタイプを表し、**Mcharset**, **Mutf**, **Miso_2022**, **Mnil** (p. 16) のいずれかでなくてはならない。

  タイプが **Mcharset** ならば **extra_info** は無視される。

  タイプが **Mutf** ならば **extra_info** は **MCodingInfoUTF** (p. 159) へのポインタでなくてはならない。

  タイプが **Miso_2022 ならば extra_info** は **MCodingInfoISO2022** (p. 158) へのポインタでなくてはならない。

  タイプが **Mnil** (p. 16) ならば、引数 **resetter**, **decoder**, **encoder** を与えなくてはならない。**extra_info** は無視される。 それ以外の場合にはこれらは `NULL` でよく、 m17n ライブラリが適切なデフォルト値を与える。

- キーが **Mcharsets** で値が plist の時

  値はこのコード系でサポートされる文字セットのリストである。plist のキーは **Msymbol** (p. 17)、値は文字セットを示すシンボルでなくてはならない。

- キーが **Mflags** 値が plist の時

  タイプが **Miso_2022** ならば、この値は, ISO 2022 インタプリタ用の制御フラッグを示す。plist のキーは **Msymbol** (p. 17) であり、値は以下のいずれかである。

  - **Mreset_at_eol**
    このフラグがあれば、図形文字集合の指示や呼出は行末でリセットされて当初の状態に戻る。
  - **Mreset_at_cntl**
    このフラグがあれば、図形文字集合の指示や呼出は制御文字に出会った時点でリセットされて当初の状態に戻る。
  - **Meight_bit**
    このフラグがあれば、図形文字集合の右半面が用いられる。
  - **Mlong_form**
    このフラグがあれば、文字集合 JISX0208.1978, GB2312, JISX0208 を指示する際に over-long エスケープシーケンス (ESC '$' '(' <final_byte>) が用いられる。
  - **Mdesignation_g0**
    このフラグと **Mfull_support** があれば、文字セットリストに現われない文字セットを G0 集合に指示する。
  - **Mdesignation_g1**
    このフラグと **Mfull_support** があれば、文字セットリストに現われない文字セットを G1 集合に指示する。
  - **Mdesignation_ctext**
    このフラグと **Mfull_support** があれば、文字セットリストに現われない文字セットを G0 集合または G1 集合に、コンパウンドテキストの基準にそって指示する。
  - **Mdesignation_ctext_ext**
    このフラグと **Mfull_support** があれば、文字セットリストに現われない文字セットを G0 集合または G1 集合に、あるいは拡張セグメントにコンパウンドテキストの基準にそって指示する。
  - **Mlocking_shift**
    このフラグがあれば、ロッキングシフトを用いる。
  - **Msingle_shift**
    このフラグがあれば、シングルシフトを用いる。
  - **Msingle_shift_7**
    このフラグがあれば、7-bit シングルシフトコード (0x19) を用いる。
  - **Meuc_tw_shift**
    このフラグがあれば、EUC-TW に沿った特別なシフトを用いる。

  - **Miso_6429**
    現時点では用いられていない。
  - **Mrevision_number**
    このフラグがあれば、revision number を持つ文字セットを指示する際に revision number エスケープシークエンスを用いる。
  - **Mfull_support**
    このフラグがあれば、the International Registry に登録されている全文字セットをサポートする。

- キーが **Mdesignation** で値が plist の時

  タイプが **Miso_2022** ならば、値は各文字をどのように指示するかを示す。 plist のキーは **Minteger** (p. 23)、値は集合（graphic register） を示す数字である。N 番目の要素の値は、文字セットリストの N 番目の文字セットに対応する。値が 0..3 であれば、文字セットがすでに G0..G3 に指示 されている。

  値が負 (-4..-1) であれば、初期状態では文字セットがどこにも指示されていないこと、必要な際には G0..G3 のそれぞれに指示することを意味する。

- キーが **Minvocation** で値が plist の時

  タイプが **Miso_2022** ならば、値は各集合をどのように呼び出すかを示す。 plist の長さは 1 ないし 2 である。plist のキーは **Minteger** (p. 23)、値は集合（graphic register) を示す数字である。 最初の要素の値が図形文字集合左半面に呼び出される集合を示す。 plist の長さが 1 ならば、右半面には何も呼び出されない。 そうでければ、２つめの要素の値が図形文字集合右半面に呼び出される集合を示す。

- キーが **Mcode_unit** で値が整数値の時

  タイプが **Mutf** ならば、値はコードユニットのビット長であり、8, 16, 32 のいずれかである。

- キーが **Mbom** で値がシンボルの時

  タイプが **Mutf** でコードユニットのビット長が 16 か 32 ならば、値は BOM (Byte Order Mark) を使用するかどうかを示す。値がデフォルト値の **Mnil** (p. 16) ならば、使用しない。値が **Mmaybe** (p. 87) ならばデコード時に BOM があるかどうかを調べる。それ以外ならば使用する。

- キーが **Mlittle_endian** で値がシンボルの時

  タイプが **Mutf** でコードユニットのビット長が 16 か 32 ならば、値はエンコードが little endian かどうかを示す。値がデフォルト値の **Mnil** (p. 16) ならば big endian であり、そうでなければ little endian である。

resetter はこのコード系用のコンバータを初期状態にリセットする関数へのポインタである。 この関数はコンバータオブジェクトへのポインタという１引数をとる。

decoder はバイト列をこのコード系に従ってデコードする関数へのポインタである。 この関数は以下の４引数をとる。

- デコードするバイト列へのポインタ
- デコードすべきバイト数
- デコード結果の文字を付加する M-text へのポインタ
- コンバータオブジェクトへのポインタ

decoder は成功したときには 0 を、失敗したときには -1 を返さなくてはならない。

encoder は M-text をこのコード系に従ってエンコードする関数へのポインタである。 この関数は以下の６引数をとる。

- エンコードする M-text へのポインタ
- M-text のエンコード開始位置

- M-text のエンコード終了位置
- 生成したバイトを保持するメモリ領域へのポインタ
- メモリ領域のサイズ
- コンバータオブジェクトへのポインタ

encoder は成功したときには 0 を、失敗したときには -1 を返さなくてはならない。

extra_info はコーディグシステムに関する追加情報を含むデータ構造へのポインタである。 このデータ構造の型 **type** に依存する。

戻り値:

処理に成功すれば **mconv_define_coding()** (p. 76) は **name** という名前のシンボルを返す。 エラーが検出された場合は **Mnil** (p. 16) を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:

MERROR_CODING

2.13.3.2 MSymbol mconv_resolve_coding (MSymbol *symbol*)

コード系の名前を解決する.

関数 **mconv_resolve_coding()** (p. 79) は **symbol** がコード系を示していればそれを返す。 そうでなければコード系の名前として **symbol** を正規化し、それがコード系を表していれば正規化した **symbol** を返す。そうでなければ **Mnil** (p. 16) を返す。

2.13.3.3 int mconv_list_codings (MSymbol ** *symbols*)

コード系を表わすシンボルを列挙する.

関数 mchar_list_codings() は、コード系を示すシンボルを並べた配列を作り、 **symbols** でポイントされた場所にこの配列へのポインタを置き、配列の長さを返す。

2.13.3.4 MConverter* mconv_buffer_converter (MSymbol *name*, const unsigned char * *buf*, int *n*)

バッファに結び付けられたコードコンバータを作る.

関数 **mconv_buffer_converter()** (p. 79) は、コード系 **name** 用のコードコンバータを作る。このコードコンバータは、**buf** で示される大きさ **n** バイトのバッファ領域に結び付けられる。 これ以降のデコードおよびエンコードは、このバッファ領域に対して行なわれる。

name は **Mnil** (p. 16) であってもよい。この場合は現在のロケール (LC_CTYPE) に関連付けられたコード系が使われる。

戻り値:

もし処理が成功すれば **mconv_buffer_converter()** (p. 79) は 作成したコードコンバータを返す。 そうでなければ NULL を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:

MERROR_SYMBOL, MERROR_CODING

参照:

mconv_stream_converter() (p. 80)

2.13.3.5 MConverter* mconv_stream_converter (MSymbol *name*, FILE * *fp*)

ストリームに結び付けられたコードコンバータを作る.

関数 **mconv_stream_converter()** (p. 80) は、コード系 **name** 用のコードコンバータを作る。このコードコンバータは、ストリーム **fp** に結び付けられる。 これ以降のデコードおよびエンコードは、このストリームに対して行なわれる。

name は **Mnil** (p. 16) であってもよい。この場合は現在のロケール (LC_CTYPE) に関連付けられたコード系が使われる。

戻り値:
もし処理が成功すれば、mconv_stream_converter() は作成したコードコンバータを返す。そうでなければ NULL を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_SYMBOL, MERROR_CODING

参照:
mconv_buffer_converter() (p. 79)

2.13.3.6 int mconv_reset_converter (MConverter * *converter*)

コードコンバータをリセットする.

関数 **mconv_reset_converter()** (p. 80) はコードコンバータ **converter** を初期状態に戻す。

戻り値:
もし **converter->coding** にリセット用の関数が定義されているならば、 **mconv_reset_converter()** (p. 80) はその関数に **converter** を適用した結果を返し、そうでなければ 0 を返す。

2.13.3.7 void mconv_free_converter (MConverter * *converter*)

コードコンバータを解放する.

関数 **mconv_free_converter()** (p. 80) はコードコンバータ **converter** を解放する。

2.13.3.8 MConverter* mconv_rebind_buffer (MConverter * *converter*, const unsigned char * *buf*, int *n*)

コードコンバータにバッファ領域を結び付ける.

関数 **mconv_rebind_buffer()** (p. 80) は、**buf** によって指された大きさ **n** バイトのバッファ領域をコードコンバータ **converter** に結び付ける。 これ以降のデコードおよびエンコードは、この新たに結び付けられたバッファ領域に対して行なわれるようになる。

戻り値:
この関数は常に **converter** を返す。

参照:
mconv_rebind_stream() (p. 81)

2.13.3.9 MConverter* mconv_rebind_stream (MConverter * *converter*, FILE * *fp*)

コードコンバータにストリームを結び付ける.

関数 **mconv_rebind_stream()** (p. 81) は、ストリーム **fp** をコードコンバータ **converter** に結び付ける。 これ以降のデコードおよびエンコードは、この新たに結び付けられたストリームに対して行なわれるようになる。

戻り値:
この関数は常に **converter** を返す。

参照:
mconv_rebind_buffer() (p. 80)

2.13.3.10 MText* mconv_decode (MConverter * *converter*, MText * *mt*)

バイト列を M-text にデコードする.

関数 **mconv_decode()** (p. 81) は、バイト列をデコードしてその結果を M-text **mt** の末尾に追加する。デコード元のバイト列は、**converter** に現在結び付けられているバッファ領域あるいはストリームから取られる。

戻り値:
もし処理が成功すれば、mconv_decode() は更新された **mt** を返す。 そうでなければ NULL を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_IO, MERROR_CODING

参照:
mconv_rebind_buffer() (p. 80), **mconv_rebind_stream()** (p. 81), **mconv_encode()** (p. 82), **mconv_encode_range()** (p. 82), **mconv_decode_buffer()** (p. 81), **mconv_decode_stream()** (p. 82)

2.13.3.11 MText* mconv_decode_buffer (MSymbol *name*, const unsigned char * *buf*, int *n*)

コード系に基づいてバッファ領域をデコードする.

関数 **mconv_decode_buffer()** (p. 81) は、**buf** によって指された **n** バイトのバッファ領域を、コード系 **name** に基づいてデコードする。 デコードに必要なコードコンバータの作成と解放は自動的に行なわれる。

戻り値:
もし処理が成功すれば、mconv_decode_buffer() は得られた M-text を返す。 そうでなければ NULL を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_IO, MERROR_CODING

参照:
mconv_decode() (p. 81), **mconv_decode_stream()** (p. 82)

2.13.3.12 MText* mconv_decode_stream (MSymbol *name*, FILE * *fp*)

コード系に基づいてストリーム入力をデコードする.

関数 **mconv_decode_stream()** (p. 82) は、ストリーム **fp** から読み込まれるバイト列全体を、コード系 **name** に基づいてデコードする。デコードに必要なコードコンバータの作成と解放は自動的に行なわれる。

戻り値:
もし処理が成功すれば、mconv_decode_stream() は得られた M-text を返す。そうでなければ NULL を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_IO, MERROR_CODING

参照:
mconv_decode() (p. 81), **mconv_decode_buffer()** (p. 81)

2.13.3.13 int mconv_encode (MConverter * *converter*, MText * *mt*)

M-text をバイト列にエンコードする.

関数 **mconv_encode()** (p. 82) は、M-text **mt** をエンコードして、コードコンバータ **converter** に現在結び付けられているバッファ領域あるいはストリームに得られたバイト列を書き込む。

戻り値:
もし処理が成功すれば、mconv_encode() は書き込まれたバイト数を返す。 そうでなければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_IO, MERROR_CODING

参照:
mconv_rebind_buffer() (p. 80), **mconv_rebind_stream()** (p. 81), **mconv_decode()** (p. 81), **mconv_encode_range()** (p. 82)

2.13.3.14 int mconv_encode_range (MConverter * *converter*, MText * *mt*, int *from*, int *to*)

M-text の一部をバイト列にエンコードする.

関数 **mconv_encode_range()** (p. 82) は、M-text **mt** の **from** （**from** 自体も含む）から **to** （**to 自体は含まない**） までの範囲のテキストをエンコードして、コードコンバータ **converter** に現在結び付けられているバッファ領域あるいはストリームに得られたバイト列を書き込む。

戻り値:
もし処理が成功すれば、mconv_encode_range() は書き込まれたバイト数を返す。そうでなければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_RANGE, MERROR_IO, MERROR_CODING

参照:
mconv_rebind_buffer() (p. 80), **mconv_rebind_stream()** (p. 81), **mconv_decode()** (p. 81), **mconv_encode()** (p. 82)

2.13.3.15 int mconv_encode_buffer (MSymbol *name*, MText * *mt*, unsigned char * *buf*, int *n*)

M-text をエンコードしてバッファ領域に書き込む.

関数 **mconv_encode_buffer()** (p. 83) は M-text **mt** をコード系 **name** に基づいてエンコードし、得られたバイト列を **buf** の指すバッファ領域に書き込む。 **n** は書き込む最大バイト数である。 エンコードに必要なコードコンバータの作成と解放は自動的に行なわれる。

戻り値:
もし処理が成功すれば、mconv_encode_buffer() は書き込まれたバイト数を返す。 そうでなければ-1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_IO, MERROR_CODING

参照:
mconv_encode() (p. 82), **mconv_encode_stream()** (p. 83)

2.13.3.16 int mconv_encode_stream (MSymbol *name*, MText * *mt*, FILE * *fp*)

M-text をエンコードしてストリームに書き込む.

関数 **mconv_encode_stream()** (p. 83) は M-text **mt** をコード系 **name** に基づいてエンコードし、得られたバイト列をストリーム **fp** に書き出す。エンコードに必要なコードコンバータの作成と解放は自動的に行なわれる。

戻り値:
もし処理が成功すれば、mconv_encode_stream() は書き込まれたバイト数を返す。そうでなければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_IO, MERROR_CODING

参照:
mconv_encode() (p. 82), **mconv_encode_buffer()** (p. 83), mconv_encode_file()

2.13.3.17 int mconv_getc (MConverter * *converter*)

コードコンバータ経由で一文字を読みこむ.

関数 **mconv_getc()** (p. 83) は、コードコンバータ **converter** に現在結び付けられているバッファ領域あるいはストリームから文字を一つ読み込む。 バイト列のデコードには **converter** のデコーダが用いられる。 **converter** の内部状態は必要に応じて更新される。

戻り値:
処理が成功すれば、mconv_getc() は読み込まれた文字を返す。入力源が EOF に達した場合は、外部変数 **merror_code** (p. 152) を変えずに EOF を返す。エラーが検出された場合は EOF を返し、 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_CODING

参照:
mconv_ungetc() (p. 84), **mconv_putc()** (p. 84), **mconv_gets()** (p. 84)

2.13.3.18 int mconv_ungetc (MConverter * *converter*, int *c*)

コードコンバータに一文字戻す.

関数 **mconv_ungetc()** (p. 84) は、コードコンバータ **converter** に文字 **c** を押し戻す。戻される文字数に制限はない。この後で **mconv_getc()** (p. 83) を呼び出した際には、最後に戻された文字が最初に読まれる。戻された文字は **converter** の内部に蓄えられるだけであり、実際に入力源に書き込まれるわけではない。**converter** の内部状態は必要に応じて更新される。

戻り値:
処理が成功すれば、mconv_ungetc() は **c** を返す。そうでなければ EOF を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_CODING, MERROR_CHAR

参照:
mconv_getc() (p. 83), **mconv_putc()** (p. 84), **mconv_gets()** (p. 84)

2.13.3.19 int mconv_putc (MConverter * *converter*, int *c*)

コードコンバータを経由して一文字書き出す.

関数 **mconv_putc()** (p. 84) は、コードコンバータ **converter** に現在結び付けられているバッファ領域あるいはストリームに文字 **c** を書き出す。文字のエンコードには **converter** のエンコーダが用いられる。実際に書き出されたバイト数は、**converter** のメンバー nbytes にセットされる。**converter** の内部状態は必要に応じて更新される。

戻り値:
処理が成功すれば、mconv_putc() は **c** を返す。エラーが検出された場合は EOF を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_CODING, MERROR_IO, MERROR_CHAR

参照:
mconv_getc() (p. 83), **mconv_ungetc()** (p. 84), **mconv_gets()** (p. 84)

2.13.3.20 MText* mconv_gets (MConverter * *converter*, MText * *mt*)

コードコンバータを使って一行読み込む.

関数 **mconv_gets()** (p. 84) は、コードコンバータ **converter** に現在結び付けられているバッファ領域あるいはストリームから 1 行を読み込む。 バイト列のデコードには **converter** のデコーダが用いられる。デコードされた文字列は M-text **mt** の末尾に追加される。元のバイト列の終端改行文字は追加されない。**converter** の内部状態は必要に応じて更新される。

戻り値:
処理が成功すれば、mconv_gets() は変更された **mt** を返す。もし 1 文字も読まずに EOF に遭遇した場合は、**mt** を変更せずにそのまま返す。エラーが検出された場合は NULL を返し、 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_CODING

参照:
mconv_getc() (p. 83), **mconv_ungetc()** (p. 84), **mconv_putc()** (p. 84)

2.13.4 変数

2.13.4.1 MSymbol Mcoding_us_ascii

US-ASCII コード系のシンボル.

シンボル **Mcoding_us_ascii** (p. 85) は "us-ascii" という名前を持ち、 CES US-ASCII 用のコード系を示す。

2.13.4.2 MSymbol Mcoding_iso_8859_1

ISO-8859-1 コード系のシンボル.

シンボル **Mcoding_iso_8859_1** (p. 85) は "iso-8859-1" という名前を持ち、CES ISO-8859-1 用のコード系を示す。

2.13.4.3 MSymbol Mcoding_utf_8

UTF-8 コード系のシンボル.

シンボル **Mcoding_utf_8** (p. 85) は "utf-8" という名前を持ち、CES UTF-8 用のコード系を示す。

2.13.4.4 MSymbol Mcoding_utf_8_full

UTF-8-FULL コード系のシンボル.

シンボル **Mcoding_utf_8_full** (p. 85) は "utf-8-full" という名前を持ち、"UTF-8" の拡張であるコード系を示す。 このコード系は UTF-8 と同じエンコーディングアルゴリズムを用いるが、対象は Unicode 文字には限定されない。 また m17n ライブラリが扱う全ての文字をエンコードすることができる。

2.13.4.5 MSymbol Mcoding_utf_16

UTF-16 コード系のシンボル.

シンボル **Mcoding_utf_16** (p. 85) は "utf-16" という名前を持ち、 CES UTF-16 (RFC 2279) 用のコード系を示す。

2.13.4.6 MSymbol Mcoding_utf_16be

UTF-16BE コード系のシンボル.

シンボル **Mcoding_utf_16be** (p. 85) は "utf-16be" という名前を持ち、 CES UTF-16BE (RFC 2279) 用のコード系を示す。

2.13.4.7 MSymbol Mcoding_utf_16le

UTF-16LE コード系のシンボル.

シンボル **Mcoding_utf_16le** (p. 85) は "utf-16le" という名前を持ち、 CES UTF-16LE (RFC 2279) 用のコード系を示す。

2.13.4.8 MSymbol Mcoding_utf_32

UTF-32 コード系のシンボル.

シンボル **Mcoding_utf_32** (p. 85) は "utf-32" という名前を持ち、 CES UTF-32 (RFC 2279) 用のコード系を示す。

2.13.4.9 MSymbol Mcoding_utf_32be

UTF-32BE コード系のシンボル.

シンボル **Mcoding_utf_32be** (p. 86) は "utf-32be" という名前を持ち、 CES UTF-32BE (RFC 2279) 用のコード系を示す。

2.13.4.10 MSymbol Mcoding_utf_32le

UTF-32LE コード系のシンボル.

シンボル **Mcoding_utf_32le** (p. 86) は "utf-32le" という名前を持ち、 CES UTF-32LE (RFC 2279) 用のコード系を示す。

2.13.4.11 MSymbol Mcoding_sjis

SJIS コード系のシンボル.

シンボル **Mcoding_sjis** (p. 86) has は "sjis" という名前を持ち、 CES Shift-JIS 用のコード系を示す。

2.13.4.12 MSymbol Mtype

mconv_define_coding() (p. 76) 用パラメータキー (詳細は **mconv_define_coding()** (p. 76) 参照).

2.13.4.13 MSymbol Mcharsets

2.13.4.14 MSymbol Mflags

2.13.4.15 MSymbol Mdesignation

2.13.4.16 MSymbol Minvocation

2.13.4.17 MSymbol Mcode_unit

2.13.4.18 MSymbol Mbom

2.13.4.19 MSymbol Mlittle_endian

2.13.4.20 MSymbol Mutf

関数 **mconv_define_coding()** (p. 76) の引数として用いられるコード系のパラメータ **Mtype** (p. 86) の値となり得るシンボル。(詳細は **mconv_define_coding()** (p. 76) 参照)。

2.13.4.21 MSymbol Miso_2022

2.13.4.22 MSymbol Mreset_at_eol

2.13.4.23 MSymbol Mreset_at_cntl

2.13.4.24 MSymbol Meight_bit

2.13.4.25 MSymbol Mlong_form

2.13.4.26 MSymbol Mdesignation_g0

2.13.4.27 MSymbol Mdesignation_g1

2.13.4.28 MSymbol Mdesignation_ctext

2.13.4.29 MSymbol Mdesignation_ctext_ext

2.13.4.30 MSymbol Mlocking_shift

2.13.4.31 MSymbol Msingle_shift

2.13.4.32 MSymbol Msingle_shift_7

2.13.4.33 MSymbol Meuc_tw_shift

2.13.4.34 MSymbol Miso_6429

2.13.4.35 MSymbol Mrevision_number

2.13.4.36 MSymbol Mfull_support

2.13.4.37 MSymbol Mmaybe

"maybe"という名前を持つシンボル.

変数 **Mmaybe** (p. 87) は `"maybe"` という名前を持つ。これは関数 **mconv_define_coding()** (p. 76) パラメータ **Mbom** の値として用いられる。 (詳細は **mconv_define_coding()** (p. 76) 参照)。

2.13.4.38 MSymbol Mcoding

シンボル `Mcoding`.

デコードされた M-text はすべて、キーが定義済みシンボル `Mcoding` であるようなテキストプロパティを持つ。シンボル `Mcoding` は `"coding"` という名前を持つ。

2.14 ロケール

ロケールオブジェクトとそれに関する API.

型定義

- typedef struct **MLocale MLocale**
  `MLocale` **構造体**.

関数

- **MLocale** ∗ **mlocale_set** (int category, const char ∗name)
  現在のロケールを設定する.
- **MSymbol mlocale_get_prop** (**MLocale** ∗locale, **MSymbol** key)
  ロケールプロパティの値を得る.
- int **mtext_ftime** (**MText** ∗mt, const char ∗format, const struct tm ∗tm, **MLocale** ∗locale)
  日付と時間をフォーマットする.
- **MText** ∗ **mtext_getenv** (const char ∗name)
  環境変数を得る.
- int **mtext_putenv** (**MText** ∗mt)
  環境変数を変更／追加する.
- int **mtext_coll** (**MText** ∗mt1, **MText** ∗mt2)
  現在のロケールを用いて２つの *M-text* **を比較する**.

変数

- **MSymbol Mterritory**
- **MSymbol Mmodifier**
- **MSymbol Mcodeset**

2.14.1 説明

ロケールオブジェクトとそれに関する API.

m17n ライブラリはロケール関連情報を **MLocale** (p. 88) 型のオブジェクトで表現する。

2.14.2 型定義

2.14.2.1 typedef struct MLocale MLocale

`MLocale` 構造体.

`MLocale` 構造体は、ロケールの名前、言語、地域、モディファイア、コードセット、および対応するコード系に関する情報を保持するために用いられる。

この構造体の内容は実装に依存する。 内部構造はアプリケーションプログラムからは見えない。

参照:
mlocale_get_prop() (p. 89)

2.14.3 関数

2.14.3.1 MLocale* mlocale_set (int *category*, const char * *name*)

現在のロケールを設定する.

関数 **mlocale_set()** (p. 89) は現在のロケールの一部を設定したり問い合わせたりする。ここで一部とは **category** で指定され、`setlocale()` の有効な第一引数となるものでなくてはならない。

locale が NULL でなければ、指定した部分のロケールが **locale** に設定される。 **locale** がシステムにサポートされていなければ、設定は行われず、現在のロケールは変わらない。

locale が NULL ならば、現在のロケールの指定した部分を問い合わせる。

戻り値:

呼び出しに成功すれば、mlocale_set() はロケールに対応する opaque ロケールオブジェクトを返す。ロケールの名前は関数 **mlocale_get_prop()** (p. 89) によって得ることができる。 そうでなければ NULL を返す。

エラー:
`MERROR_LOCALE`

2.14.3.2 MSymbol mlocale_get_prop (MLocale * *locale*, MSymbol *key*)

ロケールプロパティの値を得る.

関数 **mlocale_get_prop()** (p. 89) は、ロケール **locale** の **key** プロパティの値を返す。 **key** は **Mname** (p. 27), **Mlanguage** (p. 47), **Mterritory** (p. 90), **Mcodeset** (p. 90), **Mmodifier** (p. 90), **Mcoding** (p. 87) のいずれかである。

2.14.3.3 int mtext_ftime (MText * *mt*, const char * *format*, const struct tm * *tm*, MLocale * *locale*)

日付と時間をフォーマットする.

関数 **mtext_ftime()** (p. 89) は時刻データ (broken-down time) **tm** を **format** で指定された形式に清書し、結果を M-text **mt** に付加する。フォーマットは NULL でなければ ロケール **locale** に、または現在のロケール (LC_TIME) に従う。

引数 **tm** と **format** の意味は strftime() の場合と同じ。

参照:
strftime().

2.14.3.4 MText* mtext_getenv (const char * *name*)

環境変数を得る.

関数 **mtext_getenv()** (p. 89) は **name** で指される文字列と合致する文字列を環境変数のリスト中から探す。

見つかった場合には、その値を現在のロケール (LC_CTYPE) に従って M-text にデコードし、その M-text を返す。

見つからなければ、NULL を返す。

2.14.3.5 int mtext_putenv (MText * *mt*)

環境変数を変更／追加する.

関数 **mtext_putenv()** (p. 90) は M-text **mt** に従って、環境変数の値を変更したり追加したりする。この関数は、現在のロケール (LC_CTYPE) に従って **mt** をエンコードし、それを引数として関数 `putenv` を呼ぶ。

戻り値:
この関数は、成功した場合には 0 を、エラーが起これば -1 を返す。

2.14.3.6 int mtext_coll (MText * *mt1*, MText * *mt2*)

現在のロケールを用いて２つの M-text を比較する.

関数 **mtext_coll()** (p. 90) は２つの M-text **mt1** と **mt2** を比較する。戻り値は負の整数値, 0, 正の整数値のいずれかであり、それぞれ **mt1** が **mt2** より小さい、同じ、大きい場合に相当する。比較は現在のロケール (LC_COLLATE) に基づいて行われる。

この関数は M-text のテキストプロパティとして自動的にキャッシュされる情報を利用するので、２度目以降の同じ比較は１度目より速く実行される。

2.14.4 変数

2.14.4.1 MSymbol Mterritory

"territory" という名前を持つシンボル.

2.14.4.2 MSymbol Mmodifier

"modifier" という名前を持つシンボル.

2.14.4.3 MSymbol Mcodeset

"codeset" という名前を持つシンボル.

2.15 入力メソッド (基本部分)

入力メソッド用 API.

データ構造

- struct **MInputDriver**
  入力ドライバ用構造体.
- struct **MInputMethod**
  入力メソッドの構造体.
- struct **MInputContext**
  入力コンテクスト用構造体.

変数： コールバックコマンド用定義済みシンボル.

入力メソッドドライバのコールバック関数において COMMAND 引数として用いられる定義済みシンボル (**MInputDriver::callback_list** (p. 187) 参照)。

ほとんどは追加の引数を必要としないし値を返さないが、以下は例外である。

Minput_get_surrounding_text: このコマンドに割り当てられたコールバッ ク関数が呼ばれた際には、**MInputContext::plist** (p. 185) の第一要素はキーとし て **Minteger** (p. 23) をとり、その値はサラウンディングテキストのうちどの部分 を取って来るかを指定する。値が正であれば、現在のカーソル位置に続く 値の個数分の文字を取る。負であれば、カーソル位置に先行する値の絶対 値分の文字を取る。現在サラウンドテキストがサポートされているかどう かを知りたいだけであれば、この値はゼロでも良い。

サラウンディングテキストがサポートされていれば、コールバック関数は この要素のキーを **Mtext** (p. 23) に、値を取り込んだ M-text に設定しなくてはな らない。もしテキストの長さが充分でなければ、この M-text の長さは要 求されている文字数より短くて良い。最悪の場合 0 でもよいし、アプリケー ション側で必要で効率的だと思えば長くても良い。

サラウンディングテキストがサポートされていなければ、コールバック関 数は **MInputContext::plist** (p. 185) の第一要素を変更してはならない。

Minput_delete_surrounding_text: このコマンドに割り当てられたコール バック関数が呼ばれた際には、**MInputContext::plist** (p. 185) の第一要素は、キー として **Minteger** (p. 23) をとり、値は削除するべきサラウンディングテキストを Minput_get_surrounding_text と同様のやり方で指定する。コールバック 関数は指定されたテキストを削除しなければならない。また **MInputContext::plist** (p. 185) を変えてはならない。

- **MSymbol Minput_preedit_start**
- **MSymbol Minput_preedit_done**
- **MSymbol Minput_preedit_draw**
- **MSymbol Minput_status_start**
- **MSymbol Minput_status_done**
- **MSymbol Minput_status_draw**
- **MSymbol Minput_candidates_start**
- **MSymbol Minput_candidates_done**
- **MSymbol Minput_candidates_draw**
- **MSymbol Minput_set_spot**
- **MSymbol Minput_toggle**
- **MSymbol Minput_reset**
- **MSymbol Minput_get_surrounding_text**
- **MSymbol Minput_delete_surrounding_text**

変数: 特別な入力イベント用定義済みシンボル.

minput_filter() (p. 96) の KEY 引数として用いられる定義済みシンボル。

- **MSymbol Minput_focus_out**
- **MSymbol Minput_focus_in**
- **MSymbol Minput_focus_move**

変数: 入力メソッド情報用定義済みシンボル.

- **MSymbol Minherited**
- **MSymbol Mcustomized**
- **MSymbol Mconfigured**

関数

- **MInputMethod** ∗ **minput_open_im** (**MSymbol** language, **MSymbol** name, void ∗arg)
  入力メソッドをオープンする.
- void **minput_close_im** (**MInputMethod** ∗im)
  入力メソッドをクローズする.
- **MInputContext** ∗ **minput_create_ic** (**MInputMethod** ∗im, void ∗arg)
  入力コンテクストを生成する.
- void **minput_destroy_ic** (**MInputContext** ∗ic)
  入力コンテクストを破壊する.
- int **minput_filter** (**MInputContext** ∗ic, **MSymbol** key, void ∗arg)
  入力キーをフィルタする.
- int **minput_lookup** (**MInputContext** ∗ic, **MSymbol** key, void ∗arg, **MText** ∗mt)
  入力コンテクスト中のテキストを探す.
- void **minput_set_spot** (**MInputContext** ∗ic, int x, int y, int ascent, int descent, int fontsize, **MText** ∗mt, int pos)
  入力コンテクストのスポットを設定する.
- void **minput_toggle** (**MInputContext** ∗ic)
  入力メソッドを切替える.
- void **minput_reset_ic** (**MInputContext** ∗ic)
  入力コンテクストをリセットする.
- **MPlist** ∗ **minput_get_title_icon** (**MSymbol** language, **MSymbol** name)
  入力メソッドのタイトルとアイコン用ファイル名を得る.
- **MText** ∗ **minput_get_description** (**MSymbol** language, **MSymbol** name)
  入力メソッドの説明テキストを得る.
- **MPlist** ∗ **minput_get_command** (**MSymbol** language, **MSymbol** name, **MSymbol** command)

入力メソッドのコマンドに関する情報を得る.

- int **minput_config_command** (**MSymbol** language, **MSymbol** name, **MSymbol** command, **MPlist** *keyseqlist)

  入力メソッドのコマンドのキーシークエンスを設定する.

- **MPlist** * **minput_get_variable** (**MSymbol** language, **MSymbol** name, **MSymbol** variable)

  入力メソッドの変数に関する情報を得る.

- int **minput_config_variable** (**MSymbol** language, **MSymbol** name, **MSymbol** variable, **MPlist** *value)

  入力メソッドの変数の値を設定する.

- char * **minput_config_file** ()

  ユーザ毎のカスタマイズファイルの名前を得る.

- int **minput_save_config** (void)

  設定をユーザ毎のカスタマイズファイルに保存する.

Obsolete な関数

- **MPlist** * **minput_get_variables** (**MSymbol** language, **MSymbol** name)

  入力メソッドの変数リストを得る.

- int **minput_set_variable** (**MSymbol** language, **MSymbol** name, **MSymbol** variable, void *value)

  入力メソッド変数の初期値を設定する.

- **MPlist** * **minput_get_commands** (**MSymbol** language, **MSymbol** name)

  入力メソッドのコマンドに関する情報を得る.

- int **minput_assign_command_keys** (**MSymbol** language, **MSymbol** name, **MSymbol** command, **MPlist** *keyseq)

  入力メソッドコマンドにキーシークエンスを割り当てる.

- int **minput_callback** (**MInputContext** *ic, **MSymbol** command)

型定義

- typedef struct **MInputMethod** **MInputMethod**

  *See struct **MInputMethod*** (p. *189*)

- typedef struct **MInputContext** **MInputContext**

  *See struct **MInputContext*** (p. *182*)

- typedef void(* **MInputCallbackFunc**)(**MInputContext** *ic, **MSymbol** command)

  入力メソッドコールバック関数の型宣言.

列挙型

- enum **MInputCandidatesChanged** {
  MINPUT_CANDIDATES_LIST_CHANGED = 1,
  MINPUT_CANDIDATES_INDEX_CHANGED = 2,
  MINPUT_CANDIDATES_SHOW_CHANGED = 4,
  MINPUT_CANDIDATES_CHANGED_MAX }
  入力メソッドの入力候補がどう変更されたかを示すビットマスク.

変数

- **MSymbol Minput_method**
  "input-method" **を名前として持つシンボル**.
- **MInputDriver minput_default_driver**
  内部入力メソッド用デフォルトドライバ.
- **MInputDriver** ∗ **minput_driver**
  内部入力メソッド用ドライバ.
- **MSymbol Minput_driver**

2.15.1 説明

入力メソッド用 API.

入力メソッドは多様な文字を入力するためのオブジェクトである。 入力メソッドはシンボル LANGUAGE と NAME の組によって識別され、 この組合せによって入力メソッドドライバが決定する。 入力メソッドドライバとは、ある入力メソッドを扱うための関数の集まりである。 入力メソッドには内部メソッドと外部メソッドの二種類がある。

- 内部入力メソッド
  内部入力メソッドとは LANGUAGE が `Mnil` 以外のものであり、その本体 は m17n データベースに <Minput_method, LANGUAGE, NAME> というタグを付 けて定義されている。この種の入力メソッドに対して、m17n ライブラリで は CUI 用と GUI 用それぞれの入力メソッドドライバをあらかじめ定義して いる。これらのドライバは m17n ライブラリ自体の入力処理エンジンを利 用する。m17n データベースには、特定の言語専用でない入力メソッドを定 義することもでき、そのような入力メソッドの LANGUAGE は `Mt` である。
  内部入力メソッドは、ユーザの入力イベントに対応したシンボルである入 力キーを受け取る。`m17n` ライブラリ は入力イベントがアプリケー ションプログラムでどう表現されているかを知ることができないので、入 力イベントから入力キーへの変換はアプリケーションプログラマの責任で 行わなくてはならない。詳細については関数 **minput_event_to_key()** (p. 147) の 説明を参照。
- 外部入力メソッド
  外部入力メソッドとは LANGUAGE が `Mnil` のものであり、その本体は外 部のリソースとして定義される。(たとえば X Window System の XIM な ど。) この種の入力メソッドでは、シンボル NAME は **Minput_driver** (p. 106) を キーとするプロパティを持ち、その値は入力メソッドドライバへのポイン タである。このことにより、適切なドライバを準備することによって、い かなる種類の入力メソッドも `m17n` ライブラリ の枠組の中で扱う事 ができる。
  利便性の観点から、m17n X ライブラリは XIM の OverTheSpot の入力スタ イルを実現する入力メソッドドライバを提供し、またシンボル `Mxim` の **Minput_driver** (p. 106) プロパティの値としてその

ドライバへのポインタを保持 している。詳細については m17n GUI API のドキュメントを参照のこと。

処理の流れ

入力メソッド処理の典型的な処理は以下のようになる。

- 入力メソッドのオープン
- その入力メソッドの入力コンテクストの生成
- 入力イベントのフィルタ
- 入力コンテクストでの生成テキストの検索

2.15.2 型定義

2.15.2.1 typedef struct MInputMethod MInputMethod

See struct **MInputMethod** (p. 189)

2.15.2.2 typedef struct MInputContext MInputContext

See struct **MInputContext** (p. 182)

2.15.2.3 typedef void(∗ MInputCallbackFunc)(MInputContext ∗ic, MSymbol command)

入力メソッドコールバック関数の型宣言.

入力メソッドから呼ばれるコールバック関数の型である。**ic** は入力コンテクストへのポインタ、**command** は関数が呼ばれるコールバックの名前である。

2.15.3 列挙型

2.15.3.1 enum MInputCandidatesChanged

入力メソッドの入力候補がどう変更されたかを示すビットマスク.

列挙型の値:

MINPUT_CANDIDATES_LIST_CHANGED

MINPUT_CANDIDATES_INDEX_CHANGED

MINPUT_CANDIDATES_SHOW_CHANGED

MINPUT_CANDIDATES_CHANGED_MAX

2.15.4 関数

2.15.4.1 MInputMethod∗ minput_open_im (MSymbol *language*, MSymbol *name*, void ∗ *arg*)

入力メソッドをオープンする.

関数 **minput_open_im()** (p. 95) は言語 **language** と名前 **name** に合致する入力メソッドをオープンし、新たに割り当てられた入力メソッドオブジェクトへのポインタを返す。

この関数は、まず入力メソッド用のドライバを以下のようにして決定する。

language が **Mnil** (p. 16) でなければ、変数 **minput_driver** (p. 106) で指されているドライバを用いる。

language が **Mnil** (p. 16) であり、**name** が **Minput_driver** (p. 106) プロパティを持つ場合には、そのプロパティの値で指されている入力ドライバを用いて入力メソッドをオープンする。 **name** にそのようなプロパティが無かった場合は NULL を返す。

次いで、ドライバのメンバ **MInputDriver::open_im()** (p. 186) が呼ばれる。

arg は構造体 **MInputMethod** (p. 189) のメンバ arg に設定され、ドライバから参照できる。

2.15.4.2 void minput_close_im (MInputMethod * *im*)

入力メソッドをクローズする.

関数 **minput_close_im()** (p. 96) は、入力メソッド **im** をクローズする。 この入力メソッド **im** は **minput_open_im()** (p. 95) によって作られたものでなければならない。

2.15.4.3 MInputContext* minput_create_ic (MInputMethod * *im*, void * *arg*)

入力コンテクストを生成する.

関数 **minput_create_ic()** (p. 96) は入力メソッド **im** に対応する入力コンテクストオブジェクトを生成し、**Minput_preedit_start**, **Minput_status_start**, **Minput_status_draw** に対応するコールバック関数をこの順に呼ぶ。

戻り値:
入力コンテクストが生成された場合、minput_create_ic() はその入力コンテクストへのポインタを返す。失敗した場合は NULL を返す。

2.15.4.4 void minput_destroy_ic (MInputContext * *ic*)

入力コンテクストを破壊する.

関数 **minput_destroy_ic()** (p. 96) は、入力コンテクスト **ic** を破壊する。 この入力コンテクストは **minput_create_ic()** (p. 96) によって作られたものでなければならない。この関数は **Minput_preedit_done**, **Minput_status_done**, **Minput_candidates_done** に対応するコールバック関数をこの順に呼ぶ。

2.15.4.5 int minput_filter (MInputContext * *ic*, MSymbol *key*, void * *arg*)

入力キーをフィルタする.

関数 **minput_filter()** (p. 96) は入力キー **key** を入力コンテクスト **ic** に応じてフィルタし、preedit テキスト、ステータス、現時点での候補が変化した時点で、それぞれ **Minput_preedit_draw**, **Minput_status_draw**, **Minput_candidates_draw** に対応するコールバック関数を呼ぶ。

戻り値:
key がフィルタされれば、この関数は 1 を返す。 この場合呼び出し側はこのキーを捨てるべきである。そうでなければ 0 を返し、呼び出し側は、たとえば同じキーで関数 **minput_lookup()** (p. 96) を呼ぶなどして、このキーを処理する。

2.15.4.6 int minput_lookup (MInputContext * *ic*, MSymbol *key*, void * *arg*, MText * *mt*)

入力コンテクスト中のテキストを探す.

関数 **minput_lookup()** (p. 96) は入力コンテクスト **ic** 中のテキストを探す。 **key** は関数 **minput_filter()** (p. 96) への直前の呼び出しに用いられたものと同じでなくてはならない。

テキストが入力メソッドによって生成されていれば、テキストは M-text **mt** に連結される。

この関数は、**MInputDriver::lookup** (p. 187) を呼ぶ。

戻り値:

key が入力メソッドによって適切に処理できれば、この関数は 0 を返す。 そうでなければ -1 を返す。この場合でも **mt** に何らかのテキストが生成されていることがある。

2.15.4.7 void minput_set_spot (MInputContext * *ic*, int *x*, int *y*, int *ascent*, int *descent*, int *fontsize*, MText * *mt*, int *pos*)

入力コンテクストのスポットを設定する.

関数 **minput_set_spot()** (p. 97) は、入力コンテクスト **ic** のスポットを、座標 (**x**, **y**) の位置に 、高さ **ascent**、 **descent** で設定する。 これらの値の意味は入力メソッドドライバに依存する。

たとえば CUI 環境で動作するドライバは **x** と **y** をそれぞれ列と行の番号として用い、**ascent** と **descent** を無視するかもしれない。 またウィンドウシステム用のドライバは **x** と **y** をクライアントウィンドウの原点からのオフセットをピクセル単位で表したものとして扱い、 **ascent** と **descent** を (**x** . **y**) の列のアセントとディセントをピクセル単位で表したものとして扱うかもしれない。

fontsize には preedit テキストのフォントサイズを 1/10 ポイント単位で指定する。

mt と **pos** はそのスポットの M-text と文字位置である。**mt** は `NULL` でもよく、その場合には入力メソッドはスポット周辺のテキストに関する情報を得ることができない。

2.15.4.8 void minput_toggle (MInputContext * *ic*)

入力メソッドを切替える.

関数 **minput_toggle()** (p. 97) は入力コンテクスト **ic** に対応付けられた入力メソッドをトグルする。

2.15.4.9 void minput_reset_ic (MInputContext * *ic*)

入力コンテクストをリセットする.

関数 **minput_reset_ic()** (p. 97) は **Minput_reset** に対応するコールバック関数 を呼ぶことによって入力コンテクスト **ic** をリセットする。リセットとは、 実際には入力メソッドを初期状態に移すことである。現在入力中のテキス トはコミットされることなく削除されるので、アプリケーションプログラ ムは、必要ならば予め **minput_filter()** (p. 96) を引数 **key Mnil** (p. 16) で呼んで 強制的にプリエディットテキストをコミットさせること。

2.15.4.10 MPlist* minput_get_title_icon (MSymbol *language*, MSymbol *name*)

入力メソッドのタイトルとアイコン用ファイル名を得る.

関数 **minput_get_title_icon()** (p. 97) は、 **language** と **name** で指定される 入力メソッドのタイトルと（あれば）アイコン用ファイルを含む plist を 返す。

plist の第一要素は、**Mtext** (p. 23) をキーに持ち、値は入力メソッドを識別する タイトルを表す M-text である。第二要素があれば、キーは **Mtext** (p. 23) であ り、値は識別用アイコン画像ファイルの絶対パスを表す M-text である。

戻り値:
指定の入力メソッドが存在し、タイトルが定義されていれば plist を返す。そうでなければ NULL を返す。呼出側は 関数 **m17n_object_unref()** (p. 12) を用いて plist を解放しなくてはならない。

2.15.4.11 MText* minput_get_description (MSymbol *language*, MSymbol *name*)

入力メソッドの説明テキストを得る.

関数 **minput_get_description()** (p. 98) は、**language** と **name** によって指定 された入力メソッドを説明する M-text を返す。

戻り値:
指定された入力メソッドが説明するテキストを持っていれば、 **MText** (p. 36) へのポインタを返す。呼び出し側は、それを m17n_object_unref () を用いて解放しなくてはならない。入力メソッドに説明テキストが無け れば NULL を返す。

2.15.4.12 MPlist* minput_get_command (MSymbol *language*, MSymbol *name*, MSymbol *command*)

入力メソッドのコマンドに関する情報を得る.

関数 **minput_get_command()** (p. 98) は、**language** と **name** で指定される入力 メソッドのコマンド **command** に関する情報を返す。入力メソッドのコマ ンドとは、疑似キーイベントであり、１つ以上の実際の入力キーシークエ ンスが割り当てられる。

コマンドには、グローバルとローカルの２種類がある。グローバルなコマンド はグローバルに定義され、ローカルなコマンドはその説明とキー割り当て を継承することができる。各入力メソッドはローカルなキー割当を持つロー カルなコマンドを定義する。また同名のグローバルなコマンドの定義を継 承するローカルなコマンドを宣言することもできる。

language が **Mt** (p. 17) で **name** が **Mnil** (p. 16) の場合は、この関数はグローバルコ マンドに関する情報を返す。そうでなければローカルコマンドに関するも のを返す。

command が **Mnil** (p. 16) の場合は、すべてのコマンドに関する情報を返す。

戻り値は以下の形式の *well-formed* plist (**プロパティリスト** (p. 18)) である。

```
((NAME DESCRIPTION STATUS [KEYSEQ ...]) ...)
```

NAME はコマンド名を示すシンボルである。

DESCRIPTION はコマンドを説明する M-text であるか、説明が無い場合に は **Mnil** (p. 16) である。

STATUS はキー割り当てがどのように定められるかをあらわすシンボル であり、その値は **Mnil** (p. 16)（デフォルトの割り当て）, **Mcustomized**（ユー ザ毎のカスタマイズファイルによってカスタマイズされた割り当て）, **Mconfigured**（minput_config_command() を呼ぶことによって設定される 割り当て）のいずれかである。ローカルコマンドの場合には、 **Minherited**（対応するグローバルコマンドからの継承による割り当て） でもよい。

KEYSEQ は１つ以上のシンボルからなる plist であり、各シンボルはコマ ンドに割り当てられているキーシークエンスを表す。KEYSEQ が無い場合は、 そのコマンドは現状で使用不能である。（すなわちコマンドの動作を起 動できるキーシークエンスが無い。）

command が **Mnil** (p. 16) でなければ、返される plist の最初の要素は、 **command** に関する情報を含む。

戻り値:

求められた情報が見つかれば、空でない plist へのポインタを返す。リス トはライブラリが管理しているので、呼出側が変更したり解放したりする ことはできない。

そうでなければ、すなわち指定の入力メソッドやコマンドが存在しなければ NULL を返す。

例：

```
MText *
get_im_command_description (MSymbol language, MSymbol name, MSymbol command)
{
  /* Return a description of the command COMMAND of the input method
     specified by LANGUAGE and NAME.  */
  MPlist *cmd = minput_get_command (langauge, name, command);
  MPlist *plist;

  if (! cmds)
    return NULL;
  plist = mplist_value (cmds);  /* (NAME DESCRIPTION STATUS KEY-SEQ ...) */
  plist = mplist_next (plist);  /* (DESCRIPTION STATUS KEY-SEQ ...) */
  return  (mplist_key (plist) == Mtext
           ? (MText *) mplist_value (plist)
           : NULL);
}
```

2.15.4.13 int minput_config_command (MSymbol *language*, MSymbol *name*, MSymbol *command*, MPlist * *keyseqlist*)

入力メソッドのコマンドのキーシークエンスを設定する.

関数 **minput_config_command()** (p. 99) はキーシークエンスのリスト **keyseqlist** を、**language** と **name** によって指定される入力メソッドの コマンド **command** に割り当てる。

keyseqlist が空リストでなければ、キーシークエンスのリストであり、 各キーシークエンスはシンボルの plist である。

keyseqlist が空の plist ならば、そのコマンドの設定やカスタマイズは すべてキャンセルされ、デフォルトのキーシークエンスが有効になる。

keyseqlist が NULL であれば、そのコマンドの設定はキャンセルされ、 元のキーシークエンス（ユーザ毎のカスタマイズファイルに保存されてい るもの、あるいはデフォルトのもの）が有効になる。

後のふたつの場合には、**command** は **Mnil** (p. 16) をとることができ、指定の入 力メソッドの全てのコマンド設定のキャンセルを意味する。

name が **Mnil** (p. 16) ならば、この関数は個々の入力メソッドではなくグローバ ルなコマンドのキー割り当てを設定する。

これらの設定は、現行のセッション中で入力メソッドがオープン（または 再オープン）された時点で有効になる。将来のセッション中でも有効にす るためには、関数 **minput_save_config()** (p. 102) を用いてユーザ毎のカスタマイ ズファイルに保存しなくてはならない。

戻り値:

この関数は、処理が成功すれば 0 を、失敗すれば -1 を返す。失敗とは以下の場合である。

- **keyseqlist** が有効な形式でない。
- **command** が指定の入力メソッドで利用できない。
- **language** と **name** で指定される入力メソッドが存在しない。

参照:

minput_get_commands() (p. 103), **minput_save_config()** (p. 102).

例：

```
/* Add "C-x u" to the "start" command of Unicode input method.  */
{
  MSymbol start_command = msymbol ("start");
  MSymbol unicode = msymbol ("unicode");
  MPlist *cmd, *plist, *key_seq_list, *key_seq;

  /* At first get the current key-sequence assignment.  */
  cmd = minput_get_command (Mt, unicode, start_command);
  if (! cmd)
    {
      /* The input method does not have the command "start".  Here
         should come some error handling code.  */
    }
  /* Now CMD == ((start DESCRIPTION STATUS KEY-SEQUENCE ...) ...).
     Extract the part (KEY-SEQUENCE ...).  */
  plist = mplist_next (mplist_next (mplist_next (mplist_value (cmd))));
  /* Copy it because we should not modify it directly.  */
  key_seq_list = mplist_copy (plist);

  key_seq = mplist();
  mplist_add (key_seq, Msymbol, msymbol ("C-x"));
  mplist_add (key_seq, Msymbol, msymbol ("u"));
  mplist_add (key_seq_list, Mplist, key_seq);
  m17n_object_unref (key_seq);

  minput_config_command (Mt, unicode, start_command, key_seq_list);
  m17n_object_unref (key_seq_list);
}
```

2.15.4.14 MPlist* minput_get_variable (MSymbol *language*, MSymbol *name*, MSymbol *variable*)

入力メソッドの変数に関する情報を得る.

関数 **minput_get_variable()** (p. 100) は、**language** と **name** で指定される入力 メソッドの変数 **variable** に関する情報を返す。入力メソッドの変数とは、 入力メソッドの振舞を制御するものである。

変数には、グローバルとローカルの 2 種類がある。グローバルな変数はグ ローバルに定義され、ローカルな変数はその説明と値を継承することがで きる。各入力メソッドはローカルな値を持つローカルな変数を定義する。 また同名のグローバルな変数の定義を継承するローカルな変数を宣言する こともできる。

language が **Mt** (p. 17) で **name** が **Mnil** (p. 16) の場合は、この関数はグローバル変 数に関する情報を返す。そうでなければローカル変数に関するものを返す。

variable が **Mnil** (p. 16) の場合は、すべてのコマンドに関する情報を返す。

戻り値は以下の形式の *well-formed* plist (**プロパティリスト** (p. 18)) である。

```
((NAME DESCRIPTION STATUS VALUE [VALID-VALUE ...]) ...)
```

NAME は変数の名前を示すシンボルである。

DESCRIPTION は変数を説明する M-text であるか、説明が無い場合には **Mnil** (p. 16) である。

STATUS は値がどのように定められるかをあらわすシンボルであり、 STATUS の値は **Mnil** (p. 16) （デフォルトの値）, **Mcustomized** （ユーザ毎の カスタマイズファイルによってカスタマイズされた値）, **Mconfigured** （minput_config_variable() を呼ぶことによって設定される値）のいずれ かである。ローカル変数の場合には、**Minherited** （対応するグローバル 変数から継承した値）でもよい。

VALUE は変数の初期値である。この要素のキーが **Mt** (p. 17) であれば初期値を持 たない。そうでなければ、キーは **Minteger** (p. 23), **Msymbol** (p. 17), **Mtext** (p. 23) のいずれ かであり、値はそれぞれ対応する型のものである。

VALID-VALUE はもしあれば、変数の取り得る値を指定する。これは VALUE と同じ型 (すなわち同じキーを持つ) であるが、例外として VALUE が integer の場合は VALID-VALUE は可能な値の範囲を示す二つの

整数から なる plist となることができる。

VALID-VALUE がなければ、変数は VALUE と同じ型である限りいかなる値も とることができる。

variable が **Mnil** (p. 16) でなければ、返される plist の最初の要素は **variable** に関する情報を含む。

戻り値:

求められた情報が見つかれば、空でない plist へのポインタを返す。リス トはライブラリが管理しているので、呼出側が変更したり解放したりする ことはできない。

そうでなければ、すなわち指定の入力メソッドや変数が存在しなければ NULL を返す。

2.15.4.15 int minput_config_variable (MSymbol *language*, MSymbol *name*, MSymbol *variable*, MPlist * *value*)

入力メソッドの変数の値を設定する.

関数 **minput_config_variable()** (p. 101) は値 **value** を、**language** と **name** によって指定される入力メソッドの変数 **variable** に割り当てる。

value が 空リストでなければ、 1 要素の plist であり、そのキーは **Minteger** (p. 23), **Msymbol** (p. 17), **Mtext** (p. 23) のいずれか、値は対応する型のものである。 この値が変数 **variable** に割り当てられる。

value が 空リストであれば、変数の設定とカスタマイズがキャンセルさ れ、デフォルト値が変数 **variable** に割り当てられる。

value が NULL であれば、変数の設定はキャンセルされ、元の値（ユーザ 毎のカスタマイズファイル中の値、またはデフォルトの値）が割り当てられる。

後のふたつの場合には、**variable** は **Mnil** (p. 16) をとることができ、指定され た入力メソッドの全ての変数設定のキャンセルを意味する。

name が **Mnil** (p. 16) ならば、この関数は個々の入力メソッドではなくグローバ ルな変数の値を設定する。

これらの設定は、現行のセッション中で入力メソッドがオープン（または 再オープン）された時点で有効になる。将来のセッション中でも有効にす るためには、関数 **minput_save_config()** (p. 102) を用いてユーザ毎のカスタマイ ズファイルに保存しなくてはならない。

戻り値:

この関数は、処理が成功すれば 0 を、失敗すれば -1 を返す。失敗とは以下の場合である。

- **value が有効な形式でない。型が定義に合わない、または値が範囲外である。**
- **variable** が指定の入力メソッドで利用できない。
- **language** と **name** で指定される入力メソッドが存在しない。

参照:
minput_get_commands() (p. 103), **minput_save_config()** (p. 102).

2.15.4.16 char* minput_config_file (void)

ユーザ毎のカスタマイズファイルの名前を得る.

関数 **minput_config_file()** (p. 101) は、関数 **minput_save_config()** (p. 102) が設定を 保存するユーザ毎のカスタマイズファイルへの絶対パス名を返す。通常は、ユーザ のホームディレクトリの下のディレクトリ

".m17n.d" にある"config.mic" となる。返された名前のファイルが存在するか、読み書きで きるかは保証されない。関数 minput_save_config() が失敗して -1 を返 した場合には、アプリケーションプログラムはファイルの存在を確認し、（できれば）書き込み可能にし再度 minput_save_config() を試すことが できる。

戻り値:

この関数は文字列を返す。文字列はライブラリが管理しているので、呼出 側が修正したり解放したりすることはできない。

参照:

minput_save_config() (p. 102)

2.15.4.17 int minput_save_config (void)

設定をユーザ毎のカスタマイズファイルに保存する.

関数 **minput_save_config()** (p. 102) は現行のセッションでこれまでに行った設定 をユーザ毎のカスタマイズファイルに保存する。

戻り値:

成功すれば 1 を返す。ユーザ毎のカスタマイズファイルがロックされてい れば 0 を返す。この場合、呼出側はしばらく待って再試行できる。設定ファ イルが書き込み不可の場合、-1 を返す。この場合、minput_config_file () を呼んでファイル名をチェックし、できれば書き込み可能にし、再試行 できる。

参照:

minput_config_file() (p. 101)

2.15.4.18 MPlist* minput_get_variables (MSymbol *language*, MSymbol *name*)

入力メソッドの変数リストを得る.

関数 **minput_get_variables()** (p. 102) は、**language** と **name** によって指定さ れた入力メソッドの振る舞いを制御する変数のプロパティリスト (**MPlist** (p. 19)) を返す。このリストは *well-formed* であり (**プロパティリスト** (p. 18)) 以 下の形式である。

```
(VARNAME (DOC-MTEXT DEFAULT-VALUE [ VALUE ... ] )
 VARNAME (DOC-MTEXT DEFAULT-VALUE [ VALUE ... ] )
 ...)
```

VARNAME は変数の名前を示すシンボルである。

DOC-MTEXT は変数を説明する M-text である。

DEFAULT-VALUE は変数のデフォルト値であり、シンボル、整数もしくは M-text である。

VALUE は、もし指定されていれば変数の取り得る値を示す。もし DEFAULT-VALUE が整数なら、 VALUE は (FROM TO) という形 のリストでも良い。この場合 FROM と TO は可能な値の範囲を示す。

例として、ある入力メソッドが次のような変数を持つ場合を考えよう。

- name:intvar, 説明:"value is an integer", 初期値:0, 値の範囲:0..3,10,20

- name:symvar, 説明:"value is a symbol", 初期値:nil, 値の範囲:a, b, c, nil
- name:txtvar, 説明:"value is an M-text", 初期値:empty text, 値の範囲なし (どんな M-text でも可)

この場合、返されるリストは以下のようになる。

```
(intvar ("value is an integer" 0 (0 3) 10 20)
 symvar ("value is a symbol" nil a b c nil)
 txtvar ("value is an M-text" ""))
```

戻り値:
入力メソッドが何らかの変数を使用していれば **MPlist** (p. 19) へのポインタを返す。 返されるプロパティリストはライブラリによって管理されており、呼び出し側で変更したり解放したりしてはならない。 入力メソッドが変数を一切使用してなければ、NULL を返す。

2.15.4.19 int minput_set_variable (MSymbol *language*, MSymbol *name*, MSymbol *variable*, void * *value*)

入力メソッド変数の初期値を設定する.

関数 **minput_set_variable()** (p. 103) は、**language** と **name** によって指定された入力メソッドの入力メソッド変数 **variable** の初期値を、 **value** に設定する。

デフォルトの初期値は 0 である。

この設定は、新しくオープンされた入力メソッドから有効となる。

戻り値:
処理が成功すれば 0 を返す。そうでなければ -1 を返し、 **merror_code** (p. 152) を MERROR_IM に設定する。

2.15.4.20 MPlist* minput_get_commands (MSymbol *language*, MSymbol *name*)

入力メソッドのコマンドに関する情報を得る.

関数 **minput_get_commands()** (p. 103) は、 **language** と **name** によって指定さ れた入力メソッドの入力メソッドコマンドに関する情報を返す。入力メソッ ドコマンドとは、疑似キーイベントであり、それぞれに１つ以上の実際の 入力キーシークエンスが割り当てられているものを指す。

コマンドにはグローバルとローカルの２種類がある。グローバルコマンド は複数の入力メソッドにおいて、同じ目的で、グローバルなキー割り当て で用いられる。ローカルコマンドは特定の入力メソッドでのみ、ローカル なキー割当で使用される。

個々の入力メソッドはグローバルコマンドのキー割当を変更することもで きる。グローバルコマンド用のグローバルキー割り当ては、使用する入力 メソッドにおいてそのコマンド用のローカルなキー割当が存在しない場合 にのみ有効である。

name が **Mnil** (p. 16) であれば、グローバルコマンドに関する情報を返す。この 場合、**language** は無視される。

name が **Mnil** (p. 16) でなければ、**language** と **name** によって指定される入 力メソッドに置けるローカルなキー割り当てを持つコマンドに関する情報 を返す。

戻り値:
入力メソッドコマンドが見つからなければ、この関数は NULL を返す。

そうでなければプロパティリストへのポインタを返す。リストの各要素の キーは個々のコマンドを示すシンボルであり、値は下記の COMMAND-INFO の形式のプロパティリストである。

COMMAND-INFO の第一要素のキーは **Mtext** (p. 23) または **Msymbol** (p. 17) である。キー が **Mtext** (p. 23) なら、値はそのコマンドを説明する M-text である。キーが **Msymbol** (p. 17) なら値は **Mnil** (p. 16) であり、このコマンドは説明テキストを持たな いことになる。

それ以外の要素が無ければ、このコマンドに対してキーシークエンスが割 り当てられていないことを意味する。そうでなければ、残りの各要素はキ ーとして **Mplist** (p. 23) を、値としてプロパティリストを持つ。このプロパティ リストのキーは **Msymbol** (p. 17) であり、値は現在そのコマンドに割り当てられ ている入力キーを表すシンボルである。

返されるプロパティリストはライブラリによって管理されており、呼び出 し側で変更したり解放したりしてはならない。

2.15.4.21 int minput_assign_command_keys (MSymbol *language*, MSymbol *name*, MSymbol *command*, MPlist * *keyseq*)

入力メソッドコマンドにキーシークエンスを割り当てる.

関数 **minput_assign_command_keys()** (p. 104) は、 **language** と **name** によって 指定された入力メソッド用の入力メソッドコマンド **command** に対して、 入力キーシークエンス **keyseq** を割り当てる。 **name** が **Mnil** (p. 16) ならば、 **language** に関係なく、入力キーシークエンスはグローバルに割り当てら れる。そうでなれば、割り当てはローカルである。

keyseq の各要素はキーとして **msymbol** を、値として入力キーを表すシ ンボルを持たなくてはならない。

keyseq は `NULL` でもよい。この場合、グローバルもしくはローカルな すべての割り当てが消去される。

この割り当ては、割り当て以降新しくオープンされた入力メソッドから有 効になる。

戻り値:

処理が成功すれば 0 を返す。そうでなければ -1 を返し、 **merror_code** (p. 152) を `MERROR_IM` に設定する。

2.15.4.22 int minput_callback (MInputContext * *ic*, MSymbol *command*)

2.15.5 変数

2.15.5.1 MSymbol Minput_method

"input-method" を名前として持つシンボル.

2.15.5.2 MSymbol Minput_preedit_start

2.15.5.3 MSymbol Minput_preedit_done

2.15.5.4 MSymbol Minput_preedit_draw

2.15.5.5 MSymbol Minput_status_start

2.15.5.6 MSymbol Minput_status_done

2.15.5.7 MSymbol Minput_status_draw

2.15.5.8 MSymbol Minput_candidates_start

2.15.5.9 MSymbol Minput_candidates_done

2.15.5.10 MSymbol Minput_candidates_draw

2.15.5.11 MSymbol Minput_set_spot

2.15.5.12 MSymbol Minput_toggle

2.15.5.13 MSymbol Minput_reset

2.15.5.14 MSymbol Minput_get_surrounding_text

2.15.5.15 MSymbol Minput_delete_surrounding_text

2.15.5.16 MSymbol Minput_focus_out

2.15.5.17 MSymbol Minput_focus_in

2.15.5.18 MSymbol Minput_focus_move

2.15.5.19 MSymbol Minherited

入力メソッドのコマンドや変数の状態を表し、minput_get_command() と **minput_get_variable()** (p. 100) の戻り値として用いられる定義済みシンボル。

2.15.5.20 MSymbol Mcustomized

2.15.5.21 MSymbol Mconfigured

2.15.5.22 MInputDriver minput_default_driver

内部入力メソッド用デフォルトドライバ.

変数 **minput_default_driver** (p. 105) は内部入力メソッド用のデフォルトのドライバを表す。

メンバ **MInputDriver::open_im()** (p. 186) は m17n データベース中からタグ < **Minput_method** (p. 104), **language**, **name**> に合致する入力メソッドを探し、それをロードする。

メンバ **MInputDriver::callback_list()** (p. 187) は NULL であり、 したがって、プログラマ側で責任を持って 適切なコールバック関数の plist に設定しなくてはならない。さもないと、preedit テキストなどのフィードバック情報がユーザに表示されない。

マクロ **M17N_INIT()** (p. 7) は変数 **minput_driver** (p. 106) をこのドライバへのポインタに設定し、全ての内部入力メソッドがこのドライバを使うようにする。

したがって、`minput_driver` がデフォルト値のままであれば、minput_ で始まる関数のドライバに依存する引数 **arg** はすべて無視される。

2.15.5.23 MInputDriver* minput_driver

内部入力メソッド用ドライバ.

変数 **minput_driver** (p. 106) は内部入力メソッドによって使用されている入力メ ソッドドライバへのポインタである。マクロ **M17N_INIT()** (p. 7) はこのポイン タを **minput_default_driver** (p. 105) (<m17n.h> が include されている 時) に初期化する。

2.15.5.24 MSymbol Minput_driver

The variable **Minput_driver** (p. 106) is a symbol for a foreign input method. See **foreign input method** (p. 94) for the detail.

2.16 FLT API

libm17n-flt.so が提供する API

データ構造

- struct **MFLTGlyph**
  グリフに関する情報の型.
- struct **MFLTGlyphAdjustment**
  グリフ位置調整情報のための型.
- struct **MFLTGlyphString**
  グリフ列の情報のための型.
- struct **MFLTOtfSpec**
  GSUB **および** *GPOS OpenType* **テーブルの仕様のための型**.
- struct **MFLTFont**
  FLT **ドライバが使うフォントの型**.

型定義

- typedef struct _MFLT **MFLT**
  FLT (Font Layout Table) **の型**.

関数

- **MFLT** ∗ **mflt_get** (**MSymbol** name)
  指定された名前を持つ *FLT* **オブジェクトを返す**.
- **MFLT** ∗ **mflt_find** (int c, **MFLTFont** ∗font)
  指定された文字とフォントに合った *FLT* **を探す**.
- const char ∗ **mflt_name** (**MFLT** ∗flt)
  FLT **の名前を返す**.
- **MCharTable** ∗ **mflt_coverage** (**MFLT** ∗flt)
  FLT **の範囲を返す**.
- int **mflt_run** (**MFLTGlyphString** ∗gstring, int from, int to, **MFLTFont** ∗font, **MFLT** ∗flt)
  FLT **を使って文字をレイアウトする**.
- **MFLT** ∗ **mdebug_dump_flt** (**MFLT** ∗flt, int indent)

変数

- **MSymbol**(∗ **mflt_font_id**)(struct _MFLTFont ∗font)
- int(∗ **mflt_iterate_otf_feature**)(struct _MFLTFont ∗font, **MFLTOtfSpec** ∗spec, int from, int to, unsigned char ∗table)
- int(∗ **mflt_iterate_otf_feature**)(struct _MFLTFont ∗font, **MFLTOtfSpec** ∗spec, int from, int to, unsigned char ∗table)
- **MSymbol**(∗ **mflt_font_id**)(struct _MFLTFont ∗font)

2.16.1 説明

libm17n-flt.so が提供する API

ウィンドウシステムのための FLT サポート.

このセクションでは、FLT (Font Layout Table) を用いた文字レイアウト機能に関する m17n FLT API を定義する。 FLT の形式は **フォントレイアウトテーブル** (p. 207) に記述されている。

2.16.2 型定義

2.16.2.1 typedef struct _MFLT MFLT

FLT (Font Layout Table) の型.

型 **MFLT** (p. 108) は FLT オブジェクトのための型である。 この内部構造は、アプリケーションプログラムからは隠蔽されている。

2.16.3 関数

2.16.3.1 MFLT ∗ mflt_get (MSymbol *name*)

指定された名前を持つ FLT オブジェクトを返す.

関数 **mflt_get()** (p. 108) は、**name** という名前を持つ FLT オブジェクトを返す。

戻り値:
もし成功すれば、mflt_get() は見つかった FLT オブジェクトへのポインタを返す。失敗した場合は `NULL` を返す。

2.16.3.2 MFLT ∗ mflt_find (int *c*, MFLTFont ∗ *font*)

指定された文字とフォントに合った FLT を探す.

関数 **mflt_find()** (p. 108) は、文字 **c** をフォント **font** でレイアウトするために最も適切な FLT を返す。

戻り値:
もし成功すれば、mflt_find() は見つかった FLT オブジェクトへのポインタを返す。失敗した場合は `NULL` を返す。

2.16.3.3 const char ∗ mflt_name (MFLT ∗ *flt*)

FLT の名前を返す.

関数 **mflt_name()** (p. 108) は **flt** の名前を返す。

2.16.3.4 MCharTable * mflt_coverage (MFLT * *flt*)

FLT の範囲を返す.

関数 **mflt_coverage()** (p. 109) は、**flt** がサポートする文字に対して 0 でない値を含む文字テーブルを返す。

2.16.3.5 int mflt_run (MFLTGlyphString * *gstring*, int *from*, int *to*, MFLTFont * *font*, MFLT * *flt*)

FLT を使って文字をレイアウトする.

関数 **mflt_run()** (p. 109) は、**gstring** 中の **from** から **to** 直前までの文字を **font** を用いてレイアウトする。もし **flt** がゼロでなければ、その値をすべての文字に対して用いる。 そうでなければ適切な FLT を自動的に選択する。

戻り値:

>=0 実行成功を示す。返される値は、**gstring->glyphs** 中で以前 **to** によって示されていたグリフへのインデクスである。

-2 結果を格納するには **gstring->glyphs** が短すぎることを示す。 呼び出し側は、より長い **gstring->glyphs** を用いて再度この関数を呼ぶことができる。

-1 その他のエラーが起きたことを示す。

2.16.3.6 MFLT* mdebug_dump_flt (MFLT * *flt*, int *indent*)

2.16.4 変数

2.16.4.1 MSymbol(* mflt_font_id)(struct _MFLTFont *font)

2.16.4.2 int(* mflt_iterate_otf_feature)(struct _MFLTFont *font, MFLTOtfSpec *spec, int from, int to, unsigned char *table)

2.16.4.3 int(* mflt_iterate_otf_feature)(struct _MFLTFont *font, MFLTOtfSpec *spec, int from, int to, unsigned char *table)

2.16.4.4 MSymbol(* mflt_font_id)(struct _MFLTFont *font)

2.17 GUI API

libm17n-gui.so が提供する API

モジュール

- **フレーム**
  フレーム **とはグラフィックデバイスに対応するオブジェクトである**.
- **フォント**
  フォントオブジェクト.
- **フォントセット**
  フォントセットは文字からフォントへの対応付けを行うオブジェクトである.
- **フェース**
  フェースとは、*M-text* **の見栄えを制御するオブジェクトである**.
- **表示**
  M-text **をウィンドウに描画する**.
- **入力メソッド (GUI)**
  ウィンドウシステム上の入力メソッドのサポート.

2.17.1 説明

libm17n-gui.so が提供する API

ウィンドウシステム上の GUI サポート.

このセクションはウィンドウシステムのもとでの M-text の表示と入力にかかわる m17n GUI API を定義する。

ここでのすべての定義はウィンドウシステムとは独立である。 しかし、実際のライブラリファイルは個別のウィンドウシステムに依存する場合がある。 たとえばライブラリファイル m17n-X.so は、m17n GUI API の X ウィンドウ用の実装例である。

現実には、GUI API は主にツールキットライブラリ向けであるか、または XOM を実装するために用いられており、アプリケーションプログラムからの直接の利用を念頭においたものではない。

2.18 フレーム

フレーム とはグラフィックデバイスに対応するオブジェクトである.

変数： フレームパラメータ用キー

フレームを生成する際のパラメータに用いるシンボル。詳しくは関数 **mframe()** (p. 112) の説明参照。

Mdevice、 Mdisplay、 Mscreen、 Mdrawable、 Mdepth、 Mcolormap はフレームプロパティのキーでもある。

- **MSymbol Mdevice**
- **MSymbol Mdisplay**
- **MSymbol Mscreen**
- **MSymbol Mdrawable**
- **MSymbol Mdepth**
- **MSymbol Mcolormap**
- **MSymbol Mwidget**
- **MSymbol Mgd**

変数： フレームプロパティのキー

関数 **mframe_get_prop()** (p. 113) の引数に用いられるシンボル。

- **MSymbol Mfont**
- **MSymbol Mfont_width**
- **MSymbol Mfont_ascent**
- **MSymbol Mfont_descent**

型定義

- typedef struct **MFrame MFrame**
  フレームの型宣言.

関数

- **MFrame** ∗ **mframe** (**MPlist** ∗plist)
  新しいフレームを作る.
- void ∗ **mframe_get_prop** (**MFrame** ∗frame, **MSymbol** key)
  フレームのプロパティの値を返す.

変数

- **MFrame** ∗ **mframe_default**
  デフォルトのフレーム.

2.18.1 説明

フレーム とはグラフィックデバイスに対応するオブジェクトである.

フレーム とは **MFrame** (p. 112) 型のオブジェクトであり、個々の表示／入力デバイスの情報を格納するために用いられる。 ほとんどすべての m17n GUI 関数は、引数としてフレームへのポインタを要求する。

2.18.2 型定義

2.18.2.1 typedef struct MFrame MFrame

フレームの型宣言.

MFrame (p. 112) は、**フレーム** オブジェクト用の型である。 個々のフレームは、それが対応する物理的な表示／入力デバイスの各種情報を保持する。

MFrame (p. 112) 型の内部構造は、アプリケーションプログラムからは見えない。 またその内容は使用するウィンドウシステムに依存する。また m17n-X ライブラリにおけるフレームは、X ウィンドウの *display* と *screen* に関する情報を持つ。

2.18.3 関数

2.18.3.1 MFrame∗ mframe (MPlist ∗ *plist*)

新しいフレームを作る.

関数 **mframe()** (p. 112) は **plist** 中のパラメータを持つ新しいフレームを作る。 **plist** は `NULL` でも良い。

plist に現われるキーのうちどれが認識されるかはウィンドウシステムに依存する。

以下のキーは常に認識される。

- **Mdevice**. 値は **Mx** (p. 124), **Mgd**, **Mnil** (p. 16) のいずれかでなくてはならない。

  値が **Mx** (p. 124) ならば、新しいフレームは X ウィンドウシステム用である。 このフレームと共に指定された引数 **MDrawWindow** (p. 140) は、 `Window` 型でなくてはならない。フレームは読み書きともに可能であり、すべての GUI 関数が使用できる。

  値が **Mgd** ならば、新しいフレームは GD ライブラリのイメージオブジェクト用である。このフレームと共に指定された引数 **MDrawWindow** (p. 140) は、 `gdImagePtr` 型でなくてはならない。フレームは書き出し専用であり、 minput_ で始まる名前の関数は使用できない。

  値が **Mnil** (p. 16) ならば、新しいフレームは, null デバイス用である。このフレームは読み書きできないので、引数 **MDrawWindow** (p. 140) を必要とする mdraw_ で始まる名前の関数や、minput_ で始まる名前の関数は使用できない。

- **Mface** (p. 138). 値は **MFace** (p. 131) へのポインタでなくてはならない。

  この値はフレームのデフォルトのフェースとして用いられる。

これらのキーに加え、**Mdevice** のキーが **Mx** (p. 124) である場合に限り以下のキーも認識される。以下のキーはルートウィンドウと、フレームで利用できる drawable の深さを指定する。

- **Mdrawable**. 値は `Drawable` 型でなくてはならない。

  キー **Mdisplay** を持つパラメータも指定されている必要がある。 生成されたフレームは、指定されたディスプレイ上の指定された drawable と同じルートウィンドウと深さを持つ drawable に用いられる。

  このパラメータがある場合には、**Mscreen** をキーとするパラメータは無視される。

- **Mwidget**. 値は `Widget` 型でなくてはならない。

  生成されたフレームは、指定したウィジェットと同じルートウィンドウと深さを持つ drawable に用いられる。

  キー **Mface** (p. 138) を持つパラメータがなければ、デフォルトのフェースはこの ウィジェットのリソースから作られる。

  このパラメータがある場合には、**Mdisplay**, **Mscreen**, **Mdrawable**, **Mdepth** をキーとするパラメータは無視される。

- **Mdepth**. 値は `unsigned` 型でなくてはならない。

  生成されたフレームは、指定した深さの drawable に用いられる。

- **Mscreen**. 値は `(Screen *)` 型でなくてはならない。

  生成したフレームは、指定したスクリーンと同じルートウィンドウを持ち、スクリーンのデフォルトの深さと同じ深さを持つ drawable に用いられる。

  このパラメータがある場合には、**Mdisplay** をキーとするパラメータは無視される。

- **Mdisplay**. 値は `(Display *)` 型でなくてはならない。

  生成されたフレームは、指定したディスプレイのデフォルトスクリーンと同じルートウィンドウと同じ深さを持つ drawables に用いられる。

- **Mcolormap**. 値は `(Colormap)` 型でなくてはならない。

  生成されたフレームは、指定したカラーマップを使用する。

- **Mfont** (p. 114). 値は、**Mx** (p. 124), **Mfreetype** (p. 124), **Mxft** (p. 124) のいずれか。

  生成されたフレームは指定したフォントバックエンドを使用する。値が **Mx** (p. 124) であれば X のコアフォント、**Mfreetype** (p. 124) であれば FreeType でサポートされているローカルフォント、**Mxft** (p. 124) であれば Xft ライブラリ経由で用いるローカルフォントを使用する。 複数のフォントバックエンドを使用したい場合には、このパラメータを複数回、異なる値で指定することができる。 指定したバックエンドがサポートされていないデバイスでは、このパラメータは無視される。

  このパラメータが無い場合には、デバイスでサポートされているすべてのフォントバックエンドを利用する。

戻り値:
成功すれば **mframe()** (p. 112) は新しいフレームへのポインタを返す。そうでなければ `NULL` を返す。

2.18.3.2 void* mframe_get_prop (MFrame * *frame*, MSymbol *key*)

フレームのプロパティの値を返す.

関数 **mframe_get_prop()** (p. 113) はフレーム **frame** のキー **key** を持つプロパティの値を返す。有効なキーとその値は以下の通り。

キー	値の型	値の意味
Mface	MFace *	デフォルトのフェース
Mfont	MFont *	デフォルトのフォント
Mfont_width	int	デフォルトのフォントの幅
Mfont_ascent	int	デフォルトのフォントの ascent
Mfont_descent	int	デフォルトのフォントの descent

m17n-X ライブラリでは、以下のキーも使用できる。

キー	値の型	値の意味
Mdisplay	Display *	フレームと関連付けられたディスプレイ
Mscreen	int	フレームと関連付けられたスクリーンのスクリーンナンバ
Mcolormap	Colormap	フレームのカラーマップ
Mdepth	unsigned	フレームの深さ

2.18.4 変数

2.18.4.1 MSymbol Mdevice

2.18.4.2 MSymbol Mdisplay

2.18.4.3 MSymbol Mscreen

2.18.4.4 MSymbol Mdrawable

2.18.4.5 MSymbol Mdepth

2.18.4.6 MSymbol Mcolormap

2.18.4.7 MSymbol Mwidget

2.18.4.8 MSymbol Mgd

2.18.4.9 MSymbol Mfont

2.18.4.10 MSymbol Mfont_width

2.18.4.11 MSymbol Mfont_ascent

2.18.4.12 MSymbol Mfont_descent

2.18.4.13 MFrame* mframe_default

デフォルトのフレーム.

外部変数 **mframe_default** (p. 114) は、デフォルトのフレームへのポインタを持つ。 デフォルトのフレームは、最初に **mframe()** (p. 112) が呼び出されたときに作られる。

2.19 フォント

フォントオブジェクト.

変数: フォントプロパティを指定する定義済みシンボル

- **MSymbol Mfoundry**
  開発元を指定するフォントプロパティのキー.
- **MSymbol Mfamily**
  ファミリを指定するフォントプロパティのキー.
- **MSymbol Mweight**
  太さを指定するフォントプロパティのキー.
- **MSymbol Mstyle**
  スタイルを指定するフォントプロパティのキー.
- **MSymbol Mstretch**
  幅を指定するフォントプロパティのキー.
- **MSymbol Madstyle**
  adstyle **を指定するフォントプロパティのキー**.
- **MSymbol Mspacing**
  spacing **を指定するフォントプロパティのキー**.
- **MSymbol Mregistry**
  レジストリを指定するフォントプロパティのキー.
- **MSymbol Msize**
  サイズを指定するフォントプロパティのキー.
- **MSymbol Motf**
- **MSymbol Mfontfile**
  フォントファイルを指定するフォントプロパティのキー.
- **MSymbol Mresolution**
  解像度を指定するフォントプロパティのキー.
- **MSymbol Mmax_advance**
- **MSymbol Mfontconfig**
  "fontconfig" **という名前を持つシンボル**.
- **MSymbol Mx**
  "x" **という名前を持つシンボル**.
- **MSymbol Mfreetype**
  "freetype" **という名前を持つシンボル**.
- **MSymbol Mxft**
  "xft" **という名前を持つシンボル**.

型定義

- typedef struct **MFont MFont**
  フォントの型宣言.

関数

- **MFont** ∗ **mfont** ()
  新しいフォントを作る.
- **MFont** ∗ **mfont_parse_name** (const char ∗name, **MSymbol** format)
  フォント名からフォントを作る.
- char ∗ **mfont_unparse_name** (**MFont** ∗font, **MSymbol** format)
  フォントからフォント名を作る.
- **MFont** ∗ **mfont_copy** (**MFont** ∗font)
  フォントのコピーを作る.
- void ∗ **mfont_get_prop** (**MFont** ∗font, **MSymbol** key)
  フォントのプロパティの値を得る.
- int **mfont_put_prop** (**MFont** ∗font, **MSymbol** key, void ∗val)
  フォントのプロパティに値を設定する.
- **MSymbol** ∗ **mfont_selection_priority** ()
  フォント選択の優先度を返す.
- int **mfont_set_selection_priority** (**MSymbol** ∗keys)
  フォント選択優先度を設定する.
- **MFont** ∗ **mfont_find** (**MFrame** ∗frame, **MFont** ∗spec, int ∗score, int max_size)
  フォントを探す.
- int **mfont_set_encoding** (**MFont** ∗font, **MSymbol** encoding_name, **MSymbol** repertory_name)
  フォントのエンコーディングを設定する.
- char ∗ **mfont_name** (**MFont** ∗font)
  フォント名からフォントを作る.
- **MFont** ∗ **mfont_from_name** (const char ∗name)
  フォントからフォント名を作る.
- int **mfont_resize_ratio** (**MFont** ∗font)
  フォントのリサイズ情報を得る
- **MPlist** ∗ **mfont_list** (**MFrame** ∗frame, **MFont** ∗font, **MSymbol** language, int maxnum)
  フォントのリストを得る
- **MPlist** ∗ **mfont_list_family_names** (**MFrame** ∗frame)

- int **mfont_check** (**MFrame** *frame, **MFontset** *fontset, **MSymbol** script, **MSymbol** language, **MFont** *font)
- int **mfont_match_p** (**MFont** *font, **MFont** *spec)
- **MFont** * **mfont_open** (**MFrame** *frame, **MFont** *font)
- **MFont** * **mfont_encapsulate** (**MFrame** *frame, **MSymbol** data_type, void *data)
- int **mfont_close** (**MFont** *font)

変数

- **MPlist** * **mfont_freetype_path**

  フォントファイルとフォントファイルを含むディレクトリのリスト.

2.19.1 説明

フォントオブジェクト.

m17n GUI API はフォントを `MFont` 型のオブジェクトとして表現する。 フォントは **フォントプロパティ** を持つことができる。他のタイプのプ ロパティ同様、フォントプロパティはキーと値からなり、キーは以下のシ ンボルのいずれかである。

`Mfoundry`, `Mfamily`, `Mweight`, `Mstyle`, `Mstretch`, `Madstyle`, `Mregistry`, `Msize`, `Mresolution`, `Mspacing`

フォントプロパティのキーが `Msize` あるいは `Mresolution` の場合、値は整数値であり、キーがそれ以外の場合、値はシンボルである。

「フォント F のフォントプロパティのうちキーが `Mxxx` であるもの」のことを簡単に「F の xxx プロパティ」と呼ぶことがある。

foundry プロパティの値は、adobe, misc 等のフォントの開発元情報を示すシンボルである。

family プロパティの値は、times, helvetica 等のフォントファミリーを示すシンボルである。

weight プロパティの値は、normal, bold 等の太さに関する情報を示すシンボルである。

style プロパティの値は、normal, italic 等のスタイルに関する情報を示すシンボルである。

stretch プロパティの値は、normal, semicondensed 等の文字幅に関する情報を示すシンボルである。

adstyle プロパティの値は、serif, sans-serif 等の抽象的なフォントファミリーに関する情報を示すシンボルである。

registry プロパティの値は、iso10646, iso8895-1 等のレジストリ情報を示すシンボルである。

size プロパティの値は、フォントのデザインサイズを表わす整数値であり、 単位は 1/10 ポイントである。

resolution プロパティの値は、想定されているデバイスの解像度を表わす 整数値であり、単位は dots per inch (dpi) である。

type プロパティの値は、フォントドライバを指示し、現在 Mx もしくは Mfreetype である。

m17n ライブラリはフォントオブジェクトを２つの目的で用いている。アプ リケーションプログラムからフォントの指定を受け取る目的と、アプリケー ションプログラムに利用可能なフォントを提示する目的である。アプリケー ションプログラムに対して提示を行う際には、フォントプロパティはすべ て具体的な値を持つ。

m17n ライブラリは Window システムフォント、FreeType フォント、 OpenType フォントの３種類をサポートしている。

- Window システムフォント

m17n X ライブラリは、X サーバと X フォントサーバが取り扱う全てのフォントをサポートする。XLFD の各フィールドとフォントプロパティの対応は以下の通り。この表にないフィールドは無視される。

XLFD フィールド	プロパティ
FOUNDRY	foundry
FAMILY_NAME	family
WEIGHT_NAME	weight
SLANT	style
SETWIDTH_NAME	stretch
ADD_STYLE_NAME	adstyle
PIXEL_SIZE	size
RESOLUTION_Y	resolution
CHARSET_REGISTRY-CHARSET_ENCODING	registry

- FreeType fonts

  m17n ライブラリは、FreeType ライブラリを使うように設定された場合には、 FreeType が扱うすべてのフォントをサポートする。変数 **mfont_freetype_path** (p. 124) は m17n ライブラリの設定と環境変数 M17NDIR に応じて初期化される。詳細は変数の説明を参照のこと。

  もし m17n ライブラリが fontconfig ライブラリを使うように設定された場合には、 **mfont_freetype_path** (p. 124) に加えて、 fontconfig で使用可能なフォントもすべてサポートされる。

  FreeType フォントのファミリ名は family プロパティに対応する。 FreeType フォントのスタイル名は、下の表のように weight, style, stretch プロパティに対応する。

スタイル名	weight	style	stretch
Regular	medium	r	normal
Italic	medium	i	normal
Bold	bold	r	normal
Bold Italic	bold	i	normal
Narrow	medium	r	condensed
Narrow Italic	medium	i	condensed
Narrow Bold	bold	r	condensed
Narrow Bold Italic	bold	i	condensed
Black	black	r	normal
Black Italic	black	i	normal
Oblique	medium	o	normal
BoldOblique	bold	o	normal

  上の表に現われないスタイル名は "Regular" として扱われる。

  platform ID と encoding ID の組み合わせが registry プロパティに対応する。たとえばあるフォントが (1 1) という ID の組合せを持てば、 registry プロパティは 1-1 となる。頻繁にあらわれる組合せには以下のような定義済み registry プロパティ が与えられている。

platform ID	encoding ID	registry プロパティ
0	3	unicode-bmp
0	4	unicode-full
1	0	apple-roman
3	1	unicode-bmp
3	1	unicode-full

  したがって、二つの組合せ (1 0) 、(3 1) を持つフォントは、それぞれ registry プロパティが 1-0, apple-roman, 3-1, unicode-bmp である 4 つのフォントオブジェクトに対応する。

- OpenType フォント

  m17n ライブラリは、FreeType ライブラリと OTF ライブラリを使用するように設定すれば、すべての OpenType フォントをサポートする。実際に利用できるフォントのリストは FreeType フォントの場合と同様に作られる。OpenType フォントを FLT (Font Layout Table) 経由で使用するようフォントセットに指定されており、FLT に OTF 関連のコマンド (たとえば otf:deva) があれば、OTF ライブラリがフォントの OpenType レイアウトテーブルに従って文字列をグリフコード列に変換し、FreeType ライブラリが各グリフのビットマップイメージを提供する。

2.19.2 型定義

2.19.2.1 typedef struct MFont MFont

フォントの型宣言.

MFont (p. 119) 型はフォント指定用の構造体であり、フォントのプロパティである foundry, family, weight, style, stretch, adstyle, registry, size, resolution に関する情報を含む。

この構造体はフォントセット内のフォントを指定する際と、使用可能なシステムフォントの情報を格納する際の両方に用いられる。

内部構造はアプリケーションプログラムからは見えない。

参照:
mfont() (p. 119), **mfont_from_name()** (p. 121), **mfont_find()** (p. 121).

2.19.3 関数

2.19.3.1 MFont* mfont ()

新しいフォントを作る.

関数 **mfont()** (p. 119) はプロパティを一切持たない新しいフォントをオブジェクトを作る。

戻り値:
この関数は作ったフォントオブジェクトへのポインタを返す。

2.19.3.2 MFont* mfont_parse_name (const char * *name*, MSymbol *format*)

フォント名からフォントを作る.

関数 **mfont_parse_name()** (p. 119) は、フォント名 **name** から取り出されたプロパティを持つ、新しいフォントオブジェクトを作る。

format は **name** のフォーマットを指定する。**format** が **Mx** (p. 124) であれば、 **name** は XLFD (X Logical Font Description) に従って解析される。 **format** が **Mfontconfig** (p. 124) であれば **name** は Fontfonfig のフォントテキスト表現に従って解析される。**format** が **Mnil** (p. 16) であれば、まず XLFD に従って解析され、それに失敗したら Fontconfig に従って解析される。

戻り値:
処理が成功すれば **mfont_parse_name()** (p. 119) は新しく作られたフォントへのポインタを返す。そうでなければ NULL を返す。

2.19.3.3 char* mfont_unparse_name (MFont * *font*, MSymbol *format*)

フォントからフォント名を作る.

関数 **mfont_unparse_name()** (p. 119) は フォント名の文字列をフォント **font** を元に **format** に従って作る。

format は **Mx** (p. 124) または **Mfontconfig** (p. 124) である。 **Mx** (p. 124) ならばフォント名は XLFD (X Logical Font Description) に従う。 **Mfontconfig** (p. 124) ならばフォント名は Fontconfig のフォントテキスト表現に従う。

戻り値:
この関数は新たにアロケートしたフォント名の文字列を返す。文字列は、ユーザが free() によって明示的に解放しない限り解放されない。

2.19.3.4 MFont* mfont_copy (MFont * *font*)

フォントのコピーを作る.

関数 Mfont_copy() はフォント **font** のコピーを作り、それを返す。

2.19.3.5 void* mfont_get_prop (MFont * *font*, MSymbol *key*)

フォントのプロパティの値を得る.

関数 **mfont_get_prop()** (p. 120) はフォント **font** のプロパティのうち、キーが **key** であるものの値を返す。**key** は以下のシンボルのいずれかでなけれ ばならない。

`Mfoundry`, `Mfamily`, `Mweight`, `Mstyle`, `Mstretch`, `Madstyle`, `Mregistry`, `Msize`, `Mresolution`, `Mspacing`.

戻り値:

key が `Mfoundry`, `Mfamily`, `Mweight`, `Mstyle`, `Mstretch`, `Madstyle`, `Mregistry`, `Mspacing` のいずれかであれば、 相当する値をシンボルとして返す。フォントがそのプロパティを持たない 場合には `Mnil` を返す。**key** が `Msize` あるいは `Mresolution` の 場合には、相当する値をは整数値として返す。フォントがそのプロパティ を持たない場合には 0 を返す。**key** がそれ以外のものであれば、`NULL` を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

2.19.3.6 int mfont_put_prop (MFont * *font*, MSymbol *key*, void * *val*)

フォントのプロパティに値を設定する.

関数 **mfont_put_prop()** (p. 120) は、フォント **font** のキーが **key** であるプロパ ティの値を **val** に設定する。**key** は以下のシンボルのいずれかである。

`Mfoundry`, `Mfamily`, `Mweight`, `Mstyle`, `Mstretch`, `Madstyle`, `Mregistry`, `Msize`, `Mresolution`.

key が `Msize` か `Mresolution` であれば **val** は整数値でなくては らない。それ以外の場合、**val** はプロパティ値の名前のシンボルでなくて はならない。ただしもしその名前が "nil" の場合は、名前が "Nil" のシンボルでなくてはならない。

2.19.3.7 MSymbol* mfont_selection_priority ()

フォント選択の優先度を返す.

関数 **mfont_selection_priority()** (p. 120) は 6 つのシンボルからなる配列を作って返す。 配列の要素は、以下のフォントプロパティのキーを優先度順に並べたものである。

`Mfamily`, `Mweight`, `Mstyle`, `Mstretch`, `Madstyle`, `Msize`.

m17n ライブラリはこの配列に従って、最も合致するフォントを選択する。 目的のフォントと、それぞれ違うプロパティの値が合致しないフォントがあった場合、優先度の低いプロパティの値が合致しないフォント（優先度の高いプロパティの値が合致しているフォント）が選択される。

2.19.3.8 int mfont_set_selection_priority (MSymbol * *keys*)

フォント選択優先度を設定する.

関数 **mfont_set_selection_priority()** (p. 120) は、6 つのシンボルの配列 **keys** にしたがってフォント選択優先度を設定する。配列は以下の各要素を適切 な順番で並べたものである。

`Mfamily`, `Mweight`, `Mstyle`, `Mstretch`, `Madstyle`, `Msize`.

詳細は関数 **mfont_selection_priority()** (p. 120) の説明を参照のこと。

2.19.3.9 MFont* mfont_find (MFrame * *frame*, MFont * *spec*, int * *score*, int *max_size*)

フォントを探す.

関数 **mfont_find()** (p. 121) は、フレーム **frame** 上でフォント定義 **spec** にもっとも合致する利用可能なフォントへのポインタを返す。

score は NULL であるか、見つかったフォントが **spec** にどれほど合っているかを示すスコアを保存する場所へのポインタである。 スコアが小さいほど良く合っていることを意味する。

2.19.3.10 int mfont_set_encoding (MFont * *font*, MSymbol *encoding_name*, MSymbol *repertory_name*)

フォントのエンコーディングを設定する.

関数 **mfont_set_encoding()** (p. 121) はフォント **font** のエンコーディング情報を設定する。

encoding_name はフォントと同じエンコーディングを持つ文字セットを示すシンボルである。

repertory_name は `Mnil` であるか、フォントと同じエンコーディングを持つ文字セットを示すシンボルである。 `Mnil` であれば、個々の文字がそのフォントでサポートされているかどうかは、各々のフォントドライバに問い合わせる。

戻り値:
処理が成功すればこの関数は 0 を返す。そうでなければ -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

2.19.3.11 char* mfont_name (MFont * *font*)

フォント名からフォントを作る.

この関数は廃止予定である。 **mfont_unparse_name()** (p. 119) を使用のこと。

2.19.3.12 MFont* mfont_from_name (const char * *name*)

フォントからフォント名を作る.

これは関数は廃止予定である。 **mfont_parse_name()** (p. 119) を使用のこと。

2.19.3.13 int mfont_resize_ratio (MFont * *font*)

フォントのリサイズ情報を得る

関数 mfont_resize_ratio は m17n データベース <font, reisize> を検索し、フォント FONT のリサイズの比率（パーセンテージ） を返す。たとえば返す値が 150 であれば、m17n ライブラリは指定されたサイズの 1.5 倍のフォントを使用することを意味する。

2.19.3.14 MPlist* mfont_list (MFrame * *frame*, MFont * *font*, MSymbol *language*, int *maxnum*)

フォントのリストを得る

関数 **mfont_list()** (p. 121) はフレーム **frame** で利用可能なフォントのリストを 返す。**font** が NULL でなければ、**font** と合致する利用可能なフォント のリストを返す。**language** が `Mnil` でなければ、**language** を

サポートする利用可能なフォントのリストを返す。**maxnum** は、0 より大きい場合には、返すフォントの数の上限である。

ただし、引数 **language** は旧版との整合性のためだけにあり、その使用は勧められない。フォントの **Mlanguage** (p. 47) プロパティを使うべきである。もし **font** がすでにこのプロパティを持っていたら、引数 **language** は無

戻り値:

この関数はキーがフォントファミリ名であり値が MFont オブジェクトへのポインタであるような plist を返す。plist は **m17n_object_unref()** (p. 12) で解放する必要がある。フォントが見つからなければ NULL を返す。

2.19.3.15 MPlist* mfont_list_family_names (MFrame * *frame*)

2.19.3.16 int mfont_check (MFrame * *frame*, MFontset * *fontset*, MSymbol *script*, MSymbol *language*, MFont * *font*)

2.19.3.17 int mfont_match_p (MFont * *font*, MFont * *spec*)

2.19.3.18 MFont* mfont_open (MFrame * *frame*, MFont * *font*)

2.19.3.19 MFont* mfont_encapsulate (MFrame * *frame*, MSymbol *data_type*, void * *data*)

2.19.3.20 int mfont_close (MFont * *font*)

2.19.4 変数

2.19.4.1 MSymbol Mfoundry

開発元を指定するフォントプロパティのキー.

変数 **Mfoundry** (p. 122) は "foundry" という名前を持つシンボルであり、フォントプロパティとフェースプロパティのキーとして用いられる。値は、フォントの開発元名を名前として持つシンボルである。

2.19.4.2 MSymbol Mfamily

ファミリを指定するフォントプロパティのキー.

変数 **Mfamily** (p. 122) は "family" という名前を持つシンボルであり、フォントプロパティとフェースプロパティのキーとして用いられる。値は、フォントのファミリ名を名前として持つシンボルである。

2.19.4.3 MSymbol Mweight

太さを指定するフォントプロパティのキー.

変数 **Mweight** (p. 122) は "weight" という名前を持つシンボルであり、フォントプロパティとフェースプロパティのキーとして用いられる。値は、フォントの太さ名 ("medium", "bold" 等) を名前として持つシンボルである。

2.19.4.4 MSymbol Mstyle

スタイルを指定するフォントプロパティのキー.

変数 **Mstyle** (p. 122) は "style" という名前を持つシンボルであり、フォントプロパティとフェースプロパティのキーとして用いられる。 値は、フォントのスタイル名 ("r", "i", "o" 等) を名前として持つシンボルである。

2.19.4.5 MSymbol Mstretch

幅を指定するフォントプロパティのキー.

変数 **Mstretch** (p. 123) は "stretch" という名前を持つシンボルであり、フォントプロパティとフェースプロパティのキーとして用いられる。 値は、フォントの文字幅名 ("normal", "condensed" 等) を名前として持つシンボルである。

2.19.4.6 MSymbol Madstyle

adstyle を指定するフォントプロパティのキー.

変数 **Madstyle** (p. 123) は "adstyle" という名前を持つシンボルであり、フォントプロパティとフェースプロパティのキーとして用いられる。 値は、フォントの adstyle 名 ("serif", "", "sans" 等) を名前として持つシンボルである。

2.19.4.7 MSymbol Mspacing

spacing を指定するフォントプロパティのキー.

変数 **Mspacing** (p. 123) は "spacing" という名前を持つシンボルであり、 フォントプロパティのキーとして用いられる。値は、フォントの spacing 特性を示す名前 ("p", "m" 等) を持つシンボルである。

2.19.4.8 MSymbol Mregistry

レジストリを指定するフォントプロパティのキー.

変数 **Mregistry** (p. 123) は "registry" という名前を持つシンボルであり、フォントプロパティとフェースプロパティのキーとして用いられる。 値は、フォントのレジストリ名 ("iso8859-1", "jisx0208.1983-0" 等) を名前として持つシンボルである。

2.19.4.9 MSymbol Msize

サイズを指定するフォントプロパティのキー.

変数 **Msize** (p. 123) は "size" という名前を持つシンボルであり、フォントプロパティとフェースプロパティのキーとして用いられる。値は、 100 dpi のディスプレイ上でのフォントのデザインサイズを 1/10 ポイント単位で示す整数値である。

2.19.4.10 MSymbol Motf

2.19.4.11 MSymbol Mfontfile

フォントファイルを指定するフォントプロパティのキー.

変数 **Mfontfile** (p. 123) は "fontfile" という名前を持つシンボルであ り、フォントプロパティのキーとして用いられる。値は、フォントファイ ル名を名前として持つとするシンボルである。

2.19.4.12 MSymbol Mresolution

解像度を指定するフォントプロパティのキー.

変数 **Mresolution** (p. 124) は "resolution" という名前を持つシンボルであり、フォントプロパティとフェースプロパティのキーとして用いられる。 値は、フォントの解像度を dots per inch (dpi) 単位で示す整数値である。

2.19.4.13 MSymbol Mmax_advance

2.19.4.14 MSymbol Mfontconfig

"fontconfig" という名前を持つシンボル.

変数 **Mfontconfig** (p. 124) は関数 **mfont_parse_name()** (p. 119) と **mfont_unparse_name()** (p. 119) の引数として用いられる。

2.19.4.15 MSymbol Mx

"x" という名前を持つシンボル.

変数 **Mx** (p. 124) は構造 **MDrawGlyph** (p. 166) のメンバ <type> の値として用いられ、メンバ <fontp> の型が実際には (XFontStruct ∗) であることを表す.

2.19.4.16 MSymbol Mfreetype

"freetype" という名前を持つシンボル.

変数 **Mfreetype** (p. 124) は構造 **MDrawGlyph** (p. 166) のメンバ <type> の値として用いられ、メンバ <fontp> の型が実際には FT_Face であることを表す。

2.19.4.17 MSymbol Mxft

"xft" という名前を持つシンボル.

変数 **Mxft** (p. 124) は構造 **MDrawGlyph** (p. 166) のメンバ <type> の値として用いられ、メンバ <fontp> の型が実際には (XftFont ∗) であることを表す。

2.19.4.18 MPlist∗ mfont_freetype_path

フォントファイルとフォントファイルを含むディレクトリのリスト.

変数 mfont_freetype_path は、フォントファイルとフォントファイルを含むディレクトリの plist である。各要素のキーは Mstring であり、値はフォントファイルかディレクトリを示す文字列である。

マクロ **M17N_INIT()** (p. 7) によって、この変数は m17n データベースと環境変数 "M17NDIR" 双方のサブディレクトリ "fonts" を含むように設定される。 **mframe()** (p. 112) の最初の呼び出しの際に、この変数から実際に使用できるフォントの内部リストが作られる。 そこでアプリケーションプログラムは、mframe() を呼ぶ前に（必要ならば）この変数を変更しなくてはならない。 新しい要素を追加する場合には、その値は安全に開放できる文字列でなくてはならない。

m17n ライブラリが FreeType ライブラリを使うように設定されてない場合には、この変数は用いられない。

2.20 フォントセット

フォントセットは文字からフォントへの対応付けを行うオブジェクトである.

関数

- **MFontset** ∗ **mfontset** (char ∗name)
  フォントセットを返す.
- **MSymbol mfontset_name** (**MFontset** ∗fontset)
  フォントセットの名前を返す.
- **MFontset** ∗ **mfontset_copy** (**MFontset** ∗fontset, char ∗name)
  フォントセットのコピーを作る.
- int **mfontset_modify_entry** (**MFontset** ∗fontset, **MSymbol** script, **MSymbol** language, **MSymbol** charset, **MFont** ∗spec, **MSymbol** layouter_name, int how)
  フォントセットの内容を変更する.
- **MPlist** ∗ **mfontset_lookup** (**MFontset** ∗fontset, **MSymbol** script, **MSymbol** language, **MSymbol** charset)
  フォントセットを検索する.

2.20.1 説明

フォントセットは文字からフォントへの対応付けを行うオブジェクトである.

フォントセット は `MFontset` 型のオブジェクトである。M-text の表示の際、フォントセットは以下の情報を用いて M-text 中の個々の文字にどのフォントを用いるか決める規則を与える。

- 文字の文字プロパティ "スクリプト"
- 文字のテキストプロパティ "言語"
- 文字のテキストプロパティ "文字セット"

これらの情報がどのように用いられるかは **mdraw_text()** (p. 141) の説明を参照のこと。

2.20.2 関数

2.20.2.1 MFontset ∗ mfontset (char ∗ *name*)

フォントセットを返す.

関数 **mfontset()** (p. 125) は名前 **name** を持つフォントセットオブジェクトへのポインタを返す。 **name** が `NULL` ならば、デフォルトフォントセットへのポインタを返す。

name という名前を持つフォントセットがなければ、新しいものが作られる。その際、 m17n データベースに <`fontset`, **name**> というデータがあれば、フォントセットはそのデータに沿って初期化される。 なければ、空のままにされる。

マクロ **M17N_INIT()** (p. 7) はデフォルトのフォントセットを作る。アプリケーションプログラムは **mframe()** (p. 112) を初めて呼ぶまでの間はデフォルトフォントセットを変更することができる。

戻り値:
この関数は見つかった、あるいは作ったフォントセットへのポインタを返す。

2.20.2.2 MSymbol mfontset_name (MFontset * *fontset*)

フォントセットの名前を返す.

関数 **mfontset_name()** (p. 126) はフォントセット **fontset** の名前を返す。

2.20.2.3 MFontset * mfontset_copy (MFontset * *fontset*, char * *name*)

フォントセットのコピーを作る.

関数 **mfontset_copy()** (p. 126) はフォントセット **fontset** のコピーを作って、名前 **name** を与え、そのコピーへのポインタを返す。**name** は既存のフォントセットの名前であってはならない。そのような場合にはコピーを作らずに NULL を返す。

2.20.2.4 int mfontset_modify_entry (MFontset * *fontset*, MSymbol *script*, MSymbol *language*, MSymbol *charset*, MFont * *spec*, MSymbol *layouter_name*, int *how*)

フォントセットの内容を変更する.

関数 **mfontset_modify_entry()** (p. 126) は、**language** と **script** の組み合わせ、または **charset** に対して **font** のコピーを使うように、フォントセット **fontset** を設定する。

フォントセット中の各フォントは、特定のスクリプトと言語のペア、特定の文字セット、シンボル `Mnil` のいずれかと関連付けられている。同じものと関連付けられたフォントはグループを構成する。

script は `Mnil` であるか、スクリプトを特定するシンボルである。 シンボルである場合には、**language** は言語を特定するシンボルか `Mnil` であり、**font** は the **script / language** ペアに関連付けられる。

charset は `Mnil` であるか、文字セットオブジェクトを表すシンボルである。 シンボルである場合には **font** はその文字セットと関連付けられる。

script と **charset** の双方が `Mnil` でない場合には **font** のコピーが 2 つ作られ、それぞれ **script / language** ペアと文字セットに関連付けられる。

script と **charset** の双方が `Mnil` ならば、 **font** は `Mnil` と関連付けられる。この種のフォントは *fallback font* と呼ばれる。

引数 **how** は **font** の優先度を指定する。**how** が正ならば、**font** は同じものと関連付けられたグループ中で最高の優先度を持つ。**how** が負ならば、最低の優先度を持つ。**how** が 0 ならば、**font** は関連付けられたものに対する唯一の利用可能なフォントとなり、他のフォントはグループから取り除かれる。

layouter_name は `Mnil` であるか、**フォントレイアウトテーブル** (p. 207) （フォントレイアウトテーブル）を示すシンボルである。シンボルであれば、**font** を用いて M-text を表示する際には、そのフォントレイアウトテーブルを使って文字列からグリフコード列を生成する。

戻り値:
処理が成功したとき、mfontset_modify_entry() は 0 を返す。 失敗したときは -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
`MERROR_SYMBOL`

2.20.2.5 MPlist * mfontset_lookup (MFontset * *fontset*, MSymbol *script*, MSymbol *language*, MSymbol *charset*)

フォントセットを検索する.

関数 **mfontset_lookup()** (p. 127) は **fontset** を検索し、**fontset** の内容のうち指定したスクリプト、言語、文字セットに対応する部分を表す plist を返す。

script が Mt ならば、返す plist のキーはフォントが指定されているスクリプト名のシンボルであり、値は NULL である。

script がスクリプト名のシンボルであれば、返す plist は **language によって定まる。**

- **language** が Mt ならば、plist のキーはフォントが指定されている言語名のシンボルであり、値は NULL である。キーは Mt であることもあり、その場合そのスクリプトにフォールバックフォントがあることを意味する。

- **language** が言語名のシンボルならば、plist は指定のスクリプトと言語に対する FONT-GROUP である。FONT-GROUP とは、キーが FLT (FontLayoutTable) 名のシンボルであり、値が **MFont** (p. 119) へのポインタであるような plist である。ただしフォントに FLT が対応付けられていない時には、キーは Mt になる。

- **language** が Mnil ならば、plist はそのスクリプト用のフォールバック FONT-GROUP である。

script が Mnil ならば、返す plist は以下のように定まる。

- **charset** が Mt ならば、plist のキーはフォントが指定されている文字セット名のシンボルであり、値は NULL である。

- **charset** が文字セット名のシンボルならば、plist はその文字セット用の FONT-GROUP である。

- **charset** が Mnil ならば、plist はフォールバック FONT-GROUP である。

戻り値:

この関数はフォントセットの内容を表す plist を返す。 plist は **m17n_object_unref()** (p. 12) で解放されるべきである。

2.21 フェース

フェースとは、M-text の見栄えを制御するオブジェクトである.

データ構造

- struct **MFaceHLineProp**
  フェースの水平線指定用型宣言.
- struct **MFaceBoxProp**
  フェースの囲み枠指定用型宣言.

変数: フェースプロパティのキー

- **MSymbol Mforeground**
  前景色を指定するフェースプロパティーのキー.
- **MSymbol Mbackground**
  背景色を指定するためのフェースプロパティーのキー.
- **MSymbol Mvideomode**
  ビデオモードを指定するためのフェースプロパティーのキー.
- **MSymbol Mratio**
  フォントのサイズの比率を指定するためのフェースプロパティーのキー.
- **MSymbol Mhline**
  水平線を指定するためのフェースプロパティーのキー.
- **MSymbol Mbox**
  囲み枠を指定するためのフェースプロパティーのキー.
- **MSymbol Mfontset**
  フォントセットを指定するためのフェースプロパティーのキー.
- **MSymbol Mhook_func**
  フックを指定するためのフェースプロパティーのキー.
- **MSymbol Mhook_arg**
  フックの引数を指定するためのフェースプロパティーのキー.

変数： フェースの #Mvideomode プロパティの可能な値

変数 **Mvideomode** (p. 134) の説明を参照のこと。

- **MSymbol Mnormal**
- **MSymbol Mreverse**

変数: 定義済みフェース

- **MFace * mface_normal_video**
  標準ビデオフェース.
- **MFace * mface_reverse_video**
  リバースビデオフェース.
- **MFace * mface_underline**
  下線フェース.
- **MFace * mface_medium**
  ミディアムフェース.
- **MFace * mface_bold**
  ボールドフェース.
- **MFace * mface_italic**
  イタリックフェース.
- **MFace * mface_bold_italic**
  ボールドイタリックフェース.
- **MFace * mface_xx_small**
  最小のフェース.
- **MFace * mface_x_small**
  より小さいフェース.
- **MFace * mface_small**
  小さいフェース.
- **MFace * mface_normalsize**
  標準の大きさのフェース.
- **MFace * mface_large**
  大きいフェース.
- **MFace * mface_x_large**
  もっと大きいフェース.
- **MFace * mface_xx_large**
  最大のフェース.
- **MFace * mface_black**
  黒フェース.
- **MFace * mface_white**
  白フェース.
- **MFace * mface_red**
  赤フェース.

- **MFace** ∗ **mface_green**
  緑フェース.
- **MFace** ∗ **mface_blue**
  青フェース.
- **MFace** ∗ **mface_cyan**
  シアンフェース.
- **MFace** ∗ **mface_yellow**
  黄フェース.
- **MFace** ∗ **mface_magenta**
  マゼンタフェース.

変数: フェースを取り扱うためのその他のシンボル

- **MSymbol Mface**
  フェースを指定するテキストプロパティのキー.

型定義

- typedef struct **MFace MFace**
  フェースの型宣言.
- typedef void(∗ **MFaceHookFunc**)(**MFace** ∗face, void ∗arg, void ∗info)
  フェースのフック関数の型宣言.

関数

- **MFace** ∗ **mface** ()
  新しいフェースをつくる.
- **MFace** ∗ **mface_copy** (**MFace** ∗face)
  フェースのコピーを作る.
- int **mface_equal** (**MFace** ∗face1, **MFace** ∗face2)
- **MFace** ∗ **mface_merge** (**MFace** ∗dst, **MFace** ∗src)
  フェースを統合する.
- **MFace** ∗ **mface_from_font** (**MFont** ∗font)
  フォントからフェースを作る.
- void ∗ **mface_get_prop** (**MFace** ∗face, **MSymbol** key)
  フェースのプロパティの値を得る.
- **MFaceHookFunc mface_get_hook** (**MFace** ∗face)

フェースのフック関数を得る.

- int **mface_put_prop** (**MFace** ∗face, **MSymbol** key, void ∗val)

  フェースプロパティの値を設定する.

- int **mface_put_hook** (**MFace** ∗face, **MFaceHookFunc** func)

  フェースのフック関数を設定する.

- void **mface_update** (**MFrame** ∗frame, **MFace** ∗face)

  フェースを更新する.

2.21.1 説明

フェースとは、M-text の見栄えを制御するオブジェクトである.

フェース は **MFace** (p. 131) 型のオブジェクトであり、M-text の表示方法を制御する。フェースは固定個の **フェースプロパティ** を持つ。 他のプロパティ同様フェースプロパティはキーと値からなり、キーは以下のシンボルのいずれかである。

Mforeground (p. 133), **Mbackground** (p. 134), **Mvideomode** (p. 134), **Mhline** (p. 134), **Mbox** (p. 134), **Mfoundry** (p. 122), **Mfamily** (p. 122), **Mweight** (p. 122), **Mstyle** (p. 122), **Mstretch** (p. 123), **Madstyle** (p. 123), **Msize** (p. 123), **Mfontset** (p. 134), **Mratio** (p. 134), **Mhook_func** (p. 135), **Mhook_arg** (p. 135)

「フェース F のフェースプロパティのうちキーが `Mxxx` であるもの」のことを簡単に「F の xxx プロパティ」と呼ぶことがある。

M-text の表示関数は、まず最初にその M-text からキーがシンボル **Mface** (p. 138) であるようなテキストプロパティを探し、次にその値に従って M-text を表示する。この値はフェースオブジェクトへのポインタでなければならない。

M-text が、**Mface** (p. 138) をキーとするテキストプロパティを複数持っており、かつそれらの値が衝突しないならば、フェース情報は組み合わされて用いられる。

あるテキスト属性がどのフェースによっても指定されていない場合は、デフォルトフェースの値が用いられる。

2.21.2 型定義

2.21.2.1 typedef struct MFace MFace

フェースの型宣言.

MFace (p. 131) 型はフェースオブジェクトのための構造体である。 内部構造はアプリケーションプログラムからは見えない。

2.21.2.2 typedef void(∗ MFaceHookFunc)(MFace ∗face, void ∗arg, void ∗info)

フェースのフック関数の型宣言.

MFaceHookFunc (p. 131) はフェースのフック関数の型である。

2.21.3 関数

2.21.3.1 MFace* mface ()

新しいフェースをつくる.

関数 **mface()** (p. 132) はプロパティを一切持たない新しいフェースオブジェクトを作る。

戻り値:
この関数は作ったフェースへのポインタを返す。

2.21.3.2 MFace* mface_copy (MFace * *face*)

フェースのコピーを作る.

関数 **mface_copy()** (p. 132) はフェース **face** のコピーを作り、そのコピーへのポインタを返す。

2.21.3.3 int mface_equal (MFace * *face1*, MFace * *face2*)

2.21.3.4 MFace* mface_merge (MFace * *dst*, MFace * *src*)

フェースを統合する.

関数 **mface_merge()** (p. 132) は、フェース **src** のプロパティをフェース **dst** に統合する。

戻り値:
この関数は **dst** を返す。

2.21.3.5 MFace* mface_from_font (MFont * *font*)

フォントからフェースを作る.

関数 **mface_from_font()** (p. 132) はフォント **font** のプロパティをプロパティとして持つ新しいフェースを作り、それを返す。

2.21.3.6 void* mface_get_prop (MFace * *face*, MSymbol *key*)

フェースのプロパティの値を得る.

関数 **mface_get_prop()** (p. 132) は、フェース **face** が持つフェースプロパティの内、キーが **key** であるものの値を返す。 **key** は下記のいずれかでなければならない。

Mforeground (p. 133), **Mbackground** (p. 134), **Mvideomode** (p. 134), **Mhline** (p. 134), **Mbox** (p. 134), **Mfoundry** (p. 122), **Mfamily** (p. 122), **Mweight** (p. 122), **Mstyle** (p. 122), **Mstretch** (p. 123), **Madstyle** (p. 123), **Msize** (p. 123), **Mfontset** (p. 134), **Mratio** (p. 134), **Mhook_arg** (p. 135)

戻り値:
戻り値の型は **key** に依存する。上記のキーの説明を参照すること。 エラーが検出された場合は NULL を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

参照:
mface_put_prop() (p. 133), **mface_put_hook()** (p. 133)

エラー:
MERROR_FACE

2.21.3.7 MFaceHookFunc mface_get_hook (MFace ∗ *face*)

フェースのフック関数を得る.

関数 **mface_get_hook()** (p. 133) はフェース **face** のフック関数を返す。

2.21.3.8 int mface_put_prop (MFace ∗ *face*, MSymbol *key*, void ∗ *val*)

フェースプロパティの値を設定する.

関数 **mface_put_prop()** (p. 133) は、フェース **face** 内でキーが **key** であるプロパティの値を **val** に設定する。**key** は以下のいずれかでなくてはならない。

Mforeground (p. 133), **Mbackground** (p. 134), **Mvideomode** (p. 134), **Mhline** (p. 134), **Mbox** (p. 134), **Mfoundry** (p. 122), **Mfamily** (p. 122), **Mweight** (p. 122), **Mstyle** (p. 122), **Mstretch** (p. 123), **Madstyle** (p. 123), **Msize** (p. 123), **Mfontset** (p. 134), **Mratio** (p. 134), **Mhook_func** (p. 135), **Mhook_arg** (p. 135).

これらのうちの、フォント関連のプロパティ (**Mfamily** (p. 122) から **Msize** (p. 123) まで) は、フェースのフォントセット中のフォントに関するデフォルト値となり、個々のフォントが値を指定しなかった場合に用いられる。

戻り値の型は **key** に依存する。上記のキーの説明を参照すること。

戻り値:
処理が成功した場合、mface_put_prop() は 0 を返す。失敗した場合は -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

参照:
mface_get_prop() (p. 132)

エラー:
`MERROR_FACE`

2.21.3.9 int mface_put_hook (MFace ∗ *face*, MFaceHookFunc *func*)

フェースのフック関数を設定する.

関数 mface_set_hook() は、フェース **face** のフック関数を **func** に設 定する。

2.21.3.10 void mface_update (MFrame ∗ *frame*, MFace ∗ *face*)

フェースを更新する.

関数 **mface_update()** (p. 133) はフレーム **frame** のフェース **face** を **face** のフック関数を（あれば）呼んで更新する。

2.21.4 変数

2.21.4.1 MSymbol Mforeground

前景色を指定するフェースプロパティーのキー.

変数 **Mforeground** (p. 133) はフェースプロパティのキーとして用いられる。 プロパティの値は、色名を名前として持つシンボルか **Mnil** (p. 16) である。

Mnil (p. 16) の場合、前景色は指定されない。そうでなければ M-text の前景は指定された色で表示される。

2.21.4.2 MSymbol Mbackground

背景色を指定するためのフェースプロパティーのキー.

変数 **Mbackground** (p. 134) はフェースプロパティのキーとして用いられる。 プロパティの値は、色名を名前として持つシンボルか **Mnil** (p. 16) である。

Mnil (p. 16) の場合、背景色は指定されない。そうでなければ M-text の背景は指定された色で表示される。

2.21.4.3 MSymbol Mvideomode

ビデオモードを指定するためのフェースプロパティーのキー.

変数 **Mvideomode** (p. 134) はフェースプロパティのキーとして用いられる。プロパティの値は、 **Mnormal**, **Mreverse**, **Mnil** (p. 16) のいずれかでなくてはならない。

Mnormal の場合は、M-text を標準のビデオモード（前景を前景色で、背景を背景色で）で表示する。

Mreverse の場合はリバースビデオモードで（前景を背景色で、背景を前景色で）表示する。

Mnil (p. 16) の場合はビデオモードは指定されない。

2.21.4.4 MSymbol Mratio

フォントのサイズの比率を指定するためのフェースプロパティーのキー.

変数 **Mratio** (p. 134) はフェースプロパティのキーとして用いられる。値 RATIO は整数値でなくてはならない。

値が 0 ならば、フォントサイズは指定されない。そうでなければ、M-text は (FONTSIZE * RATIO / 100) というサイズのフォントで表示される。 FONTSIZE はフェースプロパティー **Msize** (p. 123) で指定されたサイズである。

2.21.4.5 MSymbol Mhline

水平線を指定するためのフェースプロパティーのキー.

変数 **Mhline** (p. 134) はフェースプロパティのキーとして用いられる。値は **MFaceHLineProp** (p. 173) 型オブジェクトへのポインタか `NULL` でなくてはならない。

値が `NULL` ならば、このプロパティは指定されない。 そうでなければ値が指すオブジェクトに指定されたように水平線を付加して M-text を表示する。

2.21.4.6 MSymbol Mbox

囲み枠を指定するためのフェースプロパティーのキー.

変数 **Mbox** (p. 134) はフェースプロパティのキーとして用いられる。値は **MFaceBoxProp** (p. 172) 型オブジェクトへのポインタか `NULL` でなくてはならない。

値が `NULL` ならば、このフェースは囲み枠を指定していない。 そうでなければ値が指すオブジェクトに指定されたように囲み枠を付加して M-text を表示する。

2.21.4.7 MSymbol Mfontset

フォントセットを指定するためのフェースプロパティーのキー.

変数 **Mfontset** (p. 134) はフェースプロパティのキーとして用いられる。値は **Mfontset** (p. 134) 型オブジェクトへのポインタか `NULL` でなくてはならない。

値が NULL ならば、フォントセットは指定されていない。 そうでなければ値が指すオブジェクトに指定されたフォントセットから選んだフォントで M-text を表示する。

2.21.4.8 MSymbol Mhook_func

フックを指定するためのフェースプロパティーのキー.

変数 **Mhook_func** (p. 135) はフェースプロパティのキーとして用いられる。値は **MFaceHookFunc** (p. 131) 型の関数か NULL でなくてはならない。

値が NULL ならば、フックは指定されていない。 そうでなければフェースを実現する前に指定された関数が呼ばれる。

2.21.4.9 MSymbol Mhook_arg

フックの引数を指定するためのフェースプロパティーのキー.

変数 **Mhook_arg** (p. 135) はフェースプロパティのキーとして用いられる。 値は何でもよく、フェースプロパティ **Mhook_func** (p. 135) で指定される関数に渡される。

2.21.4.10 MSymbol Mnormal

2.21.4.11 MSymbol Mreverse

2.21.4.12 MFace* mface_normal_video

標準ビデオフェース.

変数 **mface_normal_video** (p. 135) は **Mvideomode** (p. 134) プロパティの値が **Mnormal** であるフェースを指すポインタである。他のプロパティは指定されない。 このフェースで表示される M-text は標準の色 (すなわち前景は前景色、背景は背景色）で描かれる。

2.21.4.13 MFace* mface_reverse_video

リバースビデオフェース.

変数 **mface_reverse_video** (p. 135) は **Mvideomode** (p. 134) プロパティの値が **Mreverse** であるフェースを指すポインタである。他のプロパティは指定されない。 このフェースで表示される M-text は前景色と背景色が入れ替わって (すなわち前景は背景色、背景は前景色）描かれる。

2.21.4.14 MFace* mface_underline

下線フェース.

変数 **mface_underline** (p. 135) は **Mhline** (p. 134) プロパティの値が **MFaceHLineProp** (p. 173) 型オブジェクトへのポインタであるフェースを指すポインタである。オブジェクトのメンバは以下の通り。

```
メンバ  値
-----   -----
type    MFACE_HLINE_UNDER
width   1
color   Mnil
```

他のプロパティは指定されない。このフェースを持つ M-text は下線付きで表示される。

2.21.4.15 MFace* mface_medium

ミディアムフェース.

変数 **mface_medium** (p. 136) は **Mweight** (p. 122) プロパティの値が "medium" という名前をもつシンボルであるようなフェースを指すポインタである。 他のプロパティは指定されない。このフェースを持つ M-text は、ミディアムウェイトのフォントで表示される。

2.21.4.16 MFace* mface_bold

ボールドフェース.

変数 **mface_bold** (p. 136) は **Mweight** (p. 122) プロパティの値が "bold" という名前をもつシンボルであるようなフェースを指すポインタである。 他のプロパティは指定されない。このフェースを持つ M-text は、ボールドフォントで表示される。

2.21.4.17 MFace* mface_italic

イタリックフェース.

変数 **mface_italic** (p. 136) は **Mstyle** (p. 122) プロパティの値が "italic" という名前をもつシンボルであるようなフェースを指すポインタである。 他のプロパティは指定されない。このフェースを持つ M-text は、イタリック体で表示される。

2.21.4.18 MFace* mface_bold_italic

ボールドイタリックフェース.

変数 **mface_bold_italic** (p. 136) は、**Mweight** (p. 122) プロパティの値が "bold" という名前をもつシンボルであり、かつ **Mstyle** (p. 122) プロパティの値が "italic" という名前をもつシンボルであるようなフェースを指すポインタである。 他のプロパティは指定されない。このフェースを持つ M-text は、ボールドイタリック体で表示される。

2.21.4.19 MFace* mface_xx_small

最小のフェース.

変数 **mface_xx_small** (p. 136) は、**Mratio** (p. 134) プロパティの値が 50 であるフェースを指すポインタである。他のプロパティは指定されない。 このフェースを持つ M-text は標準フォントの 50% の大きさのフォントを用いて表示される。

2.21.4.20 MFace* mface_x_small

より小さいフェース.

変数 **mface_x_small** (p. 136) は、**Mratio** (p. 134) プロパティの値が 66 であるフェースを指すポインタである。他のプロパティは指定されない。 このフェースを持つ M-text は標準フォントの 66% の大きさのフォントを用いて表示される。

2.21.4.21 MFace* mface_small

小さいフェース.

変数 **mface_small** (p. 136) は、**Mratio** (p. 134) プロパティの値が 75 であるフェースを指すポインタである。他のプロパティは指定されない。 このフェースを持つ M-text は標準フォントの 75% の大きさのフォントを用いて表示される。

2.21.4.22 MFace* mface_normalsize

標準の大きさのフェース.

変数 **mface_normalsize** (p. 137) は、**Mratio** (p. 134) プロパティの値が 100 であるフェースを指すポインタである。他のプロパティは指定されない。 このフェースを持つ M-text は標準フォントと同じ大きさのフォントを用いて表示される。

2.21.4.23 MFace* mface_large

大きいフェース.

変数 **mface_large** (p. 137) は、**Mratio** (p. 134) プロパティの値が 120 であるフェースを指すポインタである。他のプロパティは指定されない。 このフェースを持つ M-text は標準フォントの 120% の大きさのフォントを用いて表示される。

2.21.4.24 MFace* mface_x_large

もっと大きいフェース.

変数 **mface_x_large** (p. 137) は、**Mratio** (p. 134) プロパティの値が 150 であるフェースを指すポインタである。他のプロパティは指定されない。 このフェースを持つ M-text は標準フォントの 150% の大きさのフォントを用いて表示される。

2.21.4.25 MFace* mface_xx_large

最大のフェース.

変数 **mface_xx_large** (p. 137) は、**Mratio** (p. 134) プロパティの値が 200 であるフェースを指すポインタである。他のプロパティは指定されない。 このフェースを持つ M-text は標準フォントの 200% の大きさのフォントを用いて表示される。

2.21.4.26 MFace* mface_black

黒フェース.

変数 **mface_black** (p. 137) は、**Mforeground** (p. 133) プロパティの値として "black" という名前のシンボルを持つようなフェースを指すポインタである。 他のプロパティは指定されない。このフェースを持つ M-text は前景色として黒を用いて表示される。

2.21.4.27 MFace* mface_white

白フェース.

変数 **mface_white** (p. 137) は、**Mforeground** (p. 133) プロパティの値として "white" という名前のシンボルを持つようなフェースを指すポインタである。 他のプロパティは指定されない。このフェースを持つ M-text は前景色として白を用いて表示される。

2.21.4.28 MFace* mface_red

赤フェース.

変数 **mface_red** (p. 138) は、**Mforeground** (p. 133) プロパティの値として "red" という名前のシンボルを持つようなフェースを指すポインタである。 他のプロパティは指定されない。このフェースを持つ M-text は前景色として赤を用いて表示される。

2.21.4.29 MFace* mface_green

緑フェース.

変数 **mface_green** (p. 138) は、**Mforeground** (p. 133) プロパティの値として "green" という名前のシンボルを持つようなフェースを指すポインタである。 他のプロパティは指定されない。このフェースを持つ M-text は前景色として緑を用いて表示される。

2.21.4.30 MFace* mface_blue

青フェース.

変数 **mface_blue** (p. 138) は、**Mforeground** (p. 133) プロパティの値として "blue" という名前のシンボルを持つようなフェースを指すポインタである。 他のプロパティは指定されない。このフェースを持つ M-text は前景色として青を用いて表示される。

2.21.4.31 MFace* mface_cyan

シアンフェース.

変数 **mface_cyan** (p. 138) は、**Mforeground** (p. 133) プロパティの値として "cyan" という名前のシンボルを持つようなフェースを指すポインタである。 他のプロパティは指定されない。このフェースを持つ M-text は前景色としてシアンを用いて表示される。

2.21.4.32 MFace* mface_yellow

黄フェース.

変数 **mface_yellow** (p. 138) は、**Mforeground** (p. 133) プロパティの値として "yellow" という名前のシンボルを持つようなフェースを指すポインタである。 他のプロパティは指定されない。このフェースを持つ M-text は前景色として黄色を用いて表示される。

2.21.4.33 MFace* mface_magenta

マゼンタフェース.

変数 **mface_magenta** (p. 138) は、**Mforeground** (p. 133) プロパティの値として "magenta" という名前のシンボルを持つようなフェースを指すポインタである。 他のプロパティは指定されない。このフェースを持つ M-text は前景色としてマゼンタを用いて表示される。

2.21.4.34 MSymbol Mface

フェースを指定するテキストプロパティのキー.

変数 **Mface** (p. 138) は `"face"` という名前を持つシンボルである。このシンボルをキーとするテキストプロパティは、 **MFace** (p. 131) 型のオブジェクトへのポインタを持たなければならない。 これは管理キーである。

2.22 表示

M-text をウィンドウに描画する.

データ構造

- struct **MDrawControl**
  テキスト表示制御の型宣言.
- struct **MDrawMetric**
  グリフとテキストの寸法の型宣言.
- struct **MDrawGlyphInfo**
  グリフに関する情報の型宣言.
- struct **MDrawGlyph**
  グリフの寸法とフォントに関する情報の型宣言.

型定義

- typedef void * **MDrawWindow**
  ウィンドウシステムに依存する、ウィンドウの型宣言.
- typedef void * **MDrawRegion**
  ウィンドウシステムに依存する、領域の型宣言.

関数

- int **mdraw_text** (**MFrame** *frame, **MDrawWindow** win, int x, int y, **MText** *mt, int from, int to)
  ウィンドウに *M-text* **を描画する**.
- int **mdraw_image_text** (**MFrame** *frame, **MDrawWindow** win, int x, int y, **MText** *mt, int from, int to)
  ディスプレイに *M-text* **を画像として描く**.
- int **mdraw_text_with_control** (**MFrame** *frame, **MDrawWindow** win, int x, int y, **MText** *mt, int from, int to, **MDrawControl** *control)
  ディスプレイに *M-text* **を詳細な制御つきで描く**.
- int **mdraw_text_extents** (**MFrame** *frame, **MText** *mt, int from, int to, **MDrawControl** *control, **MDrawMetric** *overall_ink_return, **MDrawMetric** *overall_logical_return, **MDrawMetric** *overall_line_return)
  テキストの幅（ピクセル単位）を計算する.
- int **mdraw_text_per_char_extents** (**MFrame** *frame, **MText** *mt, int from, int to, **MDrawControl** *control, **MDrawMetric** *ink_array_return, **MDrawMetric** *logical_array_return, int array_size, int *num_chars_return, **MDrawMetric** *overall_ink_return, **MDrawMetric** *overall_logical_return)
  M-text **の各文字の表示範囲を計算する**.

- int **mdraw_coordinates_position** (**MFrame** ∗frame, **MText** ∗mt, int from, int to, int x_offset, int y_offset, **MDrawControl** ∗control)

  指定した座標に最も近い文字の文字位置を得る.

- int **mdraw_glyph_info** (**MFrame** ∗frame, **MText** ∗mt, int from, int pos, **MDrawControl** ∗control, **MDrawGlyphInfo** ∗info)

  グリフに関する情報を計算する.

- int **mdraw_glyph_list** (**MFrame** ∗frame, **MText** ∗mt, int from, int to, **MDrawControl** ∗control, **MDrawGlyph** ∗glyphs, int array_size, int ∗num_glyphs_return)

  グリフ列に関する情報を計算する.

- void **mdraw_text_items** (**MFrame** ∗frame, **MDrawWindow** win, int x, int y, **MDrawTextItem** ∗items, int nitems)

  textitem **を表示する**.

- int **mdraw_default_line_break** (**MText** ∗mt, int pos, int from, int to, int line, int y)

  改行位置を計算する.

- void **mdraw_per_char_extents** (**MFrame** ∗frame, **MText** ∗mt, **MDrawMetric** ∗array_return, **MDrawMetric** ∗overall_return)

  M-text **の文字毎の表示範囲情報を得る**.

- void **mdraw_clear_cache** (**MText** ∗mt)

  キャッシュ情報を消す.

変数

- int **mdraw_line_break_option**

2.22.1 説明

M-text をウィンドウに描画する.

m17n-gui API には、M-text を表示するための関数が用意されている。

表示に用いられるフォントは、フォントセットと face のプロパティに基づいて自動的に決定される。また、フォントのサイズや色や下線などの見栄えも face によって決まる。

M-text の描画フォーマットは多様な方法で制御できるので、強力な二次元レイアウト機能が実現できる。

2.22.2 型定義

2.22.2.1 typedef void∗ MDrawWindow

ウィンドウシステムに依存する、ウィンドウの型宣言.

MDrawWindow (p. 140) はウィンドウ、すなわち幾つかの点でスクリーンのミニチュアとして働く矩形領域用の型である。

実際に何を指すかはウィンドウシステムに依存する。 m17n X ライブラリを利用するプログラムは `Drawable` 型をこの型に変換しなくてはならない。

2.22.2.2 typedef void* MDrawRegion

ウィンドウシステムに依存する、領域の型宣言.

MDrawRegion (p. 141) は領域、すなわちスクリーン上の任意のピクセルの集合（典型的には矩形領域）用の型である。

実際に何を指すかはウィンドウシステムに依存する。 m17n X ライブラリを利用するプログラムは `Region` 型をこの型に変換しなくてはならない。

2.22.3 関数

2.22.3.1 int mdraw_text (MFrame * *frame*, MDrawWindow *win*, int *x*, int *y*, MText * *mt*, int *from*, int *to*)

ウィンドウに M-text を描画する.

関数 **mdraw_text()** (p. 141) は、フレーム **frame** のウィンドウ **win** の座標 (**x**, **y**) に、M-text **mt** の **from** から **to** までのテキストを描画する。

テキストの見栄え（フォント、スタイル、色など）は、キーが `Mface` であるテキストプロパティの値によって決まる。M-text の一部あるいは全部にそのようなテキストプロパティが付いていない場合には、**frame** のデフォルトフェースを代わりに用いる。

M-text の各文字を表示するフォントは、フェースの fontset プロパティの値のうちから、以下のアルゴリズムで選ばれる。

1. その文字のテキストプロパティのうち、キーが `Mcharset` であるものの値を調べる。この値は文字セットを表わすシンボルか **Mnil** (p. 16) のどちらかである。**Mnil** (p. 16) ならば、次のステップに進む。 そうでなければ、fontset のマッピングテーブルにその文字セット用のフォントがあるかどうかを調べる。 無ければ、次のステップに進む。

   その文字セット用のフォントがみつかれば、それらのうち現在の文字用のグリフを持ち、フェースの各プロパティに最もよく合致するものを使う。 そのようなフォントが無ければ次のステップに進む。

2. その文字の文字プロパティ "script" （スクリプト）を調べる。 そのプロパティが継承されているならばそれ以前の文字の文字プロパティ "script" を調べる。前の文字がなかったり、その文字プロパティを持っていなかった場合には、次のステップに進む。

   その文字のテキストプロパティのうち、キーが `Mlanguage` であるものの値を調べる。 この値は言語を表わすシンボルか `Mnil` のいずれかである。

   その言語とスクリプトの組み合わせが fontset のマッピングテーブルにあるかどうかを調べる。見つからなければ次のステップに進む。

   見つかったばあいには、それらのフォントのうち現在の文字用のグリフを持ち、フェースの各プロパティに最もよく合致しているものを使う。 そのようなフォントが無ければ次のステップに進む。

3. その文字のグリフを持つフォントを、フォントセットの fall-back テーブルから探す。フォントが見つかればそれを使う。

以上のアルゴリズムでフォントが見つからなければ、この関数はその文字として空の四角形を表示する。

この関数が描画するのはグリフの前景だけである。背景色を指定するには、関数 **mdraw_image_text()** (p. 142) か関数 **mdraw_text_with_control()** (p. 142) を使うこと。

この関数は、X ウィンドウにおける関数 `XDrawString()`, `XmbDrawString()`, `XwcDrawString()` に相当する。

戻り値:

処理が成功した場合、mdraw_text() は 0 返す。エラーが検出された場合は -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_RANGE

参照:
mdraw_image_text() (p. 142)

2.22.3.2 int mdraw_image_text (MFrame * *frame*, MDrawWindow *win*, int *x*, int *y*, MText * *mt*, int *from*, int *to*)

ディスプレイに M-text を画像として描く.

関数 **mdraw_image_text()** (p. 142) は、フレーム **frame** のウィンドウ **win** の座標 (**x**, **y**) に、M-text **mt** の **from** から **to** までのテキストを画像として描く。

テキストの描画方法は **mdraw_text()** (p. 141) とほぼ同じであるが、この関数ではフェースで指定された色で背景も描く点が異なっている。

この関数は、X ウィンドウにおける XDrawImageString(), XmbDrawImageString(), XwcDrawImageString() に相当する。

戻り値:
処理が成功した場合、mdraw_image_text() は 0 を返す。エラーが検出された場合は -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:
MERROR_RANGE

参照:
mdraw_text() (p. 141)

2.22.3.3 int mdraw_text_with_control (MFrame * *frame*, MDrawWindow *win*, int *x*, int *y*, MText * *mt*, int *from*, int *to*, MDrawControl * *control*)

ディスプレイに M-text を詳細な制御つきで描く.

関数 **mdraw_text_with_control()** (p. 142) は、フレーム **frame** のウィンドウ **win** の座標 (**x**, **y**) に、M-text **mt** の **from** から **to** までのテキストを描く。

テキストの描画方法は **mdraw_text()** (p. 141) とほぼ同じであるが、この関数は描画制御用のオブジェクト **control** の指示にも従う点が異なっている。

たとえば **control** の <two_dimensional> がゼロでなければ、この関数は M-text を 2 次元的に描く。すなわち M-text 中の改行で行を改め、続く文字は次の行に描く。詳細は構造体 @ **MDrawControl** (p. 162) の説明を参照すること。

2.22.3.4 int mdraw_text_extents (MFrame * *frame*, MText * *mt*, int *from*, int *to*, MDrawControl * *control*, MDrawMetric * *overall_ink_return*, MDrawMetric * *overall_logical_return*, MDrawMetric * *overall_line_return*)

テキストの幅（ピクセル単位）を計算する.

関数 **mdraw_text_extents()** (p. 142) は、関数 **mdraw_text_with_control()** (p. 142) が描画制御オブジェクト **control** を用いて M-text **mt** の **from** から **to** までをフレーム **frame** に表示する際に必要となる幅を返す。

overall_ink_return が NULL でなければ、この関数は M-text の文字のインクのバウンディングボックスも計算し、**overall_ink_return** が指す構造体のメンバにその結果を設定する。M-text に囲み枠 (surrounding box) を指定するフェースがあれば、それもバウンディングボックスに含む。

overall_logical_return が NULL でなければ、この関数は M-text と他の graphical feature（囲み枠など）との間の最小のスペースを示すバウンディングボックスも計算し、**overall_logical_return** が指す構造体のメンバにその結果を設定する。

overall_line_return が NULL でなければ、この関数は他の M-text との間の最小のスペースを示すバウンディングボックスも計算し、 **overall_line_return** が指す構造体のメンバにその結果を設定する。オブジェクト **control** のメンバ min_line_ascent, min_line_descent, max_line_ascent, max_line_descent がすべて 0 の時には、この値は **overall_ink_return** と **overall_logical_return** の和となる。

戻り値:

この関数は表示に必要なテキストの幅をピクセル単位で返す。**control->two_dimensional** が 0 でなく、テキストが複数の行に渡って描かれる場合には、最大の幅を返す。エラーが生じた場合は -1 を返し、外部変数 **merror_code** (p. 152) にエラーコードを設定する。

エラー:

MERROR_RANGE

2.22.3.5 int mdraw_text_per_char_extents (MFrame * *frame*, MText * *mt*, int *from*, int *to*, MDrawControl * *control*, MDrawMetric * *ink_array_return*, MDrawMetric * *logical_array_return*, int *array_size*, int * *num_chars_return*, MDrawMetric * *overall_ink_return*, MDrawMetric * *overall_logical_return*)

M-text の各文字の表示範囲を計算する.

関数 **mdraw_text_per_char_extents()** (p. 143) は、関数 **mdraw_text_with_control()** (p. 142) が描画制御オブジェクト **control** を用いて M-text **mt** の **from** から **to** までをフレーム **frame** に表示する際の各文字のサイズを計算する。

array_size によって **ink_array_return** と **logical_array_return** のサイズを指定する。**ink_array_return** と **logical_array_return** の各要素は、それぞれ文字の描画インクと論理サイズ（M-text の表示原点からの相対位値）によって順に埋められる。設定された **ink_array_return** と **logical_array_return** の要素の数は、**num_chars_return** に戻される。

array_size がすべての寸法を戻せないほど小さい場合には、関数は -1 を返し、必要な大きさを **num_chars_return** に返す。そうでなければ 0 を返す。

ポインタ **overall_ink_return** と **overall_logical_return** が NULL でなければ、この関数はテキスト全体のサイズも計算し、結果を **overall_ink_return** と **overall_logical_return** で指される構造のメンバに保存する。

control->two_dimensional が 0 でなければ、この関数は最初の行の文字のサイズだけを計算する。

2.22.3.6 int mdraw_coordinates_position (MFrame * *frame*, MText * *mt*, int *from*, int *to*, int *x_offset*, int *y_offset*, MDrawControl * *control*)

指定した座標に最も近い文字の文字位置を得る.

関数 **mdraw_coordinates_position()** (p. 143) は、関数 **mdraw_text_with_control()** (p. 142) が描画制御オブジェクト **control** を用いて、 M-text **mt** の **from** から **to** までを座標 (0, 0) を起点として描画する際に、座標 (**x**, **y**) に描画される文字の文字位置を返す。ここで文字位置とは、当該 M-text 中においてその文字が最初から何番目かを示す整数である。ただし最初の文字の文字位置は 0 とする。

frame はデフォルトのフェースの情報を得るためだけに用いられる。

戻り値:

座標 (**x**, **y**) がある文字のグリフで覆われる場合、 関数 **mdraw_coordinates_position()** (p. 143) はその文字の文字位置を返す。
もし **y** が描画領域の最小 Y 座標よりも小さいならば **from** を返す。

もし **y** が描画領域の最大 Y 座標よりも大きいならば **to** を返す。
もし **y** が描画領域に乗っていてかつ **x** が描画領域の最小 X 座標よりも 小さい場合は、直線 y = **y** 上に描画される最初の文字の文字位置を返す。
もし **y** が描画領域に乗っていてかつ **x** が描画領域の最大 X 座標よりも 大きい場合は、直線 y = **y** 上に描画される最後の文字の文字位置を返す。

2.22.3.7 int mdraw_glyph_info (MFrame * *frame*, MText * *mt*, int *from*, int *pos*, MDrawControl * *control*, MDrawGlyphInfo * *info*)

グリフに関する情報を計算する.

関数 **mdraw_glyph_info()** (p. 144) は、関数 **mdraw_text_with_control()** (p. 142) が描 画制御オブジェクト **control** を用いて M-text **mt** の **from** から **to** までをフレーム **frame** に描画した場合、M-text の文字位置 **pos** の文字を覆うグリフに関する情報を計算する。

情報は **info** のメンバに保持される。

参照:
MDrawGlyphInfo (p. 168)

2.22.3.8 int mdraw_glyph_list (MFrame * *frame*, MText * *mt*, int *from*, int *to*, MDrawControl * *control*, MDrawGlyph * *glyphs*, int *array_size*, int * *num_glyphs_return*)

グリフ列に関する情報を計算する.

関数 **mdraw_glyph_list()** (p. 144) は、関数 **mdraw_text_with_control()** (p. 142) が描画制御オブジェクト **control** を用いて M-text **mt** の **from** から **to** までをフレーム **frame** に描画した場合の、各グリフの情報を **glyphs** が指す配列に格納する。 **array_size** はその配列のサイズである。

もし **array_size** がすべてのグリフについての情報を格納するのに十分であれば、 **num_glyphs_return** が指す場所に実際に埋めた要素の数を設定し 0 を返す。

そうでなければ、**num_glyphs_return** が指す場所に必要な配列のサイズを設定し、 -1 を返す。

参照:
MDrawGlyph (p. 166)

2.22.3.9 void mdraw_text_items (MFrame * *frame*, MDrawWindow *win*, int *x*, int *y*, MDrawTextItem * *items*, int *nitems*)

textitem を表示する.

関数 **mdraw_text_items()** (p. 144) は、一個以上のテキストアイテムを、フレーム **frame** のウィンドウ **win** の座標 (**x**, **y**) に表示する。**items** は表示すべきテキストアイテムの配列であり、**nitems** はその個数である。

参照:
MTextItem, **mdraw_text()** (p. 141).

2.22.3.10 int mdraw_default_line_break (MText * *mt*, int *pos*, int *from*, int *to*, int *line*, int *y*)

改行位置を計算する.

関数 **mdraw_default_line_break()** (p. 144) は、行が最大幅中に収まらない場合の改行位置を、行番号 **line** と座標 **y** に基づいて計算する。 **pos** は最大幅に収まる最後の文字の次の文字の位置である。 **from** はその

行の最初の文字の位置、**to** は最大幅が指定されていなければその行に表示される最後の文字の位置である。 **line** と **y** は改行文字によって行が改まった際には 0 にリセットされ、最大幅によって行が改まった場合には 1 づつ増やされる。

戻り値:
この関数は改行する文字位置を返す。

2.22.3.11 void mdraw_per_char_extents (MFrame * *frame*, MText * *mt*, MDrawMetric * *array_return*, MDrawMetric * *overall_return*)

M-text の文字毎の表示範囲情報を得る.

関数 **mdraw_per_char_extents()** (p. 145) は、M-text **mt** 中の各文字の表示範囲を計算する。この計算に用いるフォントは、 **mt** のテキストプロパティで指定されたフェースと、フレーム **frame** のデフォルトフェースによって決まる。**array_return** の各要素は、**mt** 中の各文字の表示範囲情報によって順に埋められる。表示範囲情報とは、 表示原点からの相対位置と各文字の占める長方形である。**array_return** の要素数は、M-text 中の文字数以上でなければならない。

ポインタ **overall_return** が NULL でない場合は、テキスト全体の表示範囲情報も計算し、その結果を **overall_return** の指す構造体に格納する。

2.22.3.12 void mdraw_clear_cache (MText * *mt*)

キャッシュ情報を消す.

関数 **mdraw_clear_cache()** (p. 145) は描画関数によって M-text **mt** に付加されたキャッシュ情報をすべて消去する。MDrawControl の 'format' あるいは 'line_break' メンバ関数の振舞いが変わった場合にはキャッシュを消去しなくてはならない。

参照:
MDrawControl (p. 162)

2.22.4 変数

2.22.4.1 int mdraw_line_break_option

2.23 入力メソッド (GUI)

ウィンドウシステム上の入力メソッドのサポート.

データ構造

- struct **MInputGUIArgIC**

  関数 *minput_create_ic()* (p. *96*) の引数の型宣言.

- struct **MInputXIMArgIM**

  関数 *minput_open_im()* (p. *95*) の引数 arg によって指される構造体.

- struct **MInputXIMArgIC**

  関数 *minput_create_ic()* (p. *96*) の引数 arg によって指される構造体.

関数

- **MSymbol minput_event_to_key** (**MFrame** ∗frame, void ∗event)

  イベントを入力キーに変換する.

変数

- **MInputDriver minput_gui_driver**

  ウィンドウシステムの内部入力メソッド用入力ドライバ.

- **MSymbol Mxim**

  *"xim"***を名前として持つシンボル** .

- **MInputDriver minput_xim_driver**

  XIM **用入力ドライバ**.

2.23.1 説明

ウィンドウシステム上の入力メソッドのサポート.

入力ドライバ `minput_gui_driver` は、 ウィンドウシステム上で用いられる内部入力メソッド用のドライバである。 このドライバは入力スポットに preedit テキストと status テキストを表示する。詳細については `minput_gui_driver` の説明を参照のこと。

m17n-X ライブラリは、`Mxim` と言う名前を持つ外部入力メソッドを提供している。これは XIM (X Input Method) をバックグラウンドの入力エンジンとして利用する。シンボル `Mxim` は `Minput_driver` というプロパティを持っており、その値は入力ドライバ `minput_xim_driver` へのポインタである。 詳細については `minput_xim_driver` の説明を参照のこと。

2.23.2 関数

2.23.2.1 MSymbol minput_event_to_key (MFrame * *frame*, void * *event*)

イベントを入力キーに変換する.

関数 **minput_event_to_key()** (p. 147) は、**frame** のイベント **event** に対応する入力キーを返す。ここでの「対応」はウィンドウシステムに依存する。

m17n-X ライブラリの場合には、**event** は 構造体 `XKeyEvent` へのポインタであり、次のように処理される。

まず、関数 `XKeysymToString` によって、**event** の keysym 名を取得し、次いで以下の変更を加える。

名前が "a" .. "z" のいずれかであって **event** に Shift モディファイアがあれば、名前はそれぞれ "A" .. "Z" に変換され、Shift モディファイアは取り除かれる。

名前が１バイト長で **event** に Control モディファイアがあれば、名前と 0x1F とをビット単位で and 演算し、Control モディファイアは取り除かれる。

それでも **event** にまだモディファイアがあれば、名前の前にそれぞれ "S-" (Shift), "C-" (Control), "M-" (Meta), "A-" (Alt), "s-" (Super), "H-" (Hyper) がこの順番で付く。

たとえば、keysym 名が "a" でイベントが Shift, Meta, and Hyper モディファイアを持てば、得られる名前は "M-H-A" である。

最後にその名前を持つシンボルを返す。

2.23.3 変数

2.23.3.1 MInputDriver minput_gui_driver

ウィンドウシステムの内部入力メソッド用入力ドライバ.

入力ドライバ `minput_gui_driver` は、ウィンドウシステム上で用いられる入力メソッド用ドライバである。

このドライバは、関数 **minput_set_spot()** (p. 97) によって設定された入力スポットに preedit テキスト用のサブウィンドウと status テキスト用のサブウィンドウを作り、それぞれを表示する。

マクロ **M17N_INIT()** (p. 7) は変数 `minput_driver` をこのドライバへのポインタに設定し、全ての内部入力メソッドがこのドライバを使うようにする。

したがって、`minput_driver` がデフォルト値のままであれば、minput_ で始まる名前を持つ関数の引数のうちドライバ依存のものは以下のようになる。

関数 **minput_open_im()** (p. 95) の引数 **arg** は無視される。

関数 **minput_create_ic()** (p. 96) の引数 **arg** は構造体 **`MInputGUIArgIC`** （p. 188）へのポインタでなくてはならない。詳細については **`MInputGUIArgIC`** （p. 188）の説明を参照のこと。

関数 **minput_filter()** (p. 96) の引数 **arg** が `Mnil` の場合、 **arg** は `XEvent` 型のオブジェクトへのポインタでなくてはならない。この場合 **key** は **arg** から生成される。

関数 **minput_lookup()** (p. 96) の引数 **arg** は関数 **minput_filter()** (p. 96) の引数 **arg** と同じでなくてはならない。

2.23.3.2 MSymbol Mxim

"xim"を名前として持つシンボル .

変数 Mxim は"xim"を名前として持つシンボルである。"xim" は入力メソッドドライバ **minput_xim_driver** (p. 148) の名前である。

2.23.3.3 MInputDriver minput_xim_driver

初期値:

```
{ xim_open_im, xim_close_im, xim_create_ic, xim_destroy_ic,
  xim_filter, xim_lookup, NULL }
```

XIM 用入力ドライバ.

ドライバ **minput_xim_driver** (p. 148) は **Mxim** (p. 147) を名前として持つ外部入力メソッド用であり、XIM (X Input Methods) をバックグラウンドの入力エンジンとして使用する。

シンボル **Mxim** (p. 147) はこのドライバへのポインタを値とするプロパティ **Minput_driver** (p. 106) を持ち、LANGUAGE が **Mnil** (p. 16) で名前が **Mxim** (p. 147) である入力メソッドはこのドライバを利用する。

したがって、それらの入力メソッドでは、minput_ で始まる名前を持つ関数のドライバに依存する引数は次のようなものでなくてはならない。

関数 **minput_open_im()** (p. 95) の引数 **arg** は構造体 **MInputXIMArgIM** (p. 191) へのポインタでなくてはならない。詳細については **MInputXIMArgIM** (p. 191) の説明を参照。

関数 **minput_create_ic()** (p. 96) の引数 **arg** は構造体 **MInputXIMArgIC** (p. 190) へのポインタでなくてはならない。詳細については **MInputXIMArgIC** (p. 190) の説明を参照。

関数 **minput_filter()** (p. 96) の引数 **arg** は構造体 `XEvent` へのポインタでなくてはならない。引数 **key** は無視される。

関数 **minput_lookup()** (p. 96) の引数 **arg** は関数 function **minput_filter()** (p. 96) の引数 **arg** と同じものでなくてはならない。 引数 **key** は、無視される。

2.24 MISC API

その他の API

モジュール

- **エラー処理**
  m17n **ライブラリのエラー処理**.
- **デバッグサポート**
  m17n **ライブラリユーザのためのプログラムデバッグサポート**.

2.24.1 説明

その他の API

2.25 エラー処理

m17n ライブラリのエラー処理.

列挙型

- enum **MErrorCode** {
  MERROR_NONE,
  MERROR_OBJECT,
  MERROR_SYMBOL,
  MERROR_MTEXT,
  MERROR_TEXTPROP,
  MERROR_CHAR,
  MERROR_CHARTABLE,
  MERROR_CHARSET,
  MERROR_CODING,
  MERROR_RANGE,
  MERROR_LANGUAGE,
  MERROR_LOCALE,
  MERROR_PLIST,
  MERROR_MISC,
  MERROR_WIN,
  MERROR_X,
  MERROR_FRAME,
  MERROR_FACE,
  MERROR_DRAW,
  MERROR_FLT,
  MERROR_FONT,
  MERROR_FONTSET,
  MERROR_FONT_OTF,
  MERROR_FONT_X,
  MERROR_FONT_FT,
  MERROR_IM,
  MERROR_DB,
  MERROR_IO,
  MERROR_DEBUG,
  MERROR_MEMORY,
  MERROR_GD,
  MERROR_MAX }
  m17n **ライブラリエラーコードの列挙**.

変数

- int **merror_code**

  m17n **ライブラリのエラーコードを保持する外部変数**.

- void(∗ **m17n_memory_full_handler**)(enum **MErrorCode** err)

  メモリ割当てエラーハンドラ.

2.25.1 説明

m17n ライブラリのエラー処理.

m17n ライブラリの関数では、２つの種類のエラーが起こり得る。

一つは引数のエラーである。 ライブラリの関数が妥当でない引数とともに呼ばれた場合、その関数はエラーを意味する値を返し、同時に外部変数 **merror_code** (p. 152) にゼロでない整数をセットする。

もう一つの種類はメモリ割当てエラーである。 システムが必要な量のメモリを割当てることができない場合、ライブラリ関数は外部変数 `m17n_memory_full_handler` が指す関数を呼ぶ。デフォルトでは、関数 default_error_handle() を指しており、この関数は単に `exit ()` を呼ぶ。

2.25.2 列挙型

2.25.2.1 enum MErrorCode

m17n ライブラリエラーコードの列挙.

m17n ライブラリエラーコードの列挙

ライブラリの関数が妥当でない引数とともに呼ばれた場合には、変数 **merror_code** (p. 152) をこれらの値のどれかにセットする。すべてのエラーコードは正の整数である。

メモリ割当てエラーの際には、外部変数 **m17n_memory_full_handler** (p. 152) の指す関数が、これらの値のうちのどれかを引数として呼ばれる。

列挙型の値:

MERROR_NONE

MERROR_OBJECT

MERROR_SYMBOL

MERROR_MTEXT

MERROR_TEXTPROP

MERROR_CHAR

MERROR_CHARTABLE

MERROR_CHARSET

MERROR_CODING

MERROR_RANGE

MERROR_LANGUAGE

MERROR_LOCALE

MERROR_PLIST

MERROR_MISC

MERROR_WIN

MERROR_X
MERROR_FRAME
MERROR_FACE
MERROR_DRAW
MERROR_FLT
MERROR_FONT
MERROR_FONTSET
MERROR_FONT_OTF
MERROR_FONT_X
MERROR_FONT_FT
MERROR_IM
MERROR_DB
MERROR_IO
MERROR_DEBUG
MERROR_MEMORY
MERROR_GD
MERROR_MAX

2.25.3 変数

2.25.3.1 int merror_code

m17n ライブラリのエラーコードを保持する外部変数.

外部変数 **merror_code** (p. 152) は、m17n ライブラリのエラーコードを保持する。 ライブラリ関数が妥当でない引数とともに呼ばれた際には、この変数を `enum` **MErrorCode** (p. 151) の一つにセットする。

この変数の初期値は 0 である。

2.25.3.2 void(∗ m17n_memory_full_handler)(enum MErrorCode err)

メモリ割当てエラーハンドラ.

変数 **m17n_memory_full_handler** (p. 152) は、ライブラリ関数がメモリ割当てに失敗した際に呼ぶべき関数へのポインタである。 **err** は `enum` **MErrorCode** (p. 151) のうちのいずれかであり、どのライブラリ関数でエラーが起ったかを示す。

初期設定では、この変数は単に `exit()` を **err** を引数として呼ぶ関数を指している。

これとは異なるエラー処理を必要とするアプリケーションは、この変数を適当な関数に設定することで、目的を達成できる。

2.26 デバッグサポート

m17n ライブラリユーザのためのプログラムデバッグサポート.

関数

- **MCharTable** ∗ **mdebug_dump_chartab** (**MCharTable** ∗table, int indent)
  文字テーブルをダンプする.
- **MFace** ∗ **mdebug_dump_face** (**MFace** ∗face, int indent)
  フェースをダンプする.
- **MFont** ∗ **mdebug_dump_font** (**MFont** ∗font)
  フォントをダンプする.
- **MFontset** ∗ **mdebug_dump_fontset** (**MFontset** ∗fontset, int indent)
  フォントセットをダンプする.
- **MInputMethod** ∗ **mdebug_dump_im** (**MInputMethod** ∗im, int indent)
  入力メソッドをダンプする.
- int **mdebug_hook** ()
  エラーの際に呼ばれるフック関数.
- **MText** ∗ **mdebug_dump_mtext** (**MText** ∗mt, int indent, int fullp)
  M-text **をダンプする**.
- **MPlist** ∗ **mdebug_dump_plist** (**MPlist** ∗plist, int indent)
  プロパティリストをダンプする.
- **MSymbol mdebug_dump_symbol** (**MSymbol** symbol, int indent)
  シンボルをダンプする.
- **MSymbol mdebug_dump_all_symbols** (int indent)
  すべてのシンボル名をダンプする.

2.26.1 説明

m17n ライブラリユーザのためのプログラムデバッグサポート.

m17n ライブラリは、そのユーザが自分のプログラムをデバッグするために、以下の機能をサポートしている。

- さまざまな情報のプリントを制御する環境変数。
  - MDEBUG_INIT – 1 ならば、M17N_INIT() が呼ばれた時点で、ライブラリの初期化に関する情報をプリントする。
  - MDEBUG_FINI – 1 ならば、M17N_FINI() が呼ばれた時点で、まだ解放されていないオブジェクトの参照数をプリントする。
  - MDEBUG_CHARSET – 1 ならば、m17n データベースからロードされた文字セットについての情報をプリントする。

  - MDEBUG_CODING – 1 ならば、m17n データベースからロードされたコード系についての情報をプリントする。
  - MDEBUG_DATABASE – 1 ならば、m17n データベースからロードされたデータについての情報をプリントする。
  - MDEBUG_FONT – 1 ならば、選択されてオープンされたフォントにつ いての情報をプリントする。
  - MDEBUG_FLT – 1、2、もしくは 3 ならば、Font Layout Table のど のコマンドが実行中かについてのをプリントする。より大きな値程より詳 しい情報をプリントする。
  - MDEBUG_INPUT – 1 ならば、実行中の入力メソッドの状態に付いての 情報をプリントする。
  - MDEBUG_ALL – 1 ならば、上記すべての変数を 1 にしたのと同じ効果を持つ。

- 種々のオブジェクトを人間に可読な形でプリントする関数。詳細は関数 mdebug_dump_XXXX() の説明参照。

- エラー発生時に呼ばれるフック関数。mdebug_hook() の説明参照。

2.26.2 関数

2.26.2.1 MCharTable* mdebug_dump_chartab (MCharTable * *table*, int *indent*)

文字テーブルをダンプする.

関数 **mdebug_dump_chartab()** (p. 154) は文字テーブル **table** を stderr に人間に可読な形で印刷する。 **indent** は 2 行目以降のインデントを指定する。

戻り値:
この関数は **table** を返す。

2.26.2.2 MFace* mdebug_dump_face (MFace * *face*, int *indent*)

フェースをダンプする.

関数 **mdebug_dump_face()** (p. 154) はフェース **face** を stderr に人間に可読な形で印刷する。 **indent** は 2 行目以降のインデントを指定する。

戻り値:
この関数は **face** を返す。

2.26.2.3 MFont* mdebug_dump_font (MFont * *font*)

フォントをダンプする.

関数 **mdebug_dump_font()** (p. 154) はフォント **font** を stderr に人間に可読な形で印刷する。

戻り値:
この関数は **font** を返す。

2.26.2.4 MFontset* mdebug_dump_fontset (MFontset * *fontset*, int *indent*)

フォントセットをダンプする.

関数 **mdebug_dump_face()** (p. 154) はフォントセット **fontset** を stderr に人間に可読な形で印刷する。**indent** は２行目以降のインデントを指定する。

戻り値:
この関数は **fontset** を返す。

2.26.2.5 MInputMethod* mdebug_dump_im (MInputMethod * *im*, int *indent*)

入力メソッドをダンプする.

関数 **mdebug_dump_im()** (p. 155) は入力メソッド **im** を stderr に人間に可読な形で印刷する。**indent** は２行目以降のインデントを指定する。

戻り値:
この関数は **im** を返す。

2.26.2.6 int mdebug_hook (void)

エラーの際に呼ばれるフック関数.

関数 **mdebug_hook()** (p. 155) はエラーが起こった際に呼ばれ、何もせずに-1 を返す。デバッガ内でブレークポイントを設定するために用いることができる。

2.26.2.7 MText* mdebug_dump_mtext (MText * *mt*, int *indent*, int *fullp*)

M-text をダンプする.

関数 **mdebug_dump_mtext()** (p. 155) は M-text **mt** を stderr に人間に可読な形で印刷する。 **indent** は２行目以降のインデントを指定する。 **fullp** が 0 ならば、文字コード列だけを印刷する。 そうでなければ、内部バイト列とテキストプロパティも印刷する。

戻り値:
この関数は **mt** を返す。

2.26.2.8 MPlist* mdebug_dump_plist (MPlist * *plist*, int *indent*)

プロパティリストをダンプする.

関数 **mdebug_dump_plist()** (p. 155) はプロパティリスト **plist** を stderr に人間に可読な形で印刷する。**indent** は２行目以降のインデントを指定する。

戻り値:
この関数は **plist** を返す。

2.26.2.9 MSymbol mdebug_dump_symbol (MSymbol *symbol*, int *indent*)

シンボルをダンプする.

関数 **mdebug_dump_symbol()** (p. 156) はシンボル $symbol を stderr に人間に可読な形で印刷する。 **indent** は 2 行目以降のインデントを指定する。

戻り値:
この関数は **symbol** を返す。

エラー:
MERROR_DEBUG

2.26.2.10 MSymbol mdebug_dump_all_symbols (int *indent*)

すべてのシンボル名をダンプする.

関数 **mdebug_dump_all_symbols()** (p. 156) は、すべてのシンボルの名前を stderr に印刷する。 **indent** は 2 行目以降のインデントを指定する。

戻り値:
この関数は **Mnil** (p. 16) を返す。

エラー:
MERROR_DEBUG

Chapter 3

データ構造

3.1 構造体 M17NObjectHead

管理下オブジェクトの最初のメンバ.

変数

- void * **filler** [2]

3.1.1 説明

管理下オブジェクトの最初のメンバ.

アプリケーションプログラムが新しい構造体を管理下オブジェクトとして定義する際には、最初のメンバは **M17NObjectHead** （p. 157）構造体型でなくてはならない。 **M17NObjectHead** （p. 157）の内容は m17n ライブラリが使用するので、アプリケーションプログラムは触れてはならない。

3.1.2 構造体

3.1.2.1 void* M17NObjectHead::filler[2]

Hidden from applications.

3.2 構造体 MCodingInfoISO2022

MCODING_TYPE_ISO_2022 (p. 76) タイプのコード系で必要な付加情報用構造体.

変数

- int **initial_invocation** [2]
- char **designations** [32]
- unsigned **flags**

3.2.1 説明

MCODING_TYPE_ISO_2022 (p. 76) タイプのコード系で必要な付加情報用構造体.

MCODING_TYPE_ISO_2022 タイプのコード系で必要な付加情報用を保持するための構造体。

3.2.2 構造体

3.2.2.1 int MCodingInfoISO2022::initial_invocation[2]

各図形文字領域 (Graphic Left と Graphic Right) に呼び出されている、 ISO2022 符合拡張要素の番号のテーブル。-1 はその領域にどの符合拡張要素も呼び出されていないことを示す。

3.2.2.2 char MCodingInfoISO2022::designations[32]

符合拡張要素のテーブル。N 番目の要素は、**charset_names** の N 番目の文字セットに対応する。
charset_names は関数 **mconv_define_coding()** (p. 76) の引数となる。

値が 0..3 だったら、対応する文字セットを G0..G3 のそれぞれに指示すること、また初期状態ですでに G0..G3 に指示されていることを意味する。

値が -4..-1 だったら、対応する文字セットを G0..G3 のそれぞれに指示すること、しかし初期状態ではどこにも指示されていないことを意味する。

3.2.2.3 unsigned MCodingInfoISO2022::flags

`enum MCodingFlagISO2022` のビット単位での論理 OR

3.3 構造体 MCodingInfoUTF

MCODING_TYPE_UTF (p. 75) タイプのコード系で必要な付加情報用の構造体.

変数

- int **code_unit_bits**
- int **bom**
- int **endian**

3.3.1 説明

MCODING_TYPE_UTF (p. 75) タイプのコード系で必要な付加情報用の構造体.

3.3.2 構造体

3.3.2.1 int MCodingInfoUTF::code_unit_bits

コード長（ビット数）の指定。値は 8, 16, 32 のいずれか。

3.3.2.2 int MCodingInfoUTF::bom

先頭の BOM (バイトオーダーマーク) の取り扱いを指定する。値は 0, 1, 2 のいずれかであり、それぞれの意味は以下のようになる。

0: デコードの際に最初の 2 バイトを調べる。もしそれが BOM であれば、エンディアンをそれで判定する。そうでなければ、メンバ `endian` に従ってエンディアンを決定する。エンコードの際には `endian` に従ったバイト列を先頭に BOM 付で生成する。

1: デコードの際、最初の 2 バイトを BOM として扱わず、エンディアンは `endian` で判定する。エンコードの際には、BOM を出力せず、 `endian` に応じたバイト列を生成する。

2: デコードの際に最初の 2 バイトを BOM として扱い、それに従ってエンディアンを判定する。 エンコードの際には `endian` に応じたバイト列を先頭に BOM 付きで生成する。

3.3.2.3 int MCodingInfoUTF::endian

エンディアンのタイプを指定する。値は 0 か 1 であり、0 ならばリトルエンディアン、1 ならばビッグエンディアンである。

<code_unit_bits> が 8 の場合には、この値は意味を持たない。

3.4 構造体 MConverter

コード変換に用いられる構造体.

変数

- int **lenient**
- int **last_block**
- unsigned **at_most**
- int **nchars**
- int **nbytes**
- enum **MConversionResult result**
- union {
  void * **ptr**
  double **dbl**
  char **c** [256]
  } **status**

- void * **internal_info**

3.4.1 説明

コード変換に用いられる構造体.

コード変換に用いられる構造体。最初の３つのメンバは変換を制御する。

3.4.2 構造体

3.4.2.1 int MConverter::lenient

厳密な変換が必要でない場合に値を 0 以外にする。 デフォルトでは、変換は厳密である。

変換が厳密とは、デコードの際には最初の不正なバイトでコンバータが止まること、 エンコードの際には変換されるコード系でサポートされない最初の文字でコンバータが止まることを指す。 これらの場合、 `MConverter->result` はそれぞれ `MCONVERSION_RESULT_INVALID_BYTE` か `MCONVERSION_RESULT_INVALID_CHAR` となる。

変換が厳密でない場合には、デコードの際の不正なバイトはそのバイトのまま残る。 またエンコードの際には、不正な文字が Unicode 文字の場合には "<U+XXXX>" に、そうでない場合には "<M+XXXXXX>" に置き換えられる。

3.4.2.2 int MConverter::last_block

バイト列の終端のブロックをデコードする際、または文字列の終端のブロックをエンコードする際は、値を 0 以外にする。この値は以下のように変換に影響する。

デコーディングの際に最後の数バイトが正しいバイトシークエンスを形成するには短すぎる場合：

値が 0 でなければ、変換はそのシークエンスの最初のバイトにおいて、エラー (MCONVERSION_RESULT_INVALID_BYTE) で終る。

値が 0 ならば、変換は成功して終る。 問題の数バイトはキャリーオーバとしてコンバータに保持され、変換の続きを行う際に変換するバイト列の前に付けられる。

エンコーディングの際にコード系が文脈依存の場合、

値が 0 でなければ、コンテキストを最初に戻すためのバイト列がソースの文字とかかわりなく変換の結果生成されることがある。

値が 0 ならば、そのようなバイト列は生成されない。

3.4.2.3 unsigned MConverter::at_most

0 でなければ、変換される最大の文字数を指定する。

3.4.2.4 int MConverter::nchars

以下の３つのメンバは変換の結果を表すためのものである。

最近にデコード/エンコードされた文字数。

3.4.2.5 int MConverter::nbytes

最近にデコード/エンコードされたバイト数。

3.4.2.6 enum MConversionResult MConverter::result

コード変換の結果を示すコード。

3.4.2.7 void* MConverter::ptr

3.4.2.8 double MConverter::dbl

3.4.2.9 char MConverter::c[256]

3.4.2.10 union { ... } MConverter::status

コード変換の状況に関する種々の情報。内容はコード系のタイプによって異なる。 `status` はどのような型へのキャストに対しても安全なようにメモリアラインされており、また最低 256 バイトのメモリ領域が使えるようになっている。

3.4.2.11 void* MConverter::internal_info

このメンバは内部的に使用され、アプリケーションプログラムは触れてはならない。

3.5 構造体 MDrawControl

テキスト表示制御の型宣言.

変数

- unsigned **as_image**: 1
- unsigned **align_head**: 1
- unsigned **two_dimensional**: 1
- unsigned **orientation_reversed**: 1
- unsigned **enable_bidi**: 1
- unsigned **ignore_formatting_char**: 1
- unsigned **fixed_width**: 1
- unsigned **anti_alias**: 1
- unsigned **disable_overlapping_adjustment**: 1
- unsigned int **min_line_ascent**
- unsigned int **min_line_descent**
- unsigned int **max_line_ascent**
- unsigned int **max_line_descent**
- unsigned int **max_line_width**
- unsigned int **tab_width**
- void(∗ **format**)(int line, int y, int ∗indent, int ∗width)
- int(∗ **line_break**)(**MText** ∗mt, int pos, int from, int to, int line, int y)
- int **with_cursor**
- int **cursor_pos**
- int **cursor_width**
- int **cursor_bidi**
- int **partial_update**
- int **disable_caching**
- **MDrawRegion clip_region**

3.5.1 説明

テキスト表示制御の型宣言.

MDrawControl (p. 162) 型は、M-text をどう表示するかを制御する構造体である。

3.5.2 構造体

3.5.2.1 unsigned MDrawControl::as_image

0 でなければ、 M-text を画像として、すなわち背景を M-text のフェースで指定されている背景色で埋めて表示する。そうでなければ背景は変わらない。

3.5.2.2 unsigned MDrawControl::align_head

0 でなく、各行の最初のグリフの lbearing が負ならば、グリフを水平に右にずらして、指定した位置より左にピクセルが描かれないようにする。

3.5.2.3 unsigned MDrawControl::two_dimensional

0 でなければ、M-text を 2 次元的に、すなわち M-text 中の newline で改行し、続く文字は次の行に表示する。もし <format> が NULL でなく、その関数が 0 でない行幅を返せば、その幅より長い行も改行される。

3.5.2.4 unsigned MDrawControl::orientation_reversed

0 でなければ、M-text を指定した位置の右に表示する。

3.5.2.5 unsigned MDrawControl::enable_bidi

0 なければ、bidi テキスト用にグリフを正しく整列する。

3.5.2.6 unsigned MDrawControl::ignore_formatting_char

0 でなければ、ユニコードに置ける一般カテゴリが Cf (Other, format) である文字を表示しない。

3.5.2.7 unsigned MDrawControl::fixed_width

0 でなければ、端末用のグリフを表示する。未実装。

3.5.2.8 unsigned MDrawControl::anti_alias

0 でなければ、アンチエーリアスでグリフを表示する。（バックエンドのフォントドライバがアンチエーリアス機能を持つ場合のみ。）

3.5.2.9 unsigned MDrawControl::disable_overlapping_adjustment

0 でなければ、フォント境界での水平方向のグリフの重なりを避けるためのグリフ位置の調整を無効にする。

3.5.2.10 unsigned int MDrawControl::min_line_ascent

0 でなければ、値は行の ascent の最小値を示す。

3.5.2.11 unsigned int MDrawControl::min_line_descent

0 でなければ、値は行の descent の最小値を示す。

3.5.2.12 unsigned int MDrawControl::max_line_ascent

0 でなければ、値は行の ascent の最大値を示す。

3.5.2.13 unsigned int MDrawControl::max_line_descent

0 でなければ、値は行の descent の最大値を示す。

3.5.2.14 unsigned int MDrawControl::max_line_width

0 でなければ、値はこのディスプレイ上で各行が占めることのできるピクセル数を示す。 0 は限定されないことを意味する。<format> が NULL でなければ無視される。

3.5.2.15 unsigned int MDrawControl::tab_width

0 でなければ、値はタブストップ間の距離をコラム単位 （コラムはフレームのデフォルトフォントにおける空白文字の幅である）で示す。 0 は 8 を意味する。

3.5.2.16 void(∗ MDrawControl::format)(int line, int y, int ∗indent, int ∗width)

0 でなければ、値は関数であり、その関数は行番号 LINE と座標 Y に基づいて各行のインデントと最大幅を計算し、それぞれを INDENT と WIDTH で指される場所に保存する。

インデントは、各行の最初のグリフを右（メンバ <orientation_reversed> が 0 の時）あるいは左（それ以外の時）に何ピクセルずらすかを指定する。値が負ならば逆方向にずらす。

最大幅は、各行がディスプレイ上で占めることのできるピクセル数の最大値である。値が 0 の場合は制限を受けないことを意味する。

LINE と Y は改行文字によって行が改まった際には 0 にリセットされ、長い行が最大幅の制限によって改行されるたびに 1 増やされる。

これは <two_dimensional> が 0 でない場合にのみ有効である。

3.5.2.17 int(∗ MDrawControl::line_break)(MText ∗mt, int pos, int from, int to, int line, int y)

NULL でなければ、値は行が最大幅中に収まらない場合に行を改める位置を計算する関数である。 POS は最大幅に収まる最後の文字の次の文字の位置である。FROM は行の最初の文字の位置、TO は最大幅が指定されていなければその行に表示される最後の文字の位置である。LINE と Y は <format> の引数と同様である。

この関数は行を改める文字位置を返さなくてはならない。また MT を変更してはならない。

関数 **mdraw_default_line_break()** (p. 144) は、空白を語の区切りとして用いるスクリプト用として有用である。

3.5.2.18 int MDrawControl::with_cursor

ゼロでなければ <cursor_width> にしたがってカーソルを表示する。

3.5.2.19 int MDrawControl::cursor_pos

カーソルを表示する文字位置を示す。最大の文字位置より大きければ、カーソルは M-text の最後の文字の隣に表示される。負ならば、 <cursor_width> が 0 でなくてもカーソルは表示されない。

3.5.2.20 int MDrawControl::cursor_width

0 でなければ、<cursor_pos> にカーソルを表示する。 値が正ならば、カーソルの幅はその値（ピクセル単位）である。 負ならば、カーソルのあるグリフと同じ幅である。

3.5.2.21 int MDrawControl::cursor_bidi

If 0 でなく、かつ <cursor_width> も 0 でなければ、バーカーソルを文字位置 <cursor_pos> と論理的にそれの前にある文字の 2 ヶ所に表示する。 双方とも 1 ピクセル幅で、上か下に水平の飾りがつく。

3.5.2.22 int MDrawControl::partial_update

0 でなければ、テキストの一部分を表示する際に、前後のテキストのうちその表示領域に侵入する部分も表示する。 たとえば、タイ語テキスト 子音-母音-子音 というシークエンスのいくつかは、母音が二つの子音の間に上にのるように描かれる。 このようなシークエンスがすでに描かれており、最後の子音だけを描き直す場合（たとえば、カーソル位置を更新する際など）このメンバが 0 であれば、母音の右半分が消されてしまう。これを 0 以外にすることによって、そのような際にも 子音-母音-子音 のシークエンスを正しく表示し続けることができる。

3.5.2.23 int MDrawControl::disable_caching

0 でなければ、M-text の表示に関する情報をキャッシュしない。

3.5.2.24 MDrawRegion MDrawControl::clip_region

NULL でなければ表示エリアを指定された領域に限定する。

3.6 構造体 MDrawGlyph

グリフの寸法とフォントに関する情報の型宣言.

変数

- int **from**
- int **to**
- int **glyph_code**
- int **x_advance**
- int **y_advance**
- int **x_off**
- int **y_off**
- int **lbearing**
- int **rbearing**
- int **ascent**
- int **descent**
- **MFont** ∗ **font**
- **MSymbol font_type**
- void ∗ **fontp**

3.6.1 説明

グリフの寸法とフォントに関する情報の型宣言.

MDrawGlyph (p. 166) 型はグリフの寸法とフォントに関する情報を含む構造体である。
mdraw_glyph_list() (p. 144) はこれを用いる。

3.6.2 構造体

3.6.2.1 int MDrawGlyph::from

グリフに対応する文字の範囲.

3.6.2.2 int MDrawGlyph::to

3.6.2.3 int MDrawGlyph::glyph_code

フォント内のグリフコード。

3.6.2.4 int MDrawGlyph::x_advance

グリフの論理的幅。次のグリフとの名目上の距離。

3.6.2.5 int MDrawGlyph::y_advance

グリフの論理的高さ。次のグリフとの名目上の距離。

3.6.2.6 int MDrawGlyph::x_off

グリフの位置に対する X オフセット.

3.6.2.7 int MDrawGlyph::y_off

グリフの位置に対する Y オフセット.

3.6.2.8 int MDrawGlyph::lbearing

グリフの寸法 (left-bearing).

3.6.2.9 int MDrawGlyph::rbearing

グリフの寸法 (right-bearing).

3.6.2.10 int MDrawGlyph::ascent

グリフの寸法 (ascent).

3.6.2.11 int MDrawGlyph::descent

グリフの寸法 (descent).

3.6.2.12 MFont* MDrawGlyph::font

グリフに使われるフォント。見つからなければ NULL。

3.6.2.13 MSymbol MDrawGlyph::font_type

フォントのタイプ。Mx、Mfreetype、Mxft のいずれか。

3.6.2.14 void* MDrawGlyph::fontp

フォントの構造体へのポインタ。実際の型は <font_type> メンバが Mx なら (XFontStruct *)、 Mfreetype なら FT_Face、Mxft なら (XftFont *)。

3.7 構造体 MDrawGlyphInfo

グリフに関する情報の型宣言.

変数

- int **from**
- int **to**
- int **line_from**
- int **line_to**
- int **x**
- int **y**
- **MDrawMetric metrics**
- **MFont** * **font**
- int **prev_from**
- int **next_to**
- int **left_from**
- int **left_to**
- int **right_from**
- int **right_to**
- int **logical_width**

3.7.1 説明

グリフに関する情報の型宣言.

MDrawGlyphInfo (p. 168) 型はグリフに関する情報を含む構造体である。 **mdraw_glyph_info()** (p. 144) はこれを用いる。

3.7.2 構造体

3.7.2.1 int MDrawGlyphInfo::from

グリフに対応する文字の範囲の開始位置.

3.7.2.2 int MDrawGlyphInfo::to

グリフに対応する文字の範囲の終了位置.

3.7.2.3 int MDrawGlyphInfo::line_from

一行のグリフの列に対応する文字の範囲の開始位置.

3.7.2.4 int MDrawGlyphInfo::line_to

一行のグリフの列に対応する文字の範囲の終了位置.

3.7.2.5 int MDrawGlyphInfo::x

グリフの X 座標.

3.7.2.6 int MDrawGlyphInfo::y

グリフの Y 座標.

3.7.2.7 MDrawMetric MDrawGlyphInfo::metrics

グリフの寸法.

3.7.2.8 MFont* MDrawGlyphInfo::font

グリフに使われるフォント。見つからなければ NULL。

3.7.2.9 int MDrawGlyphInfo::prev_from

論理的な前のグリフに対応する文字の範囲。メンバ prev_to は、メ ンバ from と同じであるはずなので不要である。

3.7.2.10 int MDrawGlyphInfo::next_to

論理的な後のグリフに対応する文字の範囲。メンバ next_from は メンバ to と同じであるはずなので不要である。

3.7.2.11 int MDrawGlyphInfo::left_from

表示上の左のグリフに対応する文字の範囲の開始位置。

3.7.2.12 int MDrawGlyphInfo::left_to

表示上の左のグリフに対応する文字の範囲の終了位置。

3.7.2.13 int MDrawGlyphInfo::right_from

表示上の右のグリフに対応する文字の範囲の開始位置。

3.7.2.14 int MDrawGlyphInfo::right_to

表示上の右のグリフに対応する文字の範囲の終了位置。

3.7.2.15 int MDrawGlyphInfo::logical_width

グリフの論理的幅。次のグリフとの名目上の距離。

3.8 構造体 MDrawMetric

グリフとテキストの寸法の型宣言.

変数

- int **x**
- int **y**
- unsigned int **width**
- unsigned int **height**

3.8.1 説明

グリフとテキストの寸法の型宣言.

MDrawMetric (p. 170) はグリフと表示されたテキストの寸法用の型である。 また、表示デバイスの矩形領域を表すのにも用いられる。

3.8.2 構造体

3.8.2.1 int MDrawMetric::x

X coordinates of a glyph or a text.

3.8.2.2 int MDrawMetric::y

Y coordinates of a glyph or a text.

3.8.2.3 unsigned int MDrawMetric::width

Pixel width of a glyph or a text.

3.8.2.4 unsigned int MDrawMetric::height

Pixel height of a glyph or a text.

3.9 構造体 MDrawTextItem

textitem の型宣言.

変数

- **MText ∗ mt**
- int **delta**
- **MFace ∗ face**
- **MDrawControl ∗ control**

3.9.1 説明

textitem の型宣言.

MDrawTextItem (p. 171) は **テキストアイテム** オブジェクト用の型である。 各テキストアイテムは、 1 個の M-text と、その表示を制御するための情報を含んでいる。

3.9.2 構造体

3.9.2.1 MText∗ MDrawTextItem::mt

M-text.

3.9.2.2 int MDrawTextItem::delta

M-text 表示前に行なう X 軸方向の位置調整 (ピクセル単位)

3.9.2.3 MFace∗ MDrawTextItem::face

フェースオブジェクトへのポインタ。フェースの各プロパティは Mnil でなければ <mt> で指定されたフェースの同じプロパティに優先する

3.9.2.4 MDrawControl∗ MDrawTextItem::control

表示制御オブジェクトへのポインタ。 **mdraw_text_with_control()** (p. 142) はこのオブジェクトを用いて M-text <mt> を表示する。

3.10 構造体 MFaceBoxProp

フェースの囲み枠指定用型宣言.

変数

- unsigned **width**
- **MSymbol color_top**
- **MSymbol color_bottom**
- **MSymbol color_left**
- **MSymbol color_right**
- unsigned **inner_hmargin**
- unsigned **inner_vmargin**
- unsigned **outer_hmargin**
- unsigned **outer_vmargin**

3.10.1 説明

フェースの囲み枠指定用型宣言.

MFaceBoxProp (p. 172) はフェースの **Mbox** (p. 134) プロパティの詳細を指定する型である。 このプロパティの値はこの型のオブジェクトへのポインタでなくてはならない。

3.10.2 構造体

3.10.2.1 unsigned MFaceBoxProp::width

線幅（ピクセル単位）.

3.10.2.2 MSymbol MFaceBoxProp::color_top

Colors of borders.

3.10.2.3 MSymbol MFaceBoxProp::color_bottom

3.10.2.4 MSymbol MFaceBoxProp::color_left

3.10.2.5 MSymbol MFaceBoxProp::color_right

3.10.2.6 unsigned MFaceBoxProp::inner_hmargin

Margins

3.10.2.7 unsigned MFaceBoxProp::inner_vmargin

3.10.2.8 unsigned MFaceBoxProp::outer_hmargin

3.10.2.9 unsigned MFaceBoxProp::outer_vmargin

3.11 構造体 MFaceHLineProp

フェースの水平線指定用型宣言.

Public 型

- enum **MFaceHLineType** {
  MFACE_HLINE_BOTTOM,
  MFACE_HLINE_UNDER,
  MFACE_HLINE_STRIKE_THROUGH,
  MFACE_HLINE_OVER,
  MFACE_HLINE_TOP }

変数

- enum **MFaceHLineProp::MFaceHLineType type**
- unsigned **width**
- **MSymbol color**

3.11.1 説明

フェースの水平線指定用型宣言.

MFaceHLineProp (p. 173) はフェースの **Mhline** (p. 134) プロパティの詳細を指定する型である。このプロパティの値はこの型のオブジェクトでなくてはならない。

3.11.2 列挙型

3.11.2.1 enum MFaceHLineProp::MFaceHLineType

水平線のタイプ.

列挙型の値:

MFACE_HLINE_BOTTOM

MFACE_HLINE_UNDER

MFACE_HLINE_STRIKE_THROUGH

MFACE_HLINE_OVER

MFACE_HLINE_TOP

3.11.3 構造体

3.11.3.1 enum MFaceHLineProp::MFaceHLineType MFaceHLineProp::type

水平線のタイプ.

3.11.3.2 unsigned MFaceHLineProp::width

線幅（ピクセル単位）.

3.11.3.3 MSymbol MFaceHLineProp::color

線の色. Mnil ならば、統合したフェースの前景色が使われる。

3.12 構造体 MFLTFont

FLT ドライバが使うフォントの型.

変数

- **MSymbol family**
- int **x_ppem**
- int **y_ppem**
- int(∗ **get_glyph_id**)(struct _MFLTFont ∗font, **MFLTGlyphString** ∗gstring, int from, int to)
- int(∗ **get_metrics**)(struct _MFLTFont ∗font, **MFLTGlyphString** ∗gstring, int from, int to)
- int(∗ **check_otf**)(struct _MFLTFont ∗font, **MFLTOtfSpec** ∗spec)
- int(∗ **drive_otf**)(struct _MFLTFont ∗font, **MFLTOtfSpec** ∗spec, **MFLTGlyphString** ∗in, int from, int to, **MFLTGlyphString** ∗out, **MFLTGlyphAdjustment** ∗adjustment)
- void ∗ **internal**

3.12.1 説明

FLT ドライバが使うフォントの型.

型 **MFLTFont** (p. 175) は、FLT ドライバが使うフォントに関する情報を格納するた めの構造体である。

3.12.2 構造体

3.12.2.1 MSymbol MFLTFont::family

フォントのファミリー名。フォントに適した FLT を探す際に重要でな い場合 (たとえば OpenType フォントの場合など) は、**Mnil** (p. 16) でよい。

3.12.2.2 int MFLTFont::x_ppem

フォントの水平サイズを pixels per EM で表現したもの。

3.12.2.3 int MFLTFont::y_ppem

フォントの垂直サイズを pixels per EM で表現したもの。

3.12.2.4 int(∗ MFLTFont::get_glyph_id)(struct _MFLTFont ∗font, MFLTGlyphString ∗gstring, int from, int to)

GSTRING 内の FROM から TO 直前までの各グリフに対応するグリフ ID を取得するための callback 関数。もしあるグリフのメンバー <encoded> がゼロならば、そのグリフのメンバー <code> は文字コードで ある。この関数はその文字コードを FONT のグリフ ID に変換しなくては ならない。

3.12.2.5 int(∗ MFLTFont::get_metrics)(struct _MFLTFont ∗font, MFLTGlyphString ∗gstring, int from, int to)

GSTRING 内の FROM から TO 直前までの各グリフに対応するメトリッ クを取得するための callback 関数。もしあるグリフのメンバー <measured> がゼロならば、この関数はそのグリフのメンバー <xadv>, <yadv>, <ascent>, <descent>, <lbearing>, および <rbearing> をセッ トしなければならない。

3.12.2.6 int(∗ MFLTFont::check_otf)(struct _MFLTFont ∗font, MFLTOtfSpec ∗spec)

フォントがある特定のスクリプト/言語に対する GSUB/GPOS OpenType フィーチャーを持つか否かを調べる callback 関数。この関数 はフォントが SPEC を満たすときは 1 を、そうでないときは 0 を返さな ければならない。フォントが OpenType テーブルを持たないときは NULL でなければならない。

3.12.2.7 int(∗ MFLTFont::drive_otf)(struct _MFLTFont ∗font, MFLTOtfSpec ∗spec, MFLTGlyphString ∗in, int from, int to, MFLTGlyphString ∗out, MFLTGlyphAdjustment ∗adjustment)

IN 内の FROM から TO 直前までの各グリフに SPEC 内の各 OpenType フィーチャーを適用するための callback 関数。適用結果のグリフ列は OUT の末尾に追加される。OUT が短か過ぎて結果を追加し切れない場合 は -2 を返さなくてはならない。フォントが OpenType テーブルを持た ない場合は NULL でなければならない。

3.12.2.8 void∗ MFLTFont::internal

m17n-lib の内部作業用。NULL に初値化される。

3.13 構造体 MFLTGlyph

グリフに関する情報の型.

変数

- int **c**
- unsigned int **code**
- int **from**
- int **to**
- int **xadv**
- int **yadv**
- int **ascent**
- int **descent**
- int **lbearing**
- int **rbearing**
- int **xoff**
- int **yoff**
- unsigned **encoded**: 1
- unsigned **measured**: 1
- unsigned **adjusted**: 1

3.13.1 説明

グリフに関する情報の型.

型 **MFLTGlyph** (p. 177) は、グリフに関する情報を格納する構造体である。

3.13.2 構造体

3.13.2.1 int MFLTGlyph::c

グリフの (Unicode における) 文字コード。関数 **mflt_find()** (p. 108) と **mflt_run()** (p. 109) を呼び出す前セットすべき唯一のメンバーである。

3.13.2.2 unsigned int MFLTGlyph::code

フォント内におけるそのグリフの ID。

3.13.2.3 int MFLTGlyph::from

MFLTGlyphString (p. 180) の中で、このグリフによって置き換えられる部分の先頭のインデクス。

3.13.2.4 int MFLTGlyph::to

MFLTGlyphString (p. 180) の中で、このグリフによって置き換えられる部分の末尾のインデクス。

3.13.2.5 int MFLTGlyph::xadv

横書き時の送り幅を 26.6 fractional pixel format で表現したもの。

3.13.2.6 int MFLTGlyph::yadv

縦書き時の送り高を 26.6 fractional pixel format で表現したもの。

3.13.2.7 int MFLTGlyph::ascent

このグリフのインクメトリックを 26.6 fractional pixel format で表現したもの。

3.13.2.8 int MFLTGlyph::descent

3.13.2.9 int MFLTGlyph::lbearing

3.13.2.10 int MFLTGlyph::rbearing

3.13.2.11 int MFLTGlyph::xoff

グリフ位置決めの際の水平・垂直調整値を、 26.6 fractional pixel format で表現したもの。

3.13.2.12 int MFLTGlyph::yoff

3.13.2.13 unsigned MFLTGlyph::encoded

メンバー <code> に既にグリフ ID がセットされているか否かを示すフラグ。

3.13.2.14 unsigned MFLTGlyph::measured

メンバー <xadv> から <rbearing> までの各メトリックが既に計算済か否かを示すフラグ。

3.13.2.15 unsigned MFLTGlyph::adjusted

グリフのメトリックが調整済みか否か、 すなわち以下のうち 1 つ以上が成立していることを示すフラグ。 <xadv> が標準の値と異なる、 <yadv> が標準の値と異なる、 <xoff> がゼロでない、 <yoff> がゼロでない。

3.14 構造体 MFLTGlyphAdjustment

グリフ位置調整情報のための型.

変数

- int **xadv**
- int **yadv**
- int **xoff**
- int **yoff**
- short **back**
- unsigned **advance_is_absolute**: 1
- unsigned **set**: 1

3.14.1 説明

グリフ位置調整情報のための型.

型 **MFLTGlyphAdjustment** (p. 179) は、グリフのメトリック/位置の調整に関する情報を格納するための構造体であり、 **MFLTFont** (p. 175) の callback 関数 **drive_otf** に渡される。

3.14.2 構造体

3.14.2.1 int MFLTGlyphAdjustment::xadv

水平・垂直方向の送り量の調整値を 26.6 fractional pixel format で表現したもの。

3.14.2.2 int MFLTGlyphAdjustment::yadv

3.14.2.3 int MFLTGlyphAdjustment::xoff

グリフ位置決めための水平・垂直調整値を 26.6 fractional pixel format で表現したもの。

3.14.2.4 int MFLTGlyphAdjustment::yoff

3.14.2.5 short MFLTGlyphAdjustment::back

グリフ描画のために戻るべきグリフ数。

3.14.2.6 unsigned MFLTGlyphAdjustment::advance_is_absolute

非ゼロのとき、メンバー <xadv> と <yadv> は絶対値である。 すなわちその値をグリフ本来の送り幅に加算してはならない。

3.14.2.7 unsigned MFLTGlyphAdjustment::set

他のメンバーのうち最低 1 個が非ゼロのときのみ、1 にセットされる。

3.15 構造体 MFLTGlyphString

グリフ列の情報のための型.

変数

- int **glyph_size**
- **MFLTGlyph** ∗ **glyphs**
- int **allocated**
- int **used**
- unsigned int **r2l**

3.15.1 説明

グリフ列の情報のための型.

型 **MFLTGlyphString** (p. 180) は、グリフ列の情報を格納するための構造体である。

3.15.2 構造体

3.15.2.1 int MFLTGlyphString::glyph_size

メンバー **glyphs** (p. 180) の指す配列の要素が占める実バイト数。 この値は "sizeof (MFLTGlyph)" 以上でなければならない。

3.15.2.2 MFLTGlyph∗ MFLTGlyphString::glyphs

グリフの配列。

3.15.2.3 int MFLTGlyphString::allocated

glyphs (p. 180) 内に配置されている要素の数。

3.15.2.4 int MFLTGlyphString::used

glyphs (p. 180) 内で使用中の要素の数。

3.15.2.5 unsigned int MFLTGlyphString::r2l

グリフが右から左へと描かれるべきか否かを示すフラグ。

3.16 構造体 MFLTOtfSpec

GSUB および GPOS OpenType テーブルの仕様のための型.

変数

- **MSymbol sym**
- unsigned int **script**
- unsigned int **langsys**
- unsigned int ∗ **features** [2]

3.16.1 説明

GSUB および GPOS OpenType テーブルの仕様のための型.

型 **MFLTOtfSpec** (p. 181) は、GSUB および GPOS フィーチャーの情報を格納するた めの構造体である。これらフィーチャーは特定のスクリプトおよび言語シ ステムのものであり、グリフ列に適用される。

3.16.2 構造体

3.16.2.1 MSymbol MFLTOtfSpec::sym

この仕様を表わすユニークなシンボル。 FLT の **OTF-SPEC** (p. 211) と同一の値である。

3.16.2.2 unsigned int MFLTOtfSpec::script

スクリプトおよび言語システムのタグ。

3.16.2.3 unsigned int MFLTOtfSpec::langsys

3.16.2.4 unsigned int∗ MFLTOtfSpec::features[2]

GSUB フィーチャーを第 1 要素、GPOS フィーチャーを第 2 要素とする配 列。各配列の末尾は 0 で示される。もしある要素が 0xFFFFFFFF ならば、 以前の全フィーチャーをその順序で適用し、更に以降の要素として現わ れるフィチャー以外のすべてを適用する。フィーチャーが 1 つもない場合 は NULL でもよい。

3.17 構造体 MInputContext

入力コンテクスト用構造体.

変数

- **MInputMethod** ∗ **im**
- **MText** ∗ **produced**
- void ∗ **arg**
- int **active**
- struct {
  int **x**
  int **y**
  int **ascent**
  int **descent**
  int **fontsize**
  MText ∗ **mt**
  int **pos**
  } **spot**

- void ∗ **info**
- **MText** ∗ **status**
- int **status_changed**
- **MText** ∗ **preedit**
- int **preedit_changed**
- int **cursor_pos**
- int **cursor_pos_changed**
- **MPlist** ∗ **candidate_list**
- int **candidate_index**
- int **candidate_from**
- int **candidate_to**
- int **candidate_show**
- int **candidates_changed**
- **MPlist** ∗ **plist**

3.17.1 説明

入力コンテクスト用構造体.

MInputContext （p. 182）は、入力コンテクストオブジェクト用の構造体の型である。

3.17.2 構造体

3.17.2.1 MInputMethod∗ MInputContext::im

入力メソッドへの逆ポインタ。関数 **minput_create_ic()** (p. 96) によって設定される。

3.17.2.2 MText∗ MInputContext::produced

入力メソッドによって生成される M-text。関数 **minput_lookup()** (p. 96) によって設定される。

3.17.2.3 void* MInputContext::arg

関数 **minput_create_ic()** (p. 96) に渡される引数。

3.17.2.4 int MInputContext::active

入力コンテクストがアクティブかどうかを示すフラグ。 入力コンテクストが生成された時点では値は 1（アクティブ）であり、関数 **minput_toggle()** (p. 97) によってトグルされる。

3.17.2.5 int MInputContext::x

スポットの X, Y 座標.

3.17.2.6 int MInputContext::y

3.17.2.7 int MInputContext::ascent

スポットのアセントとディセントのピクセル数.

3.17.2.8 int MInputContext::descent

3.17.2.9 int MInputContext::fontsize

preedit テキスト用のフォントサイズ (1/10 ポイント単位).

3.17.2.10 MText* MInputContext::mt

スポット上の M-text、または NULL.

3.17.2.11 int MInputContext::pos

<mt> におけるスポットの文字位置.

3.17.2.12 struct { ... } MInputContext::spot

入力コンテクストのスポットの位置と大きさ.

3.17.2.13 void* MInputContext::info

以下のメンバの使用法は入力メソッドドライバによって異なる。 以下の説明は、内部入力メソッド用の入力ドライバに対するものである。 これらは関数 <im>->driver.filter() によって設定される。
<im>->driver.create_ic() が設定する追加情報へのポインタ。 入力コンテクストの内部状態を記録するために用いられる。

3.17.2.14 MText* MInputContext::status

入力コンテクストの現在の状態を表す M-text

3.17.2.15 int MInputContext::status_changed

関数 <im>->driver.filter() は、<status> を変えた際にこの値を 1 に設定する。

3.17.2.16 MText* MInputContext::preedit

現在の preedit テキストを含む M-text。関数 <im>->driver.filter() によって設定される。

3.17.2.17 int MInputContext::preedit_changed

関数 <im>->driver.filter() は、<preedit> を変えた際にこの値を 1 に設定する。

3.17.2.18 int MInputContext::cursor_pos

<preedit> のカーソル位置

3.17.2.19 int MInputContext::cursor_pos_changed

関数 <im>->driver.filter() は、<cursor_pos> を変えた際にこの値を 1 に設定する。

3.17.2.20 MPlist* MInputContext::candidate_list

現在の候補グループの Plist 。各要素は M-text か plist である。 要素が M-text の場合（キーが Mtext である場合）には、そのグループの候補はその M-text 中の各文字である。 要素が plist の場合（キーが Mplist である場合）には、そのリストの各要素は M-text であり、それらがそのグループの候補となる。

3.17.2.21 int MInputContext::candidate_index

現在選択されている候補が全候補中で何番目かをを示すインデックス。 最初の候補のインデックスは 0。最初の候補グループに七つの候補が含まれており、この値が 8 ならば、現在の候補は二番目の候補グループの二番目の要素ということになる。

3.17.2.22 int MInputContext::candidate_from

preedit テキスト中で、<candidate_list> に対応する最初と最後の位置。

3.17.2.23 int MInputContext::candidate_to

3.17.2.24 int MInputContext::candidate_show

現在の候補グループを表示するかどうかを示すフラグ。 関数 <im>->driver.filter() は、入力メソッドが候補の表示を要求した時この値を 1 に、それ以外の時 0 に設定する。

3.17.2.25 int MInputContext::candidates_changed

関数 <im>->driver.filter() は、上記のメンバ <candidate_XXX> の 1 つでも変更した際には、この値を `enum MInputCandidatesChanged` のビット単位での論理 OR に設定する。そう でなければ 0 に設定する。

3.17.2.26 MPlist* MInputContext::plist

<im>->driver の関数群によって自由に使用できる plist。 内部入力メソッド用ドライバはこれをコールバック関数との引数や返値 の受渡しに使用する。関数 <im>->driver.create_ic() はこの plist を空に設定する。関数 <im>->driver.destroy_ic() は **m17n_object_unref()** (p. 12) を用いてこの plist を解放する。

3.18 構造体 MInputDriver

入力ドライバ用構造体.

変数

- int(∗ **open_im**)(**MInputMethod** ∗im)
  入力メソッドをオープンする.
- void(∗ **close_im**)(**MInputMethod** ∗im)
  入力メソッドをクローズする.
- int(∗ **create_ic**)(**MInputContext** ∗ic)
  入力コンテクストを生成する.
- void(∗ **destroy_ic**)(**MInputContext** ∗ic)
  入力コンテクストを破壊する.
- int(∗ **filter**)(**MInputContext** ∗ic, **MSymbol** key, void ∗arg)
  入力キーをフィルタする.
- int(∗ **lookup**)(**MInputContext** ∗ic, **MSymbol** key, void ∗arg, **MText** ∗mt)
  入力コンテクストで生成されるテキストの獲得.
- **MPlist** ∗ **callback_list**
  コールバック関数のリスト.

3.18.1 説明

入力ドライバ用構造体.

MInputDriver (p. 186) は、入力メソッドを取り扱う関数を含む入力メソッドドライバの構造体の型である。

3.18.2 構造体

3.18.2.1 int(∗ MInputDriver::open_im)(MInputMethod ∗im)

入力メソッドをオープンする.

この関数は、入力メソッド **im** をオープンする。**im** の <info> 以外の全メンバーがセットされた後で、関数 **minput_open_im()** (p. 95) から呼ばれる。**im** をオープンできれば 0 を、できなければ -1 を返す。 この関数は **im->info** を設定して、他のドライバ関数から参照される情報を保持することができる。

3.18.2.2 void(∗ MInputDriver::close_im)(MInputMethod ∗im)

入力メソッドをクローズする.

この関数は、入力メソッド **im** をクローズする。関数 **minput_close_im()** (p. 96) から呼ばれる。 入力メソッドのクローズがすべて終了した時点で、この関数は **im->info** に割り当てられているメモリを (あれば) すべて開放する。 ただし、**im** の他のメンバに影響を与えてはならない。

3.18.2.3 int(∗ MInputDriver::create_ic)(MInputContext ∗ic)

入力コンテクストを生成する.

この関数は入力コンテクスト **ic** を生成する。 **ic** の <info> 以外の全メンバーがセットされた後で、関数 **minput_create_ic()** (p. 96) から呼ばれる。 **ic** を生成できれば 0 を、できなければ -1 を返す。 この関数は **ic->info** を設定して、他のドライバ関数から参照される情報を保持することができる。

3.18.2.4 void(∗ MInputDriver::destroy_ic)(MInputContext ∗ic)

入力コンテクストを破壊する.

関数 **minput_destroy_ic()** (p. 96) から呼ばれ、入力コンテクスト **ic** を破壊する。入力コンテクストの破壊がすべて終了した時点で、**ic->info** に割り当てられているメモリを (あれば) すべて開放する。ただし、**ic** の他のメンバに影響を与えてはならない。

3.18.2.5 int(∗ MInputDriver::filter)(MInputContext ∗ic, MSymbol key, void ∗arg)

入力キーをフィルタする.

関数 **minput_filter()** (p. 96) から呼ばれ、入力キーをフィルタする。引数 **key**, **arg** は関数 **minput_filter()** (p. 96) のものと同じ。

この関数は **key** を処理し、**ic** の内部状態を更新する。 **key** が入力メソッドに吸収されてテキストが生成されなかった場合には、 1 を返す。そうでなければ 0 を返す。

メンバ <callback> に必要であれば、**ic->status**, **ic->preedit**, **ic->cursor_pos**, **ic->ncandidates**, **ic->candidates**, **ic->produced** を更新できる。

arg の意味は入力メソッドドライバに依存する。例は `minput_default_driver` または `minput_gui_driver` の説明を参照のこと。

3.18.2.6 int(∗ MInputDriver::lookup)(MInputContext ∗ic, MSymbol key, void ∗arg, MText ∗mt)

入力コンテクストで生成されるテキストの獲得.

関数 **minput_lookup()** (p. 96) から呼ばれ、入力コンテクスト **ic** で生成されるテキストを検索する。入力キー **key** によって生成されるテキストがあれば、M-text **mt** に追加する。 **key** が入力メソッド **ic** によって正しく処理されれば 0 を返す。そうでなければ 1 を返す。

arg の意味は入力メソッドドライバに依存する。例は `minput_default_driver` または `minput_gui_driver` の説明を参照の こと。

3.18.2.7 MPlist∗ MInputDriver::callback_list

コールバック関数のリスト.

コールバック関数のリスト。キーは次のいずれか。 **Minput_preedit_start**, **Minput_preedit_draw**, **Minput_preedit_done**, **Minput_status_start**, **Minput_status_draw**, **Minput_status_done**, **Minput_candidates_start**, **Minput_candidates_draw**, **Minput_candidates_done**, **Minput_set_spot**, **Minput_toggle**, **Minput_reset**, **Minput_get_surrounding_text**, **Minput_delete_surrounding_text**。 値は **MInputCallbackFunc** (p. 95) 型の関数。

3.19 構造体 MInputGUIArgIC

関数 **minput_create_ic()** (p. 96) の引数の型宣言.

変数

- **MFrame * frame**
- **MDrawWindow client**
- **MDrawWindow focus**

3.19.1 説明

関数 **minput_create_ic()** (p. 96) の引数の型宣言.

MInputGUIArgIC (p. 188) は、関数 **minput_create_ic()** (p. 96) が内部入力メソッドの入力コンテクストを生成する際の、引数 **arg** 用の型である。

3.19.2 構造体

3.19.2.1 MFrame* MInputGUIArgIC::frame

クライアントのフレーム

3.19.2.2 MDrawWindow MInputGUIArgIC::client

preedit テキストと status テキストを表示するウィンドウ

3.19.2.3 MDrawWindow MInputGUIArgIC::focus

入力コンテクストがフォーカスをおいているウィンドウ

3.20 構造体 MInputMethod

入力メソッドの構造体.

変数

- **MSymbol language**
- **MSymbol name**
- **MInputDriver driver**
- void * **arg**
- void * **info**

3.20.1 説明

入力メソッドの構造体.

MInputMethod (p. 189) は、入力メソッドオブジェクト用の構造体の型である。

3.20.2 構造体

3.20.2.1 MSymbol MInputMethod::language

どの言語用の入力メソッドか。 入力メソッドが外部のものである場合の値は Mnil 。

3.20.2.2 MSymbol MInputMethod::name

入力メソッドの名前。外部メソッドである場合には、Minput_driver をキーとするプロパティを持ち、その値は適切な入力メソッドドライバへのポインタでなくてはならない。

3.20.2.3 MInputDriver MInputMethod::driver

その入力メソッド用の入力メソッドドライバ。

3.20.2.4 void* MInputMethod::arg

minput_open_im() (p. 95) に渡される引数。

3.20.2.5 void* MInputMethod::info

<driver>.open_im() が設定する追加情報へのポインタ。

3.21 構造体 MInputXIMArgIC

関数 **minput_create_ic()** (p. 96) の引数 **arg** によって指される構造体.

変数

- XIMStyle **input_style**
- Window **client_win**
- Window **focus_win**
- XVaNestedList **preedit_attrs**
- XVaNestedList **status_attrs**

3.21.1 説明

関数 **minput_create_ic()** (p. 96) の引数 **arg** によって指される構造体.

MInputXIMArgIC (p. 190) 型は、関数 **minput_create_ic()** (p. 96) が名前 **Mxim** (p. 147) を持つ外部入力メソッド用に呼ばれる際に、引数 **arg** によって指される構造体である。

3.21.2 構造体

3.21.2.1 XIMStyle MInputXIMArgIC::input_style

`XCreateIC` の `XNInputStyle` に続く引数として用いられる。 ゼロならば、 (`XIMPreeditNothing` | `XIMStatusNothing`) が用 いられ、 <preedit_attrs> と <status_attrs> は `NULL` に設定される。

3.21.2.2 Window MInputXIMArgIC::client_win

`XCreateIC` の `XNClientWindow` に続く引数として用いられる。

3.21.2.3 Window MInputXIMArgIC::focus_win

`XCreateIC` の `XNFocusWindow` に続く引数として用いられる。

3.21.2.4 XVaNestedList MInputXIMArgIC::preedit_attrs

`NULL` でなければ、 `XCreateIC` following の `XNPreeditAttributes` に続く引数として用いられる。

3.21.2.5 XVaNestedList MInputXIMArgIC::status_attrs

`NULL` でなければ、 `XCreateIC` following の `XNStatusAttributes` に続く引数として用いられる。

3.22 構造体 MInputXIMArgIM

関数 **minput_open_im()** (p. 95) の引数 **arg** によって指される構造体.

変数

- Display ∗ **display**
- XrmDatabase **db**
- char ∗ **res_class**
- char ∗ **res_name**
- char ∗ **locale**
- char ∗ **modifier_list**

3.22.1 説明

関数 **minput_open_im()** (p. 95) の引数 **arg** によって指される構造体.

MInputXIMArgIM (p. 191) 型は、関数 **minput_open_im()** (p. 95) が名前 **Mxim** (p. 147) を持つ外部入力メソッドを生成する際に引数 **arg** によって指される構造体である。

3.22.2 構造体

3.22.2.1 Display∗ MInputXIMArgIM::display

以下の４つのメンバの意味は、XOpenIM() の引数の意味と同じである.

クライアントのディスプレイ.

3.22.2.2 XrmDatabase MInputXIMArgIM::db

X リソース・データベースへのポインタ.

3.22.2.3 char∗ MInputXIMArgIM::res_class

アプリケーションの完全なクラス名.

3.22.2.4 char∗ MInputXIMArgIM::res_name

アプリケーションの完全なリソース名.

3.22.2.5 char∗ MInputXIMArgIM::locale

XIM がオープンされたロケール名.

3.22.2.6 char∗ MInputXIMArgIM::modifier_list

XSetLocaleModifiers() の引数.

Appendix A

m17n ライブラリのコンパイル・リンクオプションの表示

A.1 SYNOPSIS

m17n-config [API-LEVEL ...] [–cflags | –libs | –libtool] [–version]

A.2 DESCRIPTION

The shell script m17n-config prints compile and link options for a program that uses the m17n library.

By default, the printed options are for such a program that uses SHELL API of the libray. But, if the first argument is "CORE", "GUI", or "FLT", the options are for a program that uses the corresponding API.

The other arguments are as follows.

- –cflags
  Print compile option (e.g. -I/usr/local/include)
- –libs
  Print link option (e.g. -L/usr/local/lib -lm17n)
- –libtool
  Print libtool option (e.g. /usr/local/lib/libm17n.la)
- –version
  Print version number of the m17n library.

Appendix B

m17n データベースの情報を表示

B.1 SYNOPSIS

m17n-db [OPTIONS] [TAG0 [TAG1 [TAG2 [TAG3]]]]

B.2 DESCRIPTION

The shell script m17n-db prints information about the m17n database.

The arguments OPTIONS has the following meanings.

- -h, –help
  Print this information.
- -v, –version
  Print the version number.
- -l, –locate
  Print absolute pathnames of database files.
  TAG0 through TAG3 specifies the tags of the database.

With no arguments, print where the m17n database is installed.

Appendix C

サンプルプログラム

ここでは以下のサンプルプログラムを説明する。これらのプログラム m17n ラ イブラリの使い方を説明するものであり、実際の使用を意図したものではない。

- **m17n-conv** (p. 198) – ファイルのコードを変換する
- **m17n-view** (p. 199) – ファイルを見る
- **m17n-date** (p. 199) – 日時を表示する
- **m17n-dump** (p. 199) – テキスト画像のダンプ
- **m17n-edit** (p. 201) – 多言語テキストの編集
- **mimx-anthy** (p. 201) – 入力メソッド <ja, anthy> 用外部モジュール.
- **mimx-ispell** (p. 202) – 入力メソッド <en, ispell> 用外部モジュール

C.1 m17n-conv – ファイルのコードを変換する

C.1.1 SYNOPSIS

m17n-conv [OPTION ...] [INFILE [OUTFILE]]

C.1.2 説明

与えられたファイルのコードを別のものに変換する。
INFILE が省略された場合は、標準入力からとる。OUTFILE が省略された 場合は、標準出力へ書き出す。
以下のオプションが利用できる。

- -f FROMCODE
  FROMCODE は INFILE のコード系である。(デフォルトは UTF-8)
- -t TOCODE
  TOCODE は OUTFILE のコード系である。(デフォルトは UTF-8)
- -k
  エラーで変換を停止しない。
- -s
  警告を表示しない。
- -v
  進行状況を表示する。
- -l
  利用可能なコード系を列挙する。
- –version
  バージョン番号を表示する。
- -h, –help
  このメッセージを表示する。

C.2 m17n-view – ファイルを見る

C.2.1 SYNOPSIS

m17n-view [XT-OPTION ...] [OPTION ...] [FILE]

C.2.2 DESCRIPTION

FILE をウィンドウに表示する。

FILE が省略された場合は、標準入力からとる。

XT-OPTIONs は Xt の標準の引数である。 (e.g. -fn, -fg).

以下のオプションが利用できる。

- -e ENCODING
  ENCODING は FILE のコード系である。(デフォルトは UTF-8)
- -s FONTSIZE
  FONTSIZE はフォントの大きさをポイント単位で示したものである。省略 された場合は、X のリソースで定義されたデフォルトフォントの大きさと なる。
- –version
  バージョン番号を表示する。
- -h, –help
  このメッセージを表示する。

C.3 m17n-date – 日時を表示する

C.3.1 シノプシス

m17n-date [OPTION ...]

C.3.2 説明

システムの日時をさまざまなロケールでウィンドウに表示する。

以下のオプションが利用できる。

- –version
  バージョン番号を表示する。
- -h, –help
  このメッセージを表示する。

C.4 m17n-dump – テキスト画像のダンプ

C.4.1 SYNOPSIS

m17n-dump [OPTION ...] [FILE]

C.4.2 DESCRIPTION

テキストを PNG 画像としてダンプする。

PNG 画像は現在のディレクトリに作られた "BASE.png" という名前の ファイルに書き込まれる。ここで BASE は FILE の basename である。 FILE が省略されれば、テキストは標準入力から読まれ、画像は "output.png" にダンプされる。

以下のオプションが利用できる。

- -s SIZE

  SIZE はフォントの大きさをポイント単位で示したものである。デフォル トの大きさは 12 ポイント。

- -d DPI

  DPI は解像度を 1 インチあたりのドット単位で示したものである。デフォ ルトの解像度は 300 dpi。

- -p PAPER

  PAPER はペーパサイズ : a4, a4r, a5, a5r, b5, b5r, letter, WxH または W。 WxH の場合、 W と H は幅と高さをミリメータ単位で示したもの。 W の場合、 W は幅をミリメータ単位で示したもの。このオプションが指定 されている場合は、 PAPER が画像サイズを制限す る。FILE が 1 ページ に納まらないほど大きい場合は、"BASE.01.png", "BASE.02.png" 等の名前 のついた複数のファイルが作られる。

- -m MARGIN

  MARGIN は水平、垂直マージンをミリメータ単位で示したものである。デ フォルトのマージンは 20 mm。PAPER が指定されていなければ無視される。

- -c POS

  POS はカーソルの文字位置。デフォルトでは、カーソルは描かれない。

- -x

  FILE は m17n ライブラリのシリアライズ機能によって作られた XML ファ イルであり、画像を生成する前にデシリアライズされる。

- -w

  語の境界で改行する。

- -f FILTER

  FILTER はシェルコマンド行を含む文字列である。このオプションが指定 されていれば、PNG 画像はファイルに書かれるのではなく、FILTER に 標準入力として渡される。 FILTER が "%s" を含んでいれば、それは FILE のベースネームに置き換えられる。このプログラムのデフォルトの振舞い と、FILTER に "cat > %s.png" を指定した場合の振舞は同一である。

  もし FILTER が単に "-" であれば、 PNG 画像は stdout に出力される。

- -a

  アンチエイリアス処理を行う。

- –family FAMILY

  ファミリィ名が FAMILY のフォントを優先的に使う。

- –language LANG

  言語 LANG 用に指定されたフォントを優先的に使う。LANG は ISO 630 の 2 文字コード（例：英語は "en" ）でなければならない。

- -fg FOREGROUND

  テキストの色を指定する。HTML 4.0 の色の名前および "#RRGGBB" 記法を サポート。

- -bg BACKGROUND
  背景の色を指定する。サポートされている色の名前は FOREGROUND 同じ。 ただし、もし "transparent" が指定されたら背景を透明にする。
- -r
  Specify that the orientation of the text is right-to-left.
- -q
  一切のメッセージを表示しない。
- –version
  バージョン番号を表示する。
- -h, –help
  このメッセージを表示する。

C.5 m17n-edit – 多言語テキストの編集

C.5.1 SYNOPSIS

m17n-edit [XT-OPTION ...] [OPTION ...] FILE

C.5.2 DESCRIPTION

FILE をウィンドウに表示し、ユーザが編集できるようにする。

XT-OPTIONs は Xt の標準の引数である。 (e.g. -fn, -fg).

以下のオプションが利用できる。

- –version
  バージョン番号を表示する。
- -h, –help
  このメッセージを表示する。

このプログラムは m17n GUI API の使い方を示すものである。m17n-edit は直接 GUI API を使っているが、この API は主にツールキットライブラ リや XOM (X Output Method) の実装用であり、アプリケーションプログラ ムからの直接の利用を意図していない。

C.6 mimx-anthy – 入力メソッド <ja, anthy> 用外部モジュール.

C.6.1 DESCRIPTION

共有ライブラリ mimx-anthy.so は入力メソッド <ja, anthy> に用いられ る外部モジュールであり、以下の関数を export している。

- init
  モジュールの初期化。

- fini
  モジュールの終了。
- convert
  現在の preedit テキスト (ひらがな列) をかな漢字テキストに変換する。
- change
  現在のセグメントの候補の変遷を記録する。
- resize
  現在のセグメントの長さを変更する。
- commit
  全セグメントの最新の候補をコミットする。

C.6.2 参照

インプットメソッド (p. 215)

C.7 mimx-ispell – 入力メソッド <en, ispell> 用外部モジュール

C.7.1 DESCRIPTION

共有ライブラリ mimx-ispell.so は入力メソッド <en, ispell> に用いら れる外部モジュールであり、以下の関数を export している。

- init
  ライブラリの初期化。
- fini
  ライブラリの終了。
- ispell_word
  現在の preedit テキスト (英文) の綴を調べ、間違っていれば候補のリ ストを返す。

このプログラムは m17n 入力メソッド用外部モジュールの書き方を示すた めのものであり、実際の利用を意図したものではない。

C.7.2 参照

インプットメソッド (p. 215)

Appendix D

M17N データベースのデータ・フォーマット

ここでは、M17N データベースで提供される各データのフォーマットを解説す る。

D.1 一般的なフォーマット

D.1.1 説明

関数 **mdatabase_load()** (p. 61) は、第一タグが `Mchartable` か `Mcharset` で ない限り、タグで指定されたデータをプロパティリストの形で返す。返される プロパティリストのキーは `Minteger`, `Msymbol`, `Mtext`, `Mplist` に限る。値の型はキーによって曖昧性なく 決定される。キーが `Minteger` ならば値は整数値であり、キーが `Msymbol` なら値はシンボル、`Mtext` ならば M-text、 `Mplist` ならばプロパティリストである。

プロパティリストを示すためにはさまざまな表現形式が可能である。たとえば `(K1:V1, K2:V2, ..., Kn:Vn)` という形で、最初の K1 と V1、二つ目 のキーと値が K2 と V2 といったプロパティリストを表現することもできる。 しかし、m17n データベースで用いられるプロパティリストのタイプはかなり 限定されているので、もっと単純な表現形式を用いることができる。

これ以降、プロパティリストの示すためには S 式に類似した表現を用いる。 (実際 m17n ライブラリのデフォルトのデータベースローダはこの表現で書か れたデータファイルを読むように設計されている。)

この表現形式は 1 ないしそれ以上の **要素** *(element)* からなる。個々 の要素はプロパティすなわちプロパティリストの一つの要素を表す。

要素間は 1 ないしそれ以上の **空白** *(whitespace)* すなわちスペース (code 32)、タブ (code 9)、改行 (code 10) のいずれかで区切られる。セミ コロン (`;`) で始まる行はコメントである。

各プロパティのキーと値は、要素のタイプに基づいて以下のように決定される。

- 整数

  正規表現 `-?[0-9]+` または `0[xX][0-9A-Fa-f]+` に合致する 要素は、キーが `Minteger` であるプロパティを示す。最初の正規表現 と合致するものは 10 進の整数、後の正規表現と合致するものは 16 進の整数 と解釈し、その解釈の結果がプロパティの値である。

  たとえば、要素 `0xA0` は 10 進で 160 を値とするプロパティである。

- シンボル

  正規表現 `[^-0-9(][^\()]|\.)+` に合致する要素は、キーが `Msymbol` であるプロパティを示す。要素中の `\t`, `\n`, `\r`, `\e` はそれぞれ、タブ (code 9)、改 行 (code 10)、復帰 (code 13)、エスケープ (code 27) に置き換えられる。バッ クスラッシュに続く他の文字はそれ自身と解釈される。プロパティの値は、結 果として得られる文字列を名前として持つシンボルである。

  たとえば、要素 `abc\ def` は "abc def" を名前として持つシンボル を値とするプロパティである。

- MTEXT

  正規表現 "([^"]|\")*" に合致する要素は、キー Mtext であるプロパティを示す。上記のバックスラッシュによるエスケープはここで も有効である。さらに、要素中の正規表現 \[xX][0-9A-Fa-f][0-9A-Fa-f] に合致する部分は、16 進で解釈した 結果に置き換えられる。

  バックスラッシュエスケープを処理した上で、ダブルクオートにはさまれたバ イト列を UTF-8 列として解釈し、M-text にデコードする。この M-text がプ ロパティの値である。

- PLIST

  対応する括弧にはさまれた 0 個以上の要素は Mplist をキーとする プロパティを示す。括弧の前後の空白は取り除かれる。プロパティの値は、プ ロパティリストであり、その括弧内の各要素を再帰的に解釈した結果である。

D.1.2 文法の表記

データのプロパティリストフォーマットの説明では、BNF 風の記法が用いられ る。この記法では、非終端は大文字（間に'-' が入ってもよい） で、終端は " で囲って表される。特別な非終端 INTEGER, SYMBOL, MTEXT, PLIST はそれぞれ対応するプロパティを意味する。

D.1.3 例

次の単純な形式のプロパティリストに読み込まれる、データベースのデータの 一例を示す：

```
DATA-FORMAT ::=
    [ INTEGER | SYMBOL | MTEXT | FUNC ] *

FUNC ::=
    '(' FUNC-NAME FUNC-ARG * ')'

FUNC-NAME ::=
    SYMBOL

FUNC-ARG ::=
    INTEGER | SYMBOL | MTEXT | '(' FUNC-ARG ')'
```

たとえば、次のテキストを含むデータファイルは上の文法に合致する：

```
abc 123 (pqr 0xff) "m\"text" (_\\_ ("string" xyz) -456)
```

そして次のようなプロパティリストとして読み込まれる：

```
第1要素: キー: Msymbol,  値: abc
第2要素: キー: Minteger, 値: 123
第3要素: キー: Mplist,   値: 次の要素からなるプロパティリスト
    第1要素: キー Msymbol,  値: pgr
    第2要素: キー Minteger, 値: 255
第4要素: キー: Mtext,    値: m"text
第5要素: キー: Mplist,   値: 次の要素からなるプロパティリスト
    第1要素: キー: Msymbol, 値: _\_
    第2要素: キー: Mplist,  値: 次の要素からなるプロパティリスト
        第1要素: キー: Mtext,    値: string
第2要素: キー: Msymbol,  値: xyz
第3要素: キー: Minteger, 値: -456
```

D.2 文字セット定義のリスト

D.2.1 説明

m17n ライブラリは、m17n データベースのタグ <charset-list> のついたデータ から、文字セット定義のリストをロードする。このデータは以下のフォーマッ トのプロパティリストとしてロードされる。

```
CHARSET-LIST ::= DEFINITION *

DEFINITION ::= '(' NAME ( KEY VALUE ) * ')'

NAME ::= SYMBOL

KEY ::= SYMBOL

VALUE ::= SYMBOL | INTEGER | MTEXT | PLIST
```

NAME は定義する文字セットの名前である。

KEY と VALUE のペアは、関数 **mchar_define_charset()** (p. 65) に 2 番目の引 数 **plist** の各要素として与えられるプロパティである。

D.2.2 参照

mdbGeneral(5) (p. 204), **mchar_define_charset()** (p. 65)

D.3 コード系定義のリスト

D.3.1 説明

m17n ライブラリは、初期化の際 m17n データベースのタグ <coding-list> のついたデータからコード系定義のリストをロードする。このデータは以下の フォーマットのプロパティリストとしてロードされる。

```
CODING-LIST ::= DEFINITION *

DEFINITION ::= '(' NAME ( KEY VALUE ) * ')'
NAME ::= SYMBOL

KEY ::= SYMBOL

VALUE ::= SYMBOL | INTEGER | MTEXT | PLIST
```

NAME は定義するコード系の名前である。

KEY と VALUE のペアは、関数 mchar_define_coding() に 2 番目の引 数として与えられるプロパティである。

D.3.2 参照

mdbGeneral(5) (p. 204), **mconv_define_coding()** (p. 76)

D.4 データベースディレクトリ中のデータのリスト

D.4.1 説明

m17n ライブラリは初期化の際、m17n データベース中のデータ定義のリストを 各データベースディレクトリ中の "mdb.dir" という名前を持つファイルからロー ドする。このファイルにおけるプロパティリストのフォーマットは以下である。

```
MDB-DIR ::= DEFINITION *

DEFINITION ::= '(' TAG0 [ TAG1 [ TAG2 [ TAG3 ] ] ] FILE [ VERSION ]')'

TAGn ::= SYMBOL

FILE ::= MTEXT

VERSION ::= MTEXT
```

TAG0 が 'charset' でも 'char-table' でもなく、TAGn (n > 0) がシンボル '*' ならば、FILE にワイルドカードが含まれて良い。そしてシェルが用いる規 則によって FILE とマッチするすべてのファイルがデータベースファイルのター ゲットとなる。この際、各ファイルは実際の TAGn の値を与える SELF-DEFINITION を持たなくてはならない。SELF-DEFINITION は以下の形式の プロパティリストの要素である。

```
SELF-DEFINITION ::= '(' TAG0 TAG1 TAG2 TAG3 [ VERSION ] ')'
```

たとえば、データベースディレクトリが下のファイルを含むとしよう。

```
zh-py.mim:
(input-method zh py)
...

ko-han2.mim:
(input-method ko han2)
...
```

この時 "mdb.dir" 中の以下の行

```
(input-method zh py "zh-py.mim")
(input-method ko han2 "ko-han2.mim")
```

はこの一行に短縮できる。

```
(input-method * "*.mim")
```

VERSION は 最低必要な m17n ライブラリのバージョン番号を示す。フォー マットは "XX.YY.ZZ" であり、XX はメジャーバージョン番号、YY はマイナー バージョン番号、ZZ はパッチレベルである。

D.5 フォントレイアウトテーブル

D.5.1 説明

単純なスクリプトの場合、表示エンジンは選択したフォントのエンコーディン グに応じて文字コードをグリフコードに一文字ずつ変換する。しかし、複雑な レイアウトを要求する文書、たとえばタイやインド系のスクリプトなどの場合、 1対1の変換では不十分である。複数の文字が一つのリガチャとして描かれたり、 2次元的にずらした位置に描かなくてはならないグリフがあったりする。

このような複雑なスクリプトを処理するため、m17n ライブラリはフォントレ イアウトテーブル (短縮して FLT と呼ぶことにする) を用いる。FLT ドライ バは FLT を解釈し、文字列を表示エンジンに渡すことのできるグリフ列に変 換する。

FLT は OpenType Layout Table に見られる情報 (CMAP, GSUB, and GPOS) に 加えて、文字列から書記素 (grapheme) クラスタを抽出したり、クラスタ内で 文字を並べ変えたりするための情報を持つことができる。

FLT は 1 つ以上の変換ステージが続いたものである。各ステージでコード列 は別のものに変換され、次のステージに読まれる。列の長さはステージ毎に異 なることがある。コード列の各要素は以下の整数値の属性を持つ。

- コード

  変換の最初のステージでは、元の文字列の文字コード。最後のステージでは、 表示エンジンに渡されるグリフコード。それ以外では中間的なグリフコード。

- カテゴリ

  そのステージの CATEGORY-TABLE で定義されたか、以前のステージで定義さ れ上書きされていないカテゴリーコード。

- 結合規則

  0 でなければ、この（中間）グリフを前のものとどう結合するかを指定する。

- 左パディングフラグ

  0 でなければ、表示関数にこの（中間）グリフの前にスペースを挿入して、前 のグリフと重ならないようにするよう指示する。

- 右パディングフラグ

  0 でなければ、表示関数にこの（中間）グリフの後にスペースを挿入して、後 のグリフと重ならないようにするよう指示する。

レイアウトエンジンがテキストを描く際には、まずテキストの各文字に対して それぞれフォントと FLT を決定する。同じフォントと FLT を用いる部分文字列 に関して、レイアウトエンジンは対応する中間的なグリフの列を生成する。中 間的なグリフコードの各要素は、コードの属性として対応する文字コード、他 の属性として 0 を持つ。この列は FLT の最初のステージで現行の **ラン** (部 分列) として処理される。

各ステージは以下のように働く。

まずこのステージに CATEGORY-TABLE があれば、現行のランのすべてのグ リフのカテゴリが更新される。カテゴリの無いグリフがあれば、ランはそのグ リフの前で終る。

次にこのステージのコードオフセット、結合規則、左パディングフラグが 0 に初期化される。

次いで、このステージの最初の変換規則が現行のランに適用される。

最後に現行のランは新しく作られた（中間）グリフ列に置き換えられる。

D.5.2 文法と意味

m17n ライブラリ は m17n データベースからタグ <font, layouter, FLT-NAME> を用いて FLT をロードする。FLT のデータのフォーマットは以下の通り：

```
FONT-LAYOUT-TABLE ::= FLT-DECLARATION ? STAGE0 STAGE *

FLT-DECLARATION ::= '(' 'font' 'layouter' NAME nil PROP * ')'
NAME ::= SYMBOL
PROP :: = VERSION | FONT
VERSION ::= '(' 'version' MTEXT ')'
FONT ::= '(' 'font' FONT-SPEC ')'
FONT-SPEC ::=
```

```
     '(' [[ FOUNDRY FAMILY
          [ WEIGHT [ STYLE [ STRETCH [ ADSTYLE ]]]]]
        REGISTRY ]
 [ OTF-SPEC ] [ LANG-SPEC ] ')'

STAGE0 ::= CATEGORY-TABLE GENERATOR

STAGE ::= CATEGORY-TABLE ? GENERATOR

CATEGORY-TABLE ::= '(' 'category' CATEGORY-SPEC + ')'

CATEGORY-SPEC ::= '(' CODE CATEGORY ')'
                | '(' CODE CODE CATEGORY ')'

CODE ::= INTEGER

CATEGORY ::= INTEGER
```

CATEGORY-SPEC の定義中で、 CODE はグリフコード CATEGORY は大 文字あるいは小文字の ASCII code、すなわち 'A', ... 'Z', 'a', .. 'z' の いずれかである。

CATEGORY-SPEC の最初の形式は、CATEGORY をコード CODE を持つグ リフに割り当て、二つ目の形式は CATEGORY を二つの CODE の間のコー ドを持つグリフに割り当てる。

```
GENERATOR ::= '(' 'generator' RULE MACRO-DEF * ')'

RULE ::= REGEXP-BLOCK | MATCH-BLOCK | SUBST-BLOCK | COND-BLOCK
         FONT-FACILITY-BLOCK | DIRECT-CODE | COMBINING-SPEC | OTF-SPEC
         | PREDEFINED-RULE | MACRO-NAME

MACOR-DEF ::= '(' MACRO-NAME RULE + ')'
```

各 RULE は、消費するグリフと生成するグリフを指定する。「消費された」 グリフは現行のランから取り除かれる。ルールは状況によっては失敗する。明 示的に失敗と書かれている場合をのぞき、成功とみなす。

```
DIRECT-CODE ::= INTEGER
```

このルールはグリフを消費せず、以下の属性を持つグリフを生成する。

- コード : INTEGER にデフォルトのコードオフセットを足したもの
- 結合規則 : デフォルト値
- 左パデイングフラグ : デフォルト値
- 右パデイングフラグ : 0

グリフ生成後、デフォルトのコードオフセット、結合規則, 左パディングフラ グ はすべて 0 にリセットされる。

```
PREDEFINED-RULE ::= '=' | '*' | '<' | '>' | '|' | '[' | ']'
```

これらは以下のように働く。

- =
  現行のランの最初のグリフを消費し、同じグリフを生成する。現行のランが空 ならば失敗する。
- *
  前のルールを繰り返し実行する。前のルールが失敗すれば、何もせず失敗する。

- <
  書記素クラスタの始めを示す。
- >
  書記素クラスタの終りを示す。
- @[
  この規則はデフォルトの左パディングフラグを 1 にする。グリフの消費や生 成はしない。
- @]
  この規則は最近生成されたグリフの右パディングフラグを 1 にする。グリフ の消費や生成はしない。
- |
  グリフを消費せず、カテゴリが ' ' で他の属性が 0 である特別なグリフを生 成する。この規則だけがこの特別なグリフを生成する。

```
REGEXP-BLOCK ::= '(' REGEXP RULE * ')'

REGEXP ::= MTEXT
```

MTEXT は現行のランのカテゴリ列に合致すべき正規表現である。合致すれ ば、この規則は一時的に現行のランを合致した部分だけに限定した上で、RULE を実行する。合致した部分はこの規則によって消費される。

括弧のついた部分表現があれば、RULE の中に出現するかもしれない MATCH-BLOCK によって使用するために記録される。

合致する部分が無ければ、この規則は失敗する。

```
MATCH-BLOCK ::= '(' MATCH-INDEX RULE * ')'

MATCH-INDEX ::= INTEGER
```

MATCH-INDEX は直前の REGEXP-BLOCK によって記録された部分表現を指 定する整数である。このような部分表現があれば、この規則は一時的に現行の ランを合致した部分表現だけに限定した上で、RULE を実行する。合致した 部分はこの規則によって消費される。

合致する部分が無ければ、この規則は失敗する。

この規則がステージの最初の規則である場合は、MATCH-INDEX は 0 でなく てはならない。この場合ラン全体に合致することになる。

```
SUBST-BLOCK ::= '(' SOURCE-PATTERN RULE * ')'

SOURCE-PATTERN ::= '(' CODE + ')'
                 | ('range' CODE CODE ')'
```

現行のランのコード列が SOURCE-PATTERN と合致すれば、この規則は一時 的に現行のランを合致した部分だけに限定した上で、RULE を実行する。 合致した部分は消費される。

SOURCE-PATTERN の最初の形式は、合致するグリフコードの列を指定する。 この場合、この規則はデフォルトのコードオフセットを 0 にリセットする。

二つめの形式は、コード列の最初のグリフコードの範囲を指定する。この場合、 この規則はデフォルトのコードオフセットを最初のグリフコードから範囲を指 定する初めの CODE を引いたものに設定する。

合致する部分が無ければ、この規則は失敗する。

```
FONT-FACILITY-BLOCK ::= '(' FONT-FACILITY RULE * ')'
FONT-FACILITY = '(' 'font-facility' CODE * ')'
       | '(' 'font-facility' FONT-SPEC ')'
```

現在のフォントが CODE のグリフを持っているか、FONT-SPEC と合致すれば、この規則は成功し、RULE を実行する。そうでなければ、この規則は失敗する。

```
COND-BLOCK ::= '(' 'cond' RULE + ')'
```

この規則は RULE を順に、どれかが成功するまで実行する。どのルールも成功しなければ、この規則は失敗する。そうでなければ成功である。

```
OTF-SPEC ::= SYMBOL
```

OTF-SPEC は、OTF ドライバへの指示を指定する名前を持つシンボルである。名前は以下の文法に従う。

```
OTF-SPEC-NAME ::= ':otf' SCRIPT LANGSYS ? GSUB-FEATURES ? GPOS-FEATURES ?

SCRIPT ::= SYMBOL

LANGSYS ::= '/' SYMBOL

GSUB-FEATURES ::= '=' FEATURE-LIST ?

GPOS-FEATURES ::= '+' FEATURE-LIST ?

FEATURE-LIST ::= ( SYMBOL ',' ) * [ SYMBOL | '*' ]
```

各 SYMBOL は OpenType specification 中でのタグ名を指定する。

SCRIPT については、SYMBOL はスクリプトタグ名を表す。(Devanagari は deva など。)

LANGSYS の場合は、SYMBOL は言語システムタグ名を指定する。LANGSYS が省略されれば、デフォルトの言語システムテーブルが使用される。

GSUB-FEATURES では、FEATURE LIST 中の各 SYMBOL は適用する GSUB feature タグ名を指定する。'*' は残りすべての feature を指定するために最後の要素として用いることができる。SYMBOL の前に'~' がついており、最後の要素が '*' ならば、SYMBOL は適用する feature から除かれる。SYMBOL が指定されていなければ、GSUB feature は適用されない。GSUB-FEATURES 自体が省略されればすべての GSUB feature が適用される。

GPOS-FEATURES の指定は GSUB-FEATURES の場合と同様である。

全てのタグ名は ASCII の表示可能文字４つからなること。

OpenType の指定方法については次のページを参照のこと。

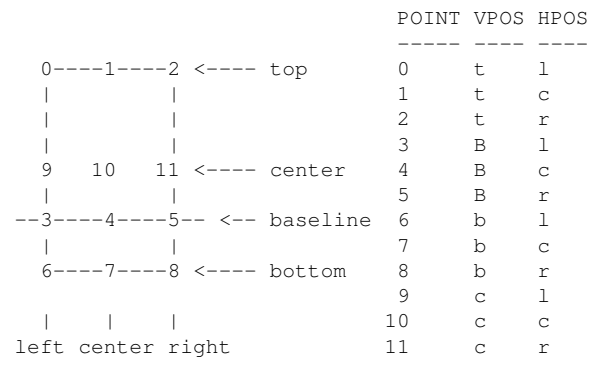
<http://www.microsoft.com/typography/otspec/default.htm>

```
COMBINING ::= SYMBOL
```

COMBINING は、次のグリフを前のものとどう結合するかの指示を名前として持つシンボルである。このルールはデフォルトの結合規則をシンボル名固有の整数コードにセットする。名前は以下の文法に従う。following syntax.

```
COMBINING-NAME ::= VPOS HPOS OFFSET VPOS HPOS

VPOS ::= 't' | 'c' | 'b' | 'B'

HPOS ::= 'l' | 'c' | 'r'

OFFSET :: = '.' | XOFF | YOFF XOFF ?

XOFF ::= ('<' | '>') INTEGER ?

YOFF ::= ('+' | '-') INTEGER ?
```

VPOS と HPOS は次のように垂直、水平位置を指定する。



POINT	VPOS	HPOS
0	t	l
1	t	c
2	t	r
3	B	l
4	B	c
5	B	r
6	b	l
7	b	c
8	b	r
9	c	l
10	c	c
11	c	r

左の図はあるグリフの 12 の参照点を 0 から 11 までの数字で示している。 四角形 0-6-8-2 はグリフの表示領域であり、位置 3, 4, 5 はベースライン上 にある。 9 と 11 はそれぞれ線 0-6 と 2-8 の中心である。1, 10, 4, 7 はそ れぞれ線 1-2, 3-5, 9-11, 6-8 の中心である。

右の表は、各参照点が VPOS と HPOS の組合せによってどのように指定 されるかを示している。

COMBINING-NAME の定義中の最初の VPOS と HPOS は、前のグリフの 参照点を、二つ目の VPOS と HPOS は次のグリフの参照点を指定する。 次のグリフはこの二個の参照点が重なるように描かれる。

OFFSET は重なりか他の詳細を指定する。'.' であれば、参照点二つは同じ 位置にある。

XOFF は、次のグリフの参照点の X 座標を、前の参照点からどれほど右 ('<') あるいは左 ('>') へずらすか指定する。

YOFF は、次のグリフの参照点の Y 座標を、前の参照点からどれほど上 ('+') あるいは下 ('-') へずらすか指定する。

どちらの場合にも、INTEGER はフォントサイズの何%ずらすかを示す値で ある。すなわち、もし INTEGER が 10 ならばフォントサイズの 10% (1/10) ずらすことになる。 INTEGER が省略された場合には、5 が指定されたもの とする。

次のグリフが前のグリフに結合されると、それらは一つの結合グリフとして扱 われる。

```
MACRO-NAME ::= SYMBOL
```

MACRO-NAME は MACRO-DEF のいずれかに現われるシンボルであり、対応 する RULE の列に展開される。

D.5.3 文脈に依存する振舞

ここまでは、特定のフォントで描かれる各文字／グリフ列が文脈自由であるこ と、すなわち前後のグリフに影響されないことを前提としてきた。これは、列 S1 がフォント F1 によって描かれ、先行する列 S0 が常にフォント F0 を要求 する場合には正しい。

列	S0	S1
現行のフォント	F0	F1
利用可能なフォント	F0	F1

しかし時には、列を明確に区切ることができない場合もある。先行する列 S0 が F0 だけでなく F1 でも描けるとしよう。

列	S0	S1
現行のフォント	F0	F1
利用可能なフォント	F0,F1	F1

この場合、先行する S0 を描くために使われたグリフが S1 のグリフの生成に 影響を与えることもある。そこで S1 の処理の際にすでに処理の終った S0 に 関する情報にアクセスする必要がある。最初のステージ（このステージのみ） の生成規則は、処理済みの部分へアクセスする特別な正規表現を許している。

```
"RE0 RE1"
```

RE0 と RE1 は先行の列 S0 と後続の列 S1 にそれぞれ対応する正規表 現である。

二つの正規表現の間のスペースに注意。これは特別なカテゴリ ' ' を示して いる (上記参照) 。この正規表現はフォント F1 を使用するグリフ生成規則に 属しており、したがって RE1 だけでなく RE0 も F1 用のカテゴリを用いて表 現されなければならない。つまり、先行する列 S0 が F1 用のカテゴリで表せ ない場合には (上の最初の例のように) このパターンを持つ生成規則には合致 しない。

D.5.4 参照

mdbGeneral(5) (p. 204), **FLTs provided by the m17n database** (p. **??**)

D.6 フォントエンコーディング

D.6.1 説明

m17n ライブラリは、m17n データベースから <font, encoding> タグによっ て個々のフォントのエンコーディングに関する情報をロードする。このデータ は以下のフォーマットのプロパティリストとしてロードされる。

```
FONT-ENCODING ::= PER-FONT *

PER-FONT ::= '(' FONT-SPEC ENCODING [ REPERTORY ] ')'

FONT-SPEC ::=
    '(' [ FOUNDRY FAMILY
        [ WEIGHT [ STYLE [ STRETCH [ ADSTYLE ]]]]]
REGISTRY ')'

ENCODING ::= SYMBOL
```

FONT-SPEC はフォントのプロパティを指定する。FOUNDRY から REGISTRY はフォントの **Mfoundry** (p. 122) から **Mregistry** (p. 123) プロパティに対応するシ ンボルである。各プロパティの意味については **フォント** (p. 115) 参照。

たとえばこの FONT-SPEC:

```
    (nil alice0\ lao iso8859-1)
```

はフォントのファミリ名が "alice0 lao" でレジストリが "iso8859-1" であ るすべてのフォントに適用できる。

ENCODING は文字セットを示すシンボルである。FONT-SPEC に合致する フォントは、その文字セットの全文字をサポートし、その文字セットによって 文字コードはそのフォントの対応するグリフコードにマップされる。

REPERTORY は文字セットを示すシンボルか "nil" である。省略した場合は、 ENCODING を REPERTORY に指定したのと同じ意味になる。"nil" でなけ れば、文字セットはフォントのレパートリ、すなわちサポートする文字を示す。 そうでなければ、特定の文字がそのフォントでサポートされているかどうかは 個々のフォントドライバに問い合わせる。

いわゆるユニコードフォント (レジストリは "iso10646-1") については、普 通ユニコード文字の一部しかサポートしていないため、REPERTORY を "nil" にすることが望ましい。

D.7 Font Size

D.7.1 DESCRIPTION

In some case, a font contains incorrect information about its size (typically in the case of a hacked TrueType font), which results in a bad text layout when such a font is used in combination with the other fonts. To overcome this problem, the m17n library loads information about font-size adjustment from the m17n database by the tags <font, resize>. The data is loaded as a plist of this format.

```
FONT-SIZE-ADJUSTMENT ::= PER-FONT *

PER-FONT ::= '(' FONT-SPEC ADJUST-RATIO ')'

FONT-SPEC ::=
    '(' [ FOUNDRY FAMILY
       [ WEIGHT [ STYLE [ STRETCH [ ADSTYLE ]]]]]
REGISTRY ')'

ADJUST-RATIO ::= INTEGER
```

`FONT-SPEC` is to specify properties of a font. `FOUNDRY` to `REGISTRY` are symbols corresponding to **Mfoundry** (p. 122) to **Mregistry** (p. 123) property of a font. See **フォント** (p. 115) for the meaning of each property.

`ADJUST-RATIO` is an integer number specifying by percentage how much the font-size must be adjusted. For instance, this `PER-FONT`:

```
    ((devanagari-cdac) 150)
```

instructs the font handler of the m17n library to open a font of 1.5 times bigger than a requested size on opening a font whose registry is "devanagari-cdac".

D.8 フォントセット

D.8.1 説明

m17n ライブラリは、m17n データベースから <fontset, FONTSET-NAME> タ グによってフォントセットの定義をロードする。このデータは以下のフォーマッ トのプロパティリストとしてロードされる。

```
FONTSET ::= PER-SCRIPT * PER-CHARSET * FALLBACK *

PER-SCRIPT ::= '(' SCRIPT PER-LANGUAGE + ')'

PER-LANGUAGE ::= '(' LANGUAGE FONT-SPEC-ELEMENT + ')'

PER-CHARSET ::= '(' CHARSET FONT-SPEC-ELEMENT + ')'

FALLBACK ::= FONT-SPEC-ELEMENT

FONT-SPEC-ELEMENT ::= '(' FONT-SPEC [ FLT-NAME ] ')'

FONT-SPEC ::=
     '(' [ FOUNDRY FAMILY
           [ WEIGHT [ STYLE [ STRETCH [ ADSTYLE ]]]]]
         REGISTRY ')'
```

`SCRIPT` はスクリプト名 (e.g. latin, han) を示すシンボルか `nil` であ る。`LANGUAGE` は ISO 639 に定義された言語名コード (e.g. ja, zh) であ る 2 文字のシンボルか `nil` である。

FONT-SPEC はフォントのプロパティを指定する。FOUNDRY から REGISTRY はフォントの **Mfoundry** (p. 122) から **Mregistry** (p. 123) プロパティに対応するシ ンボルである。各プロパティの意味については **フォント** (p. 115) 参照。

FLT-NAME はフォントレイアウトテーブルの名前である。(**フォントレイアウトテーブル** (p. 207)).

D.8.2 例

これは PER_SCRIPT の例である。

```
(han
  (ja
    ((jisx0208.1983-0)))
  (zh
    ((gb2312.1980-0)))
  (nil
    ((big5-0))))
```

これによってフォントセレクタは、"han" 文字 (つまり **Mscript** (p. 26) プロパティ が 'han' である文字) のうち、文字の M-text 中での **Mlanguage** (p. 47) テキストプ ロパティが "ja" でありその文字がフォントのレパートリーに含まれていれも のについては、レジストリが"jisx0208.1983-0" であるフォントを使うことを指示される。そうでなければ、レジストリが "gb2312.1980-0" や"big5-0" で あるものが試される。"han" 文字に **Mlanguage** (p. 47) テキストプロパティが無けれ ば、３つとも試される。

フォント選択の詳細については関数 **mdraw_text()** (p. 141) 参照。

D.9 インプットメソッド

D.9.1 説明

m17n ライブラリは、m17n データベースから動的にロードできる入力メソッド ドライバを提供している。(**入力メソッド (基本部分)** (p. 91) 参照。(P.91)).

ここでは入力メソッド定義のデータフォーマットを説明する。

D.9.2 文法と意味

以下のデータフォーマットによって入力メソッドが定義される。ドライバはファ イルやストリームから定義をロードし、プロパティリストの形式に変換する。

```
INPUT-METHOD ::=
    IM-DECLARATION ? DESCRIPTION ? TITLE ?
     VARIABLE-LIST ? COMMAND-LIST ?  MODULE-LIST ?
     MACRO-LIST ? MAP-LIST ? STATE-LIST ?

IM-DECLARATION ::= '(' 'input-method' LANGUAGE NAME EXTRA-ID ? VERSION ? ')'
VERSION ::= '(' 'version' VERSION-NUMBER ')'
DESCRIPTION ::= '(' 'description' [ MTEXT-OR-GETTEXT | nil] ')'
VARIABLE-LIST ::= '(' 'variable' VARIABLE-DECLARATION * ')'
COMMAND-LIST ::= '(' 'command' COMMAND-DECLARATION * ')'
TITLE ::= '(' 'title' TITLE-TEXT ')'

VARIABLE-DECLARATION ::=
    '(' VAR-NAME [ MTEXT-OR-GETTEXT | nil ] VALUE VALUE-CANDIDATE * ')'

COMMAND-DECLARATION ::=
    '(' CMD-NAME [ MTEXT-OR-GETTEXT | nil ] KEYSEQ * ')'
```

```
MTEXT-OR-GETTEXT ::=
    [ MTEXT | '(' '_' MTEXT ')']

LANGUAGE ::= SYMBOL
NAME ::= SYMBOL
EXTRA-ID ::= SYMBOL
VERSION ::= MTEXT
IM-DESCRIPTION ::= MTEXT
VAR-NAME ::= SYMBOL
VAR-DESCRIPTION ::= MTEXT
VALUE ::= MTEXT | SYMBOL | INTEGER
VALUE-CANDIDATE ::= VALUE | '(' RANGE-FROM RANGE-TO ')'
RANGE-FROM ::= INTEGER
RANGE-TO ::= INTEGER
CMD-NAME ::= SYMBOL
CMD-DESCRIPTION ::= MTEXT
TITLE-TEXT ::= MTEXT
```

IM-DECLARATION はこの入力メソッドの言語と名前を指定する。

LANGUAGE が t の場合、この入力メソッドは複数の言語で利用される。

NAME が nil の場合、この入力メソッドは単独で用いられるものではなく、他の入力メソッドから利用することを想定している。この場合、入力メソッドを特定するために EXTRA-ID が必要である。

VERSION はこの入力メソッドが必要とする m17n ライブラリの最小バージョンを指定する。フォーマットは "XX.YY.ZZ" であり、XX はメジャーバージョン、YY はマイナーバージョン、ZZ はパッチレベルを表す。

DESCRIPTION はこの入力メソッドの説明を MTEXT-OR-GETTEXT で指定する。もしこれが 2 番目の形式を取っていれば、MTEXT は現在のロケールに従って "gettext" によって翻訳される（翻訳文が提供されている場合）。

TITLE-TEXT はこの入力メソッドが有効な時、スクリーン上に表示されるテキストである。

"global.mim" という特別なファイルがあり、共通する変数やコマンドが定義されている。入力メソッドドライバは毎回このファイルをロードし、他の入力メソッドはここで定義された変数やコマンドを継承できる。

VARIABLE-DECLARATION はこの入力メソッドで使用される変数を宣言する。変数をデフォルト値に初期化したり、ユーザがカスタマイズしたりする場合には、ここで宣言されなくてはならない。この宣言は二通りに利用される。一つめは新しい変数を導入するためであり、この場合 VALUE は省略できない。もう一つは "global.mim" で宣言された変数を継承し、違ったデフォルト値を指定したり、この入力メソッド用にカスタマイズ可能にしたりするためである。この場合には VALUE は省略できる。

COMMAND-DECLARATION はこの入力メソッドで使用されるコマンドを宣言する。コマンドをデフォルトキーシーケンスに割り当てたり、ユーザがカスタマイズしたりする場合には、ここで宣言されなくてはならない。VARIABLE-DECLARATION 同様、この宣言は二通りに利用される。一つめは新しいコマンドを導入するためであり、この場合 KEYSEQ は省略できない。もう一つは "global.mim" で宣言されたコマンドを継承し、違ったキーバインディングを指定したり、この入力メソッド用にカスタマイズ可能にしたりするためである。この場合には KEYSEQ は省略できる。

```
MODULE-LIST ::= '(' 'module' MODULE * ')'

MODULE ::= '(' MODULE-NAME FUNCTION * ')'

MODULE-NAME ::= SYMBOL

FUNCTION ::= SYMBOL
```

各 MODULE は外部モジュール (動的ライブラリ) の名前とそのモジュールが公開している関数名を宣言する。FUNCTION が "init" という名前であれば、この入力メソッド用の入力コンテクストが生成される際に、デフォルトの引数 (CALL の節参照) のみとともに呼ばれる。FUNCTION が "fini" という名前を持てば, 入力コンテクストが破壊される際に、デフォルトの引数のみとともに呼ばれる。

```
MACRO-LIST ::=  MACRO-INCLUSION ? '(' 'macro' MACRO * ')' MACRO-INCLUSION ?

MACRO ::= '(' MACRO-NAME MACRO-ACTION * ')'

MACRO-NAME ::= SYMBOL

MACRO-ACTION ::= ACTION

TAGS ::= `(` LANGUAGE NAME EXTRA-ID ? `)`

MACRO-INCLUSION ::= '(' 'include' TAGS 'macro' MACRO-NAME ? ')'
```

MACRO-INCLUSION は、TAGS で指定される他の入力メソッドからマクロを 読み込む。MACRO-NAME が与えられていなければ、全てのマクロを読む。

```
MAP-LIST ::= MAP-INCLUSION ? '(' 'map' MAP * ')' MAP-INCLUSION ?

MAP ::= '(' MAP-NAME RULE * ')'

MAP-NAME ::= SYMBOL

RULE ::= '(' KEYSEQ MAP-ACTION * ')'

KEYSEQ ::= MTEXT | '(' [ SYMBOL | INTEGER ] * ')'

MAP-INCLUSION ::= '(' 'include' TAGS 'map' MAP-NAME ? ')'
```

入力メソッドがそれ単体で利用されることがなく常に他の入力メソッドに読 み込まれて用いられる場合には、MAP-LIST は省略できる。

MAP-NAME 定義中の SYMBOL は、 t あるいは nil であってはな らない。

KEYSEQ 定義中の MTEXT は、キーボードから生成できる文字で構成され る。すなわち MTEXT は通常 ASCII 文字のみを含む。しかし、入力メソッド がたとえば西ヨーロッパ用キーボードを使うことを想定したものであれば、MTEXT は Latin-1 文字を含んでもよい。

KEYSEQ 定義中の SYMBOL は、関数 **minput_event_to_key()** (p. 147) の戻り値で なくてはならない。X ウィンドウシステムの元では、 xev コマンドを用い て値を簡単にチェックできる。たとえば、 リターンキー、バックスペースキー、 キーパッドの 0 のキーなどは、それぞれ (Return) , (BackSpace) , (KP_0) としてあらわされる。シフト、コントロール、メタ、アルト、スーパー、 ハイパーも押されている場合には、それぞれ S- , C- , M- , A- , s- , H- が 前にこの順に置かれる。したがって "リターンキーをシフトしてメタしてハイ パーしたもの" は (S-M-H-Return) である。"a をシフト" から "z をシフト " までは、単に A から Z として表されることに注意。したがって、 "a をシ フトしてメタしてハイパーしたもの" は (M-H-A) となる。

KEYSEQ 定義中の INTEGER は、有効な文字コードでなくてはならない。

MAP-INCLUSION は、TAGS で指定される他の入力メソッドからマップを読 み込む。MAP-NAME が与えられていなければ、全てのマップを読む。

```
MAP-ACTION ::= ACTION

ACTION ::= INSERT | DELETE | SELECT | MOVE | MARK
           | SHOW | HIDE | PUSHBACK | POP | UNDO
   | COMMIT | UNHANDLE | SHIFT | CALL
   | SET | IF | COND | '(' MACRO-NAME ')'

PREDEFINED-SYMBOL ::=
    '@0' | '@1' | '@2' | '@3' | '@4'
    | '@5' | '@6' | '@7' | '@8' | '@9'
    | '@<' | '@=' | '@>' | '@-' | '@+' | '@[' | '@]'
    | '@@'
    | '@-0' | '@-N' | '@+N'
```

```
STATE-LIST ::= STATE-INCUSION ? '(' 'state' STATE * ')'  STATE-INCUSION ?

STATE ::= '(' STATE-NAME [ STATE-TITLE-TEXT ] BRANCH * ')'

STATE-NAME ::= SYMBOL

STATE-TITLE-TEXT ::= MTEXT

BRANCH ::= '(' MAP-NAME BRANCH-ACTION * ')'
   | '(' nil BRANCH-ACTION * ')'
   | '(' t BRANCH-ACTION * ')'

STATE-INCLUSION ::= '(' 'include' TAGS 'state' STATE-NAME ? ')'
```

入力メソッドがそれ単体で利用されることがなく常に他の入力メソッドに読 み込まれて用いられる場合には、STATE-LIST は省略できる。

STATE-INCLUSION は、TAGS で指定される他の入力メソッドからステート を読み込む。STATE-NAME が与えられていなければ、すべてのステートを読 む。

STATE-TITLE-TEXT は、もし指定されていれば、入力メソッドがこの 状態にある時スクリーン上に表示されるテキストである。省略された場合には TITLE-TEXT が用いられる。

BRANCH の第一の形式では、MAP-NAME は MAP に現われるものでなく てはならない。この場合、MAP-NAME の KEYSEQ の一つに合致するキー列 がタイプされれば、 BRANCH-ACTION が実行される。

BRANCH の第二の形式では、その時点の状態のいずれの BRANCH にも合致 しないキー列がタイプされれば、BRANCH-ACTION が実行される。

nil で始まる BRANCH がなく、入力されたキー列がその時点でのいずれの BRANCH にも合致しない場合には、入力メソッドは初期状態に遷移する。

BRANCH の第三の形式では、その状態に移動した時点で BRANCH-ACTION が実行される。もし初期状態であれば、入力メソッドの入力コンテクストを生 成した時点で BRANCH-ACTION を実行する。

```
BRANCH-ACTION ::= ACTION
```

入力メソッドはシンボルのリストを二つ持つ。

- マーカリスト

  マーカは preediting テキスト中での文字位置を示すシンボルである。MARK アクションはマーカを特定の位置に設定する。MOVE と DELETE ア クションはマーカの位置を参照する。

- 変数リスト

  変数は整数の値を持つシンボルである。値は SET アクションによって設定さ れ、SET, INSERT, IF に参照される。すべての変数の初期値は（暗黙 に） 0 である。

各 PREDEFINED-SYMBOL はマーカとして用いられた場合特別な意味を持つ。

- @0, @1, @2, @3, @4, @5, @6, @7, @8, @9

  それぞれ 0 番目から 9 番目の位置

- @<, @=, @>

  最初の、今の、最後の位置

- @-, @+

  前の、次の位置

- @[, @]

  候補リストが変化する際の前と次の位置

PREDEFINED-SYMBOL のいくつかは、 SELECT アクション中で候補のイン デックスとして用いられた際特別な意味を持つ。

- @<, @=, @>
  現在の候補グループ中での最初の、今の、最後の候補
- @-
  前候補。今の候補が今の候補グループ中での最初のものであれば、前の候補グ ループの最後の候補。
- @+
  次候補。今の候補が今の候補グループ中での最後のものであれば、次の候補グ ループの最初の候補。
- @[, @]
  それぞれ前と後の候補グループ中で、今の候補と同じ候補インデックスを持つ もの。

また、これも特別な意味を持つ。

- @@
  その時点で処理されているキーの数。

以下はサラウンドテキスト処理に用いられる。

- @-0
  サラウンドテキストがサポートされていれば -1 、そうでなければ -2 。
- @-N
  ここで N は正の整数である。この変数の値は、プリエデット中の現在の位 置から N 文字前の文字である。もしプリエディット中で先行する文字が M (M<N) 文字しかなければ、その値は入力スポットから数えて (N-M) 文字前の文 字となる。delete アクションの引数として用いられた場合、この変数は削 除する文字数を指定する。
- @+N
  ここで N は正の整数である。この変数の値は、プリエデット中の現在の位 置から N 文字後の文字である。もしプリエディット中で後続する文字が M (M<N) 文字しかなければ、その値は入力スポットから数えて (N-M) 文字後の文 字となる。delete アクションの引数として用いられた場合、この変数は削 除する文字数を指定する。

各アクションの引数と振舞いは以下の通り。

```
INSERT ::= '(' 'insert' MTEXT ')'
         | MTEXT
   | INTEGER
         | '(' 'insert' SYMBOL ')'
         | '(' 'insert' '(' CANDIDATES * ')' ')'
         | '(' CANDIDATES * ')'

CANDIDATES ::= MTEXT | '(' MTEXT * ')'
```

第一、第二の形式は MTEXT を現在の位置の前に挿入する。

第三の形式は、文字 INTEGER を現在の位置の前に挿入する。

第四の形式は、SYMBOL を変数として扱い、その値が正しい文字コードであ れば現在の位置の前に挿入する。

第五、第六の形式では、CANDIDATES は候補グループを表し、CANDIDATES の各要素が候補を表す。つまり CANDIDATES が M-text であれば、 候補はその M-text 中の文字であり、CANDIDATES が M-text のリストであれ ば、候補はそれらの M-text である。

これらの形式は現在の位置の直前に最初の候補を挿入する。挿入された文字列 には、候補のリストと現在選択されている候補を指す情報が付加されている。

挿入によってマーカの位置は自動的に変更される。

```
DELETE ::= '(' 'delete' SYMBOL ')'
         | '(' 'delete' INTEGER ')'
```

第一の形式は SYMBOL をマーカとして、マーカと現在の位置の間の文字を 削除する。

第二の形式は INTEGER を文字位置として、その文字位置と現在の位置の間 の文字を削除する。

削除によってマーカの位置は自動的に変更される。

```
SELECT ::= '(' 'select' PREDEFINED-SYMBOL ')'
         | '(' 'select' INTEGER ')'
   | '(' 'select' SYMBOL ')'
```

このアクションはまず、現在の位置の直前の文字が、候補リストが付加されて いる文字列に属すかどうかを調べる。そうであれば、その文字列を引数によっ て指定された候補に入れ換える。

第一の形式では PREDEFINED-SYMBOL を前述の候補インデックスとして扱い、 それによって候補リスト中の新しい候補が指定される。

第二の形式では INTEGER は候補インデックスであり、候補リスト中の新し い候補を指定する。

第三の形式では SYMBOL は整数の値を持たなければならず、その値が候補イ ンデックスとして取り扱われる。

```
SHOW ::= '(show)'
```

このアクションは、入力メソッドドライバに現在の位置の前にある文字列に付 加されている候補リストを示すように指示する。

```
HIDE ::= '(hide)'
```

このアクションは、入力メソッドドライバに現在示されている候補リストを隠 すように指示する。

```
MOVE ::= '(' 'move' SYMBOL ')'
       | '(' 'move' INTEGER ')'
```

第一の形式は SYMBOL をマーカとして、それを新しい現在の位置とする。

第二の形式は INTEGER を文字位置として、その位置を新しい現在の位置と する。

```
MARK ::= '(' 'mark' SYMBOL ')'
```

このアクションは SYMBOL をマーカとして、それを現在の位置に設定する。 SYMBOL は PREDEFINED-SYMBOL であってはならない。

```
PUSHBACK :: = '(' 'pushback' INTEGER ')'
            | '(' 'pushback' KEYSEQ ')'
```

第一の形式は、INTEGER の値が正ならば最新の INTEGER 個のキーイベン トをイベントキューに差し戻す。0 ならばすべてのキーイベントを差し戻す。

第二の形式は、KEYSEQ 中のキーをイベントキューに差し戻す。

```
POP ::= '(' 'pop' ')'
```

このアクションはまだ処理されていない最初のイベントをイベントキューから 取り出し破棄する。

```
UNDO :: = '(' 'undo' [ INTEGER | SYMBOL ] ')'
```

引数が無い場合、このアクションは最新の二つのキーイベント、すなわちこの コマンドによって引き起こされたものとその直前のもの、をキャンセルする。

整数値の引数 NUM がある場合、それは正か負であり 0 であってはならない。 正ならば最新のものから数えて NUM 個目のイベントをキャンセルする。負なら ば最新の (- NUM) 個のイベントをキャンセルする。

シンボルの引数がある場合、それは整数に帰着されなくてはならず、その値が 上記の正数値引数の場合と同様に取り扱われる。

```
COMMIT :: = '(commit)'
```

このアクションは現在の preediting テキストをコミットする。

```
UNHANDLE :: = '(unhandle)'
```

このアクションは現在の preediting テキストをコミットし、最新のキーを未 処理として返す。

```
SHIFT :: = '(' 'shift' STATE-NAME ')'
```

STATE-NAME が t ならば、このアクションは現在の状態を一つ前の状態 に遷移させる。そうでなければ STATE-NAME で表される状態に遷移させる。 後者の場合には、STATE-NAME は STATE-LIST に現われるものでなくては ならない。

```
CALL ::= '(' 'call' MODULE-NAME FUNCTION ARG * ')'

ARG ::= INTEGER | SYMBOL | MTEXT | PLIST
```

このアクションは外部モジュール MODULE-NAME の関数 FUNCTION を呼 ぶ。MODULE-NAME と FUNCTION は MODULE-LIST に現われるものでな くてはならない。

関数は (**MPlist** (p. 19) $*$) 型の引数とともに呼ばれる。最初の要素のキーは **Mt** (p. 17) で あり、その値は **MInputContext** (p. 182) 型のオブジェクトへのポインタである。第二 の要素のキーは **Msymbol** (p. 17) であり、値は現在の状態名である。 ARGs は三つ 目以降の要素の値として用いられる。それらの要素のキーは自動的に決定され る。ARG が整数値ならば対応するキーは **Minteger** (p. 23) であり、ARG がシ ンボルならば、対応するキーは **Msymbol** (p. 17)、などのように。

関数は NULL を返すか、または行うべきアクションのリストを表す (**MPlist** (p. 19)) 型の値を返さなくてはならない。

```
SET ::= '(' CMD SYMBOL1 EXPRESSION ')'

CMD ::= 'set' | 'add' | 'sub' | 'mul' | 'div'

EXPRESSION ::= INTEGER | SYMBOL2 | '(' OPERAND EXPRESSION * ')'

OPERAND ::= '+' | '-' | '*' | '/' | '|' | '&' | '!'
            | '=' | '<' | '>' | '<=' | '>='
```

このアクションは SYMBOL1 と SYMBOL2 を変数として、SYMBOL1 の 値を以下のように設定する。

CMD が 'set' ならば、SYMBOL1 の値を EXPRESSION の値に設定する。

CMD が 'add' ならば、SYMBOL1 の値を EXPRESSION の値だけ増やす。

CMD が 'sub' ならば、SYMBOL1 の値を EXPRESSION の値だけ減らす。

CMD が 'mul' ならば、SYMBOL1 の値を EXPRESSION の値を掛けたも のにする。

CMD が 'div' ならば、SYMBOL1 の値を EXPRESSION の値で割ったも のにする。

```
IF ::= '(' CONDITION ACTION-LIST1 ACTION-LIST2 ? ')'

CONDITION ::= [ '=' | '<' | '>' | '<=' | '>=' ] EXPRESSION1 EXPRESSION2

ACTION-LIST1 ::= '(' ACTION * ')'

ACTION-LIST2 ::= '(' ACTION * ')'
```

このアクションは、 CONDITION が真であれば ACTION-LIST1 を実行し、 そうでなければ ACTION-LIST2 を (もしあれば) 実行する。

SYMBOL1 と SYMBOL2 は変数として扱われる。

```
COND ::= '(' 'cond' [ '(' EXPRESSION ACTION * ') ] * ')'
```

このアクションは対応する EXPRESSION が 0 でない値をとる最初のアクショ ン ACTION を実行する。

D.9.3 SEE ALSO

Input Methods provided by the m17n database (p. **??**), **mdbGeneral(5)** (p. 204)

Appendix E

Tutorial for writing the m17n database

This section contains tutorials for writing various database files of the m17n database.

E.1 Tutorial of input method

E.1.1 Structure of an input method file

An input method is defined in a ∗.mim file with this format.

```
(input-method LANG NAME)

(description (_ "DESCRIPTION"))

(title "TITLE-STRING")

(map
  (MAP-NAME
    (KEYSEQ MAP-ACTION MAP-ACTION ...)        <- rule
    (KEYSEQ MAP-ACTION MAP-ACTION ...)        <- rule
    ...)
  (MAP-NAME
    (KEYSEQ MAP-ACTION MAP-ACTION ...)        <- rule
    (KEYSEQ MAP-ACTION MAP-ACTION ...)        <- rule
    ...)
  ...)

(state
  (STATE-NAME
    (MAP-NAME BRANCH-ACTION BRANCH-ACTION ...)   <- branch
    ...)
  (STATE-NAME
    (MAP-NAME BRANCH-ACTION BRANCH-ACTION ...)   <- branch
    ...)
  ...)
```

Lowercase letters and parentheses are literals, so they must be written as they are. Uppercase letters represent arbitrary strings.

KEYSEQ specifies a sequence of keys in this format:

```
  (SYMBOLIC-KEY SYMBOLIC-KEY ...)
```

where SYMBOLIC-KEY is the keysym value returned by the xev command. For instance

```
  (n i)
```

represents a key sequence of <n> and <i>. If all SYMBOLIC-KEYs are ASCII characters, you can use the short form

```
  "ni"
```

instead. Consult **インプットメソッド** (p. 215) for Non-ASCII characters.

Both MAP-ACTION and BRANCH-ACTION are a sequence of actions of this format:

```
  (ACTION ARG ARG ...)
```

The most common action is `insert`, which is written as this:

```
(insert "TEXT")
```

But as it is very frequently used, you can use the short form

```
"TEXT"
```

If `"TEXT"` contains only one character "C", you can write it as

```
(insert ?C)
```

or even shorter as

```
?C
```

So the shortest notation for an action of inserting "a" is

```
?a
```

E.1.2 Simple example of capslock

Here is a simple example of an input method that works as CapsLock.

```
(input-method en capslock)
(description (_ "Upcase all lowercase letters"))
(title "a->A")
(map
  (toupper ("a" "A") ("b" "B") ("c" "C") ("d" "D") ("e" "E")
           ("f" "F") ("g" "G") ("h" "H") ("i" "I") ("j" "J")
           ("k" "K") ("l" "L") ("m" "M") ("n" "N") ("o" "O")
           ("p" "P") ("q" "Q") ("r" "R") ("s" "S") ("t" "T")
           ("u" "U") ("v" "V") ("w" "W") ("x" "X") ("y" "Y")
           ("z" "Z")))
(state
  (init (toupper)))
```

When this input method is activated, it is in the initial condition of the first state (in this case, the only state `init`). In the initial condition, no key is being processed and no action is suspended. When the input method receives a key event <a>, it searches branches in the current state for a rule that matches <a> and finds one in the map `toupper`. Then it executes MAP-ACTIONs (in this case, just inserting "A" in the preedit buffer). After all MAP-ACTIONs have been executed, the input method shifts to the initial condition of the current state.

The shift to *the initial condition of the first state* has a special meaning; it commits all characters in the preedit buffer then clears the preedit buffer.

As a result, "A" is given to the application program.

When a key event does not match with any rule in the current state, that event is unhandled and given back to the application program.

Turkish users may want to extend the above example for "聴" (U+0130: LATIN CAPITAL LETTER I WITH DOT ABOVE). It seems that assigning the key sequence <i> <i> for that character is convenient. So, he will add this rule in `toupper`.

```
("ii" "聴")
```

However, we already have the following rule:

```
("i" "I")
```

What will happen when a key event <i> is sent to the input method?

No problem. When the input method receives <i>, it inserts "I" in the preedit buffer. It knows that there is another rule that may match the additional key event <i>. So, after inserting "I", it suspends the normal behavior of shifting to the initial condition, and waits for another key. Thus, the user sees "I" with underline, which indicates it is not yet committed.

When the input method receives the next <i>, it cancels the effects done by the rule for the previous "i" (in this case, the preedit buffer is cleared), and executes MAP-ACTIONs of the rule for "ii". So, "聽" is inserted in the preedit buffer. This time, as there are no other rules that match with an additional key, it shifts to the initial condition of the current state, which leads to commit "聽".

Then, what will happen when the next key event is <a> instead of <i>?

No problem, either.

The input method knows that there are no rules that match the <i> <a> key sequence. So, when it receives the next <a>, it executes the suspended behavior (i.e. shifting to the initial condition), which leads to commit "I". Then the input method tries to handle <a> in the current state, which leads to commit "A".

So far, we have explained MAP-ACTION, but not BRANCH-ACTION. The format of BRANCH-ACTION is the same as that of MAP-ACTION. It is executed only after a matching rule has been determined and the corresponding MAP-ACTIONs have been executed. A typical use of BRANCH-ACTION is to shift to a different state.

To see this effect, let us modify the current input method to upcase only word-initial letters (i.e. to capitalize). For that purpose, we modify the "init" state as this:

```
  (init
    (toupper (shift non-upcase)))
```

Here `(shift non-upcase)` is an action to shift to the new state `non-upcase`, which has two branches as below:

```
  (non-upcase
    (lower)
    (nil (shift init)))
```

The first branch is simple. We can define the new map `lower` as the following to insert lowercase letters as they are.

```
(map
  ...
  (lower ("a" "a") ("b" "b") ("c" "c") ("d" "d") ("e" "e")
         ("f" "f") ("g" "g") ("h" "h") ("i" "i") ("j" "j")
         ("k" "k") ("l" "l") ("m" "m") ("n" "n") ("o" "o")
         ("p" "p") ("q" "q") ("r" "r") ("s" "s") ("t" "t")
         ("u" "u") ("v" "v") ("w" "w") ("x" "x") ("y" "y")
         ("z" "z")))
```

The second branch has a special meaning. The map name `nil` means that it matches with any key event that does not match any rules in the other maps in the current state. In addition, it does not consume any key event. We will show the full code of the new input method before explaining how it works.

```
(input-method en titlecase)
(description (_ "Titlecase letters"))
(title "abc->Abc")
(map
  (toupper ("a" "A") ("b" "B") ("c" "C") ("d" "D") ("e" "E")
           ("f" "F") ("g" "G") ("h" "H") ("i" "I") ("j" "J")
           ("k" "K") ("l" "L") ("m" "M") ("n" "N") ("o" "O")
           ("p" "P") ("q" "Q") ("r" "R") ("s" "S") ("t" "T")
```

```
           ("u" "U") ("v" "V") ("w" "W") ("x" "X") ("y" "Y")
           ("z" "Z") ("ii" "聴"))
  (lower ("a" "a") ("b" "b") ("c" "c") ("d" "d") ("e" "e")
         ("f" "f") ("g" "g") ("h" "h") ("i" "i") ("j" "j")
         ("k" "k") ("l" "l") ("m" "m") ("n" "n") ("o" "o")
         ("p" "p") ("q" "q") ("r" "r") ("s" "s") ("t" "t")
         ("u" "u") ("v" "v") ("w" "w") ("x" "x") ("y" "y")
         ("z" "z")))
(state
  (init
    (toupper (shift non-upcase)))
  (non-upcase
    (lower (commit))
    (nil (shift init))))
```

Let's see what happens when the user types the key sequence <a> <b> < >. Upon <a>, "A" is inserted into the buffer and the state shifts to `non-upcase`. So, the next <b> is handled in the `non-upcase` state. As it matches a rule in the map `lower`, "b" is inserted in the preedit buffer and characters in the buffer ("Ab") are committed explicitly by the "commit" command in BRANCH-ACTION. After that, the input method is still in the `non-upcase` state. So the next < > is also handled in `non-upcase`. For this time, no rule in this state matches it. Thus the branch `(nil (shift init))` is selected and the state is shifted to `init`. Please note that < > is not yet handled because the map `nil` does not consume any key event. So, the input method tries to handle it in the `init` state. Again no rule matches it. Therefore, that event is given back to the application program, which usually inserts a space for that.

When you type "a quick blown fox" with this input method, you get "A Quick Blown Fox". OK, you find a typo in "blown", which should be "brown". To correct it, you probably move the cursor after "l" and type <Backspace> and <r>. However, if the current input method is still active, a capital "R" is inserted. It is not a sophisticated behavior.

E.1.3 Example of utilizing surrounding text support

To make the input method work well also in such a case, we must use "surrounding text support". It is a way to check characters around the inputting spot and delete them if necessary. Note that this facility is available only with Gtk+ applications and Qt applications. You cannot use it with applications that use XIM to communicate with an input method.

Before explaining how to utilize "surrounding text support", you must understand how to use variables, arithmetic comparisons, and conditional actions.

At first, any symbol (except for several preserved ones) used as ARG of an action is treated as a variable. For instance, the commands

```
(set X 32) (insert X)
```

set the variable `X` to integer value 32, then insert a character whose Unicode character code is 32 (i.e. SPACE).

The second argument of the `set` action can be an expression of this form:

```
(OPERATOR ARG1 [ARG2])
```

Both ARG1 and ARG2 can be an expression. So,

```
(set X (+ (* Y 32) Z))
```

sets `X` to the value of `Y * 32 + Z`.

We have the following arithmetic/bitwise OPERATORs (require two arguments):

```
+ - * / & |
```

these relational OPERATORs (require two arguments):

```
== <= >= < >
```

and this logical OPERATOR (requires one argument):

```
!
```

For surrounding text support, we have these preserved variables:

```
@-0, @-N, @+N (N is a positive integer)
```

The values of them are predefined as below and can not be altered.

- `@-0`

  -1 if surrounding text is supported, -2 if not.

- `@-N`

  The Nth previous character in the preedit buffer. If there are only M (M<N) previous characters in it, the value is the (N-M)th previous character from the inputting spot.

- `@+N`

  The Nth following character in the preedit buffer. If there are only M (M<N) following characters in it, the value is the (N-M)th following character from the inputting spot.

So, provided that you have this context:

```
ABC|def|GHI
```

("def" is in the preedit buffer, two "|"s indicate borders between the preedit buffer and the surrounding text) and your current position in the preedit buffer is between "d" and "e", you get these values:

```
@-3 -- ?B
@-2 -- ?C
@-1 -- ?d
@+1 -- ?e
@+2 -- ?f
@+3 -- ?G
```

Next, you have to understand the conditional action of this form:

```
(cond
  (EXPR1 ACTION ACTION ...)
  (EXPR2 ACTION ACTION ...)
  ...)
```

where EXPRn are expressions. When an input method executes this action, it resolves the values of EXPRn one by one from the first branch. If the value of EXPRn is resolved into nonzero, the corresponding actions are executed.

Now you are ready to write a new version of the input method "Titlecase".

```
(input-method en titlecase2)
(description (_ "Titlecase letters"))
(title "abc->Abc")
(map
```

```
  (toupper ("a" "A") ("b" "B") ("c" "C") ("d" "D") ("e" "E")
           ("f" "F") ("g" "G") ("h" "H") ("i" "I") ("j" "J")
           ("k" "K") ("l" "L") ("m" "M") ("n" "N") ("o" "O")
           ("p" "P") ("q" "Q") ("r" "R") ("s" "S") ("t" "T")
           ("u" "U") ("v" "V") ("w" "W") ("x" "X") ("y" "Y")
           ("z" "Z") ("ii" "聴")))
(state
  (init
    (toupper

     ;; Now we have exactly one uppercase character in the preedit
     ;; buffer.  So, "@-2" is the character just before the inputting
     ;; spot.

     (cond ((| (& (>= @-2 ?A) (<= @-2 ?Z))
               (& (>= @-2 ?a) (<= @-2 ?z))
               (= @-2 ?聴))

    ;; If the character before the inputting spot is A..Z,
    ;; a..z, or 聴, remember the only character in the preedit
    ;; buffer in the variable X and delete it.

    (set X @-1) (delete @-)

    ;; Then insert the lowercase version of X.

    (cond ((= X ?聴) "i")
                  (1 (set X (+ X 32)) (insert X))))))))
```

The above example contains the new action `delete`. So, it is time to explain more about the preedit buffer. The preedit buffer is a temporary place to store a sequence of characters. In this buffer, the input method keeps a position called the "current position". The current position exists between two characters, at the beginning of the buffer, or at the end of the buffer. The `insert` action inserts characters before the current position. For instance, when your preedit buffer contains "ab.c" ("." indicates the current position),

```
  (insert "xyz")
```

changes the buffer to "abxyz.c".

There are several predefined variables that represent a specific position in the preedit buffer. They are:

- `@<, @=, @>`

  The first, current, and last positions.

- `@-, @+`

  The previous and the next positions.

The format of the `delete` action is this:

```
  (delete POS)
```

where POS is a predefined positional variable. The above action deletes the characters between POS and the current position. So, `(delete @-)` deletes one character before the current position. The other examples of `delete` include the followings:

```
  (delete @+)  ; delete the next character
  (delete @<)  ; delete all the preceding characters in the buffer
  (delete @>)  ; delete all the following characters in the buffer
```

You can change the current position using the `move` action as below:

```
(move @-)  ; move the current position to the position before the
             previous character
(move @<)  ; move to the first position
```

Other positional variables work similarly.

Let's see how our new example works. Whatever a key event is, the input method is in its only state, `init`. Since an event of a lower letter key is firstly handled by MAP-ACTIONs, every key is changed into the corresponding uppercase and put into the preedit buffer. Now this character can be accessed with `@-1`.

How can we tell whether the new character should be a lowercase or an uppercase? We can do so by checking the character before it, i.e. `@-2`. BRANCH-ACTIONs in the `init` state do the job.

It first checks if the character `@-2` is between A to Z, between a to z, or 聽 by the conditional below.

```
(cond ((| (& (>= @-2 ?A) (<= @-2 ?Z))
          (& (>= @-2 ?a) (<= @-2 ?z))
          (= @-2 ?聽))
```

If not, there is nothing to do specially. If so, our new key should be changed back into lowercase. Since the uppercase character is already in the preedit buffer, we retrieve and remember it in the variable `X` by

```
(set X @-1)
```

and then delete that character by

```
(delete @-)
```

Lastly we re-insert the character in its lowercase form. The problem here is that "聽" must be changed into "i", so we need another conditional. The first branch

```
((= X ?聽) "i")
```

means that "if the character remembered in X is ' 聽', 'i' is inserted".

The second branch

```
(1 (set X (+ X 32)) (insert X))
```

starts with "1", which is always resolved into nonzero, so this branch is a catchall. Actions in this branch increase `X` by 32, then insert `X`. In other words, they change A...Z into a...z respectively and insert the resulting lowercase character into the preedit buffer. As the input method reaches the end of the BRANCH-ACTIONs, the character is commited.

This new input method always checks the character before the current position, so "A Quick Blown Fox" will be successfully fixed to "A Quick Brown Fox" by the key sequence <BackSpace> <r>.

Appendix F

GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF

and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses

a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Index