The m17n Library

1.5.5

Generated by Doxygen 1.5.6

Thu Oct 15 14:11:30 2009

Contents

Chapter 1

The m17n Library Documentation

1.1 What is the m17n library?

The *m17n library* is a multilingual text processing library for the C language.

- It is a free and open source software.
- It is for any GNU/Linux and Unix applications/libraries.
- It realizes multilingualization of many aspects of applications/libraries.

The word "m17n" is an abbreviation of "multilingualization".

The m17n library provides following facilities to handle multilingual text.

- *M-text*: A data structure for a multilingual text. It is basically a string but with attributes called text property, and is designed to substitute for the C string. It is the most important object of the m17n library.
- Functions for creating and processing M-texts.
- Functions for converting M-texts from/to strings encoded in various existing formats.
- A huge character space, which contains all the Unicode characters and more non-Unicode characters.
- *Chartable:* A data structure that contains per-character information efficiently.
- Functions for inputting and displaying M-texts on a window system.

1.2 How to use it?

Simply include <m17n.h> in your program, and link it with the m17n library by -lm17n. See **Introduction** (p. 5) for the detail.

1.3 External libraries and data

The m17n library utilizes these external libraries. They are not mandatory but many functions of the m17n library depend on them.

- m17n-db – `http://www.m17n.org/m17n-lib-en/download/m17n-db-1.5.0.tar.gz`

  Provide various information to the m17n library.

- libxml2 – `http://xmlsoft.org/`

  Used by the functions **mtext_serialize()** (p. 57) and **mtext_deserialize()** (p. 58). Those functions return NULL when libxml2 is not available,

- fribidi – `http://fribidi.sourceforge.net/`

  Used for BIDI processing. If it is not available, the rendering engine of the m17n library can't handle such script as Arabic and Hebrew correctly.

- freetype – `http://www.freetype.org/`

  Used for handling local fonts.

- fontconfig – `http://www.fontconfig.org/`

  Used for handling local fonts supported by the freetype library.

- fontconfig – `http://freedesktop.org/Software/fontconfig`

  Used for finding local fonts in combination with Xft.

- xft – `http://freedesktop.org/Software/Xft`

  Used for drawing text with local fonts by X Render Extension of X server in combination with fontconfig.

- GD

  Used for rendering text with local fonts on bitmap/pixmap.

- libotf – `http://www.m17n.org/libotf/`

  Used for handling OpenTypee fonts in combination with freetype and Xft.

- anthy – `http://anthy.sourceforge.jp/`

  Used for the Japanese input method ja-anthy.mim.

- wordcut – `http://thaiwordseg.sourceforge.net/`

  Used for finding Thai word boundary in the example program example/linebreak.c.

1.4 Contact us:

Global IT Security Group

Information Technology Research Institute

National Institute of Advanced Industrial Science and Technology

Web: `http://www.m17n.org/m17n-lib-en/`

Bug report: `m17n-lib-bug-XXXX@m17n.org` (Replace XXXX with current year in 4 digits.)

Mailing lists: `http://www.m17n.org/m17n-lib-en/mailinglist.html`

1.5 Acknowledgements

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

Module Documentation

2.1 Introduction

Introduction to the m17n library.

Defines

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

  Initialize the m17n library.

- #define **M17N_FINI**()

  Finalize the m17n library.

Enumerations

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

  Enumeration for the status of the m17n library.

Functions

- enum **M17NStatus** **m17n_status** (void)

  Report which part of the m17n library is initialized.

2.1.1 Detailed Description

Introduction to the m17n library.

API LEVELS

The API of the m17n library is divided into these five.

1. CORE API

   It provides basic modules to handle M-texts. To use this API, an application program must include <m17n-core.h> and be linked with -lm17n-core.

2. SHELL API

   It provides modules for character properties, character set handling, code conversion, etc. They load various kinds of data from the database on demand. To use this API, an application program must include <m17n.h> and be linked with -lm17n-core -lm17n.

   When you use this API, CORE API is also available.

3. FLT API

   It provides modules for text shaping using **Font Layout Table** (p. 211). To use this API, an application program must include <m17n.h> and be linked with -lm17n-core -lm17n-flt.

   When you use this API, CORE API is also available.

4. GUI API

   It provides GUI modules such as drawing and inputting M-texts on a graphic device. This API itself is independent of graphic devices, but most functions require an argument MFrame that is created for a specific type of graphic devices. The currently supported graphic devices are null device, the X Window System, and image data (gdImagePtr) of the GD library.

   On a frame of a null device, you cannot draw text nor use input methods. However, functions like **mdraw_glyph_list()** (p. 148), etc. are available.

   On a frame of the X Window System, you can use the whole GUI API.

   On a frame of the GD library, you can use all drawing API but cannot use input methods.

   To use this API, an application program must include <m17n-gui.h> and be linked with -lm17n-core -lm17n -lm17n-gui.

   When you use this API, CORE, SHELL, and FLT APIs are also available.

5. MISC API

   It provides miscellaneous functions to support error handling and debugging. This API cannot be used standalone; it must be used with one or more APIs listed above. To use this API, an application program must include <m17n-misc.h> in addition to one of the header files described above.

See also the section **m17n-config(1)** (p. 197).

ENVIRONMENT VARIABLES

The m17n library pays attention to the following environment variables.

- `M17NDIR`

  The name of the directory that contains data of the m17n database. See **Database** (p. 60) for details.

- `MDEBUG_XXX`

  Environment variables whose names start with "MDEBUG_" control debug information output. See **Debugging** (p. 157) for details.

API NAMING CONVENTION

The m17n library exports functions, variables, macros, and types. All of them start with the letter ’m’ or ’M’, and are followed by an object name (e.g. "symbol", "plist") or a module name (e.g. draw, input). Note that the name of M-text objects start with "mtext" and not with "mmtext".

- functions – mobject() or mobject_xxx()

  They start with ’m’ and are followed by an object name in lower case. Words are separated by ’_’. For example, **msymbol()** (p. 14), **mtext_ref_char()** (p. 38), **mdraw_text()** (p. 145).

- non-symbol variables – mobject, or mobject_xxx

  The naming convention is the same as functions (e.g. mface_large).

- symbol variables – Mname

  Variables of the type MSymbol start with ’M’ and are followed by their names. Words are separated by ’_’. For example, Mlanguage (the name is "language"), Miso_2022 (the name is "iso-2022").

- macros – MOBJECT_XXX

  They start with ’M’ and are followed by an object name in upper case. Words are separated by ’_’.

- types – MObject or MObjectXxx

  They start with ’M’ and are followed by capitalized object names. Words are concatenated directly and no ’_’ are used. For example, **MConverter** (p. 164), **MInputDriver** (p. 190).

2.1.2 Define Documentation

2.1.2.1 #define M17NLIB_MAJOR_VERSION

The **M17NLIB_MAJOR_VERSION** (p. 7) macro gives the major version number of the m17n library.

2.1.2.2 #define M17NLIB_MINOR_VERSION

The **M17NLIB_MINOR_VERSION** (p. 7) macro gives the minor version number of the m17n library.

2.1.2.3 #define M17NLIB_PATCH_LEVEL

The **M17NLIB_PATCH_LEVEL** (p. 7) macro gives the patch level number of the m17n library.

2.1.2.4 #define M17NLIB_VERSION_NAME

The **M17NLIB_VERSION_NAME** (p. 7) macro gives the version name of the m17n library as a string.

2.1.2.5 #define M17N_INIT()

Initialize the m17n library.

The macro **M17N_INIT()** (p. 7) initializes the m17n library. This macro must be called before any m17n functions are used.

It is safe to call this macro multiple times, but in that case, the macro **M17N_FINI()** (p. 8) must be called the same times to free the memory.

If the initialization was successful, the external variable **merror_code** (p. 156) is set to 0. Otherwise it is set to -1.

See Also:

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

2.1.2.6 #define M17N_FINI()

Finalize the m17n library.

The macro **M17N_FINI()** (p. 8) finalizes the m17n library. It frees all the memory area used by the m17n library. Once this macro is called, no m17n functions should be used until the macro **M17N_INIT()** (p. 7) is called again.

If the macro **M17N_INIT()** (p. 7) was called N times, the Nth call of this macro actually free the memory.

See Also:

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

2.1.3 Enumeration Type Documentation

2.1.3.1 enum M17NStatus

Enumeration for the status of the m17n library.

The enum **M17NStatus** (p. 8) is used as a return value of the function **m17n_status()** (p. 8).

Enumerator:

M17N_NOT_INITIALIZED No modules is initialized, and all modules are finalized.

M17N_CORE_INITIALIZED Only the modules in CORE API are initialized.

M17N_SHELL_INITIALIZED Only the modules in CORE and SHELL APIs are initialized.

M17N_GUI_INITIALIZED All modules are initialized.

2.1.4 Function Documentation

2.1.4.1 enum M17NStatus m17n_status (void)

Report which part of the m17n library is initialized.

The **m17n_status()** (p. 8) function returns one of these values depending on which part of the m17n library is initialized:

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

2.2 CORE API

API provided by libm17n-core.so.

Modules

- **Managed Object**

  Objects managed by the reference count.

- **Symbol**

  Symbol objects and API for them.

- **Property List**

  Property List objects and API for them.

- **Character**

  Character objects and API for them.

- **Chartable**

  Chartable objects and API for them.

- **M-text**

  M-text objects and API for them.

- **Text Property**

  Function to handle text properties.

- **Database**

  The m17n database and API for it.

Defines

- #define **M17N_FUNC**(func) ((**M17NFunc**) (func))

  Wrapper for a generic function type.

Typedefs

- typedef void(∗ **M17NFunc**)(void)

  Generic function type.

2.2.1 Detailed Description

API provided by libm17n-core.so.

2.2.2 Define Documentation

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

Wrapper for a generic function type.

The macro **M17N_FUNC()** (p. 10) casts a function to the type **M17NFunc** (p. 10).

2.2.3 Typedef Documentation

2.2.3.1 typedef void(∗ M17NFunc)(void)

Generic function type.

M17NFunc (p. 10) is a generic function type for setting a function pointer as a value of **MSymbol** (p. 14) property or **MPlist** (p. 19).

See Also:

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

2.3 Managed Object

Objects managed by the reference count.

Data Structures

- struct **M17NObjectHead**

  The first member of a managed object.

Functions

- void ∗ **m17n_object** (int size, void(∗freer)(void ∗))

  Allocate a managed object.
- int **m17n_object_ref** (void ∗object)

  Increment the reference count of a managed object.
- int **m17n_object_unref** (void ∗object)

  Decrement the reference count of a managed object.

2.3.1 Detailed Description

Objects managed by the reference count.

Managed objects are objects managed by the reference count.

There are some types of m17n objects that are managed by their reference count. Those objects are called *managed objects*. When created, the reference count of a managed object is initialized to one. The **m17n_object_ref()** (p. 12) function increments the reference count of a managed object by one, and the **m17n_object_unref()** (p. 12) function decrements by one. A managed object is automatically freed when its reference count becomes zero.

A property whose key is a managing key can have only a managed object as its value. Some functions, for instance **msymbol_put()** (p. 16) and **mplist_put()** (p. 20), pay special attention to such a property.

In addition to the predefined managed object types, users can define their own managed object types. See the documentation of the **m17n_object()** (p. 11) for more details.

2.3.2 Function Documentation

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

Allocate a managed object.

The **m17n_object()** (p. 11) function allocates a new managed object of **size** bytes and sets its reference count to 1. **freer** is the function that is used to free the object when the reference count becomes 0. If **freer** is NULL, the object is freed by the free() function.

The heading bytes of the allocated object is occupied by **M17NObjectHead** (p. 161). That area is reserved for the m17n library and application programs should never touch it.

Return value:

This function returns a newly allocated object.

Errors:

This function never fails.

Example:

```
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*)

Increment the reference count of a managed object.

The **m17n_object_ref()** (p. 12) function increments the reference count of the managed object pointed to by **object**.

Return value:

This function returns the resulting reference count if it fits in a 16-bit unsigned integer (i.e. less than 0x10000). Otherwise, it return -1.

Errors:

This function never fails.

2.3.2.3 int m17n_object_unref (void * *object*)

Decrement the reference count of a managed object.

The **m17n_object_unref()** (p. 12) function decrements the reference count of the managed object pointed to by **object**. When the reference count becomes zero, the object is freed by its freer function.

Return value:

This function returns the resulting reference count if it fits in a 16-bit unsigned integer (i.e. less than 0x10000). Otherwise, it returns -1. Thus, the return value zero means that **object** is freed.

Errors:

This function never fails.

2.4 Symbol

Symbol objects and API for them.

Typedefs

- typedef struct MSymbolStruct ∗ **MSymbol**

  Type of symbols.

Functions

- **MSymbol msymbol** (const char ∗name)

  Get a symbol.

- **MSymbol msymbol_as_managing_key** (const char ∗name)

  Create a managing key.

- int **msymbol_is_managing_key** (**MSymbol** symbol)

  Check if a symbol is a managing key.

- **MSymbol msymbol_exist** (const char ∗name)

  Search for a symbol that has a specified name.

- char ∗ **msymbol_name** (**MSymbol** symbol)

  Get symbol name.

- int **msymbol_put** (**MSymbol** symbol, **MSymbol** key, void ∗val)

  Set the value of a symbol property.

- void ∗ **msymbol_get** (**MSymbol** symbol, **MSymbol** key)

  Get the value of a symbol property.

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

  Set the value (function pointer) of a symbol property.

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

  Get the value (function pointer) of a symbol property.

Variables

- **MSymbol Mnil**

  Symbol whose name is "nil".

- **MSymbol Mt**

  Symbol whose name is "t".

- **MSymbol Mstring**

  Symbol whose name is "string".

- **MSymbol Msymbol**

  Symbol whose name is "symbol".

2.4.1 Detailed Description

Symbol objects and API for them.

The m17n library uses objects called *symbols* as unambiguous identifiers. Symbols are similar to atoms in the X library, but a symbol can have zero or more *symbol properties*. A symbol property consists of a *key* and a *value*, where key is also a symbol and value is anything that can be cast to `(void *)`. "The symbol property that belongs to the symbol S and whose key is K" may be shortened to "K property of S".

Symbols are used mainly in the following three ways.

- As keys of symbol properties and other properties.
- To represent various objects, e.g. charsets, coding systems, fontsets.
- As arguments of the m17n library functions to control their behavior.

There is a special kind of symbol, a *managing key*. The value of a property whose key is a managing key must be a *managed object*. See **Managed Object** (p. 11) for the detail.

2.4.2 Typedef Documentation

2.4.2.1 typedef struct MSymbolStruct∗ MSymbol

Type of symbols.

The type **MSymbol** (p. 14) is for a *symbol* object. Its internal structure is concealed from application programs.

2.4.3 Function Documentation

2.4.3.1 MSymbol msymbol (const char ∗ *name*)

Get a symbol.

The **msymbol()** (p. 14) function returns the canonical symbol whose name is **name**. If there is none, one is created. The created one is not a managing key.

Symbols whose name starts by two spaces are reserved by the m17n library, and are used by the library only internally.

Return value:

This function returns the found or created symbol.

Errors:

This function never fails.

See Also:

msymbol_as_managing_key() (p. 15), **msymbol_name()** (p. 15), **msymbol_exist()** (p. 15)

2.4.3.2 MSymbol msymbol_as_managing_key (const char ∗ *name*)

Create a managing key.

The **msymbol_as_managing_key**() (p. 15) function returns a newly created managing key whose name is **name**. It there already exists a symbol of name **name**, it returns **Mnil** (p. 17).

Symbols whose name starts by two spaces are reserved by the m17n library, and are used by the library only internally.

Return value:
If the operation was successful, this function returns the created symbol. Otherwise, it returns **Mnil** (p. 17).

Errors:
MERROR_SYMBOL

See Also:
msymbol() (p. 14), **msymbol_exist()** (p. 15)

2.4.3.3 int msymbol_is_managing_key (MSymbol *symbol*)

Check if a symbol is a managing key.

The **msymbol_is_managing_key()** (p. 15) function checks if the symbol **symbol** is a managing key or not.

Return value:
Return 1 if the symbol is a managing key. Otherwise, return 0.

2.4.3.4 MSymbol msymbol_exist (const char ∗ *name*)

Search for a symbol that has a specified name.

The **msymbol_exist()** (p. 15) function searches for the symbol whose name is **name**.

Return value:
If such a symbol exists, **msymbol_exist()** (p. 15) returns that symbol. Otherwise it returns the predefined symbol **Mnil** (p. 17).

Errors:
This function never fails.

See Also:
msymbol_name() (p. 15), **msymbol()** (p. 14)

2.4.3.5 char∗ msymbol_name (MSymbol *symbol*)

Get symbol name.

The **msymbol_name()** (p. 15) function returns a pointer to a string containing the name of **symbol**.

Errors:
This function never fails.

See Also:
msymbol() (p. 14), **msymbol_exist()** (p. 15)

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

Set the value of a symbol property.

The **msymbol_put()** (p. 16) function assigns **val** to the value of the symbol property that belongs to **symbol** and whose key is **key**. If the symbol property already has a value, **val** overwrites the old one. Both **symbol** and **key** must not be **Mnil** (p. 17).

If **key** is a managing key, **val** must be a managed object. In this case, the reference count of the old value, if not `NULL`, is decremented by one, and that of **val** is incremented by one.

Return value:

If the operation was successful, **msymbol_put()** (p. 16) returns 0. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_SYMBOL`

See Also:

msymbol_get() (p. 16)

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

Get the value of a symbol property.

The **msymbol_get()** (p. 16) function searches for the value of the symbol property that belongs to **symbol** and whose key is **key**. If **symbol** has such a symbol property, its value is returned. Otherwise `NULL` is returned.

Return value:

If an error is detected, **msymbol_get()** (p. 16) returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_SYMBOL`

See Also:

msymbol_put() (p. 16)

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

Set the value (function pointer) of a symbol property.

The **msymbol_put_func()** (p. 16) function is similar to **msymbol_put()** (p. 16) but for setting function pointer **func** as the property value of **symbol** for key **key**.

See Also:

msymbol_put() (p. 16), **M17N_FUNC()** (p. 10)

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

Get the value (function pointer) of a symbol property.

The **msymbol_get_func()** (p. 16) function is similar to **msymbol_get()** (p. 16) but for getting a function pointer form the property of symbol **symbol**.

See Also:

msymbol_get() (p. 16)

2.4.4 Variable Documentation

2.4.4.1 MSymbol Mnil

Symbol whose name is "nil".

The symbol **Mnil** (p. 17) has the name `"nil"` and, in general, represents *false* or *no*. When coerced to "int", its value is zero. **Mnil** (p. 17) can't have any symbol property.

2.4.4.2 MSymbol Mt

Symbol whose name is "t".

The symbol **Mt** (p. 17) has the name `"t"` and, in general, represents *true* or *yes*.

2.4.4.3 MSymbol Mstring

Symbol whose name is "string".

The symbol **Mstring** (p. 17) has the name `"string"` and is used as an argument of the functions **mchar_define_property**() (p. 25), etc.

2.4.4.4 MSymbol Msymbol

Symbol whose name is "symbol".

The symbol **Msymbol** (p. 17) has the name `"symbol"` and is used as an argument of the functions **mchar_define_property**() (p. 25), etc.

2.5 Property List

Property List objects and API for them.

Typedefs

- typedef struct **MPlist** **MPlist**

  Type of property list objects.

Functions

- **MPlist** ∗ **mplist_deserialize** (**MText** ∗mt)

  Generate a property list by deserializing an M-text.

- **MPlist** ∗ **mplist** (void)

  Create a property list object.

- **MPlist** ∗ **mplist_copy** (**MPlist** ∗plist)

  Copy a property list.

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

  Set the value of a property in a property list.

- void ∗ **mplist_get** (**MPlist** ∗plist, **MSymbol** key)

  Get the value of a property in a property list.

- **MPlist** ∗ **mplist_put_func** (**MPlist** ∗plist, **MSymbol** key, **M17NFunc** func)

  Set the value (function pointer) of a property in a property list.

- **M17NFunc** **mplist_get_func** (**MPlist** ∗plist, **MSymbol** key)

  Get the value (function pointer) of a property in a property list.

- **MPlist** ∗ **mplist_add** (**MPlist** ∗plist, **MSymbol** key, void ∗val)

  Add a property at the end of a property list.

- **MPlist** ∗ **mplist_push** (**MPlist** ∗plist, **MSymbol** key, void ∗val)

  Add a property at the beginning of a property list.

- void ∗ **mplist_pop** (**MPlist** ∗plist)

  Remove a property at the beginning of a property list.

- **MPlist** ∗ **mplist_find_by_key** (**MPlist** ∗plist, **MSymbol** key)

  Find a property of a specific key in a property list.

- **MPlist** ∗ **mplist_find_by_value** (**MPlist** ∗plist, void ∗val)

  Find a property of a specific value in a property list.

- **MPlist** ∗ **mplist_next** (**MPlist** ∗plist)

  Return the next sublist of a property list.

- **MPlist** ∗ **mplist_set** (**MPlist** ∗plist, **MSymbol** key, void ∗val)

  Set the first property in a property list.

- int **mplist_length** (**MPlist** ∗plist)

  Return the length of a property list.

- **MSymbol** **mplist_key** (**MPlist** ∗plist)

  Return the key of the first property in a property list.

- void ∗ **mplist_value** (**MPlist** ∗plist)

  Return the value of the first property in a property list.

Variables

- **MSymbol Minteger**

  Symbol whose name is "integer".

- **MSymbol Mplist**

  Symbol whose name is "plist".

- **MSymbol Mtext**

  Symbol whose name is "mtext".

2.5.1 Detailed Description

Property List objects and API for them.

A *property list* (or *plist* for short) is a list of zero or more properties. A property consists of a *key* and a *value*, where key is a symbol and value is anything that can be cast to `(void *)`.

If the key of a property is a *managing key*, its *value* is a *managed object*. A property list itself is a managed objects.

If each key of a plist is one of **Msymbol** (p. 17), **Mtext** (p. 23), **Minteger** (p. 23), and **Mplist** (p. 23), the plist is called as *well-formed* and represented by the following notation in the documentation.

```
PLIST ::= '(' ELEMENT * ')'

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

M-TEXT ::= '"' text data ... '"'
```

For instance, if a plist has four elements; integer -20, symbol of name "sym", M-text of contents "abc", and plist of integer 10 and symbol of name "another-symbol", it is represented as this:

(-20 sym "abc" (10 another-symbol))

2.5.2 Typedef Documentation

2.5.2.1 typedef struct MPlist MPlist

Type of property list objects.

The type **MPlist** (p. 19) is for a *property list* object. Its internal structure is concealed from application programs.

2.5.3 Function Documentation

2.5.3.1 MPlist $*$ mplist_deserialize (MText $*$ *mt*)

Generate a property list by deserializing an M-text.

The **mplist_deserialize()** (p. 20) function parses M-text **mt** and returns a property list.

The syntax of **mt** is as follows.

MT ::= '(' ELEMENT $*$ ')'

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

SYMBOL ::= ascii-character-sequence

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

M-TEXT ::= '"' character-sequence '"'

Each alternatives of `ELEMENT` is assigned one of these keys: `Msymbol`, `Minteger`, `Mtext`, `Mplist`

In an ascii-character-sequence, a backslash (\) is used as the escape character, which means that, for instance, `abc\ def` produces a symbol whose name is of length seven with the fourth character being a space.

2.5.3.2 MPlist$*$ mplist (void)

Create a property list object.

The **mplist()** (p. 20) function returns a newly created property list object of length zero.

Return value:
This function returns a newly created property list.

Errors:
This function never fails.

2.5.3.3 MPlist$*$ mplist_copy (MPlist $*$ *plist*)

Copy a property list.

The **mplist_copy()** (p. 20) function copies property list **plist**. In the copy, the values are the same as those of **plist**.

Return value:
This function returns a newly created plist which is a copy of **plist**.

Errors:
This function never fails.

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

Set the value of a property in a property list.

The **mplist_put()** (p. 20) function searches property list **plist** from the beginning for a property whose key is **key**. If such a property is found, its value is changed to **value**. Otherwise, a new property whose key is **key** and value is **value** is appended at the end of **plist**. See the documentation of **mplist_add()** (p. 21) for the restriction on **key** and **val**.

If **key** is a managing key, **val** must be a managed object. In this case, the reference count of the old value, if not `NULL`, is decremented by one, and that of **val** is incremented by one.

Return value:

If the operation was successful, **mplist_put()** (p. 20) returns a sublist of **plist** whose first element is the just modified or added one. Otherwise, it returns `NULL`.

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

Get the value of a property in a property list.

The **mplist_get()** (p. 21) function searches property list **plist** from the beginning for a property whose key is **key**. If such a property is found, its value is returned as the type of `(void *)`. If not found, `NULL` is returned.

When `NULL` is returned, there are two possibilities: one is the case where no property is found (see above); the other is the case where a property is found and its value is `NULL`. In case that these two cases must be distinguished, use the **mplist_find_by_key()** (p. 22) function.

See Also:

mplist_find_by_key() (p. 22)

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

Set the value (function pointer) of a property in a property list.

The **mplist_put_func()** (p. 21) function is similar to **mplist_put()** (p. 20) but for setting function pointer **func** in property list **plist** for key **key**. **key** must not be a managing key.

See Also:

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

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

Get the value (function pointer) of a property in a property list.

The **mplist_get_func()** (p. 21) function is similar to **mplist_get()** (p. 21) but for getting a function pointer from property list **plist** by key **key**.

See Also:

mplist_get() (p. 21)

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

Add a property at the end of a property list.

The **mplist_add()** (p. 21) function appends at the end of property list **plist** a property whose key is **key** and value is **val**. **key** can be any symbol other than `Mnil`.

If **key** is a managing key, **val** must be a managed object. In this case, the reference count of **val** is incremented by one.

Return value:

If the operation was successful, **mplist_add()** (p. 21) returns a sublist of **plist** whose first element is the just added one. Otherwise, it returns `NULL`.

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

Add a property at the beginning of a property list.

The **mplist_push()** (p. 22) function inserts at the beginning of property list **plist** a property whose key is **key** and value is **val**.

If **key** is a managing key, **val** must be a managed object. In this case, the reference count of **val** is incremented by one.

Return value:
If the operation was successful, this function returns **plist**. Otherwise, it returns `NULL`.

2.5.3.10 void∗ mplist_pop (MPlist ∗ *plist*)

Remove a property at the beginning of a property list.

The **mplist_pop()** (p. 22) function removes a property at the beginning of property list **plist**. As a result, the second key and value of the **plist** become the first ones.

Return value:
If the operation was successful, this function return the value of the just popped property. Otherwise, it returns `NULL`.

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

Find a property of a specific key in a property list.

The **mplist_find_by_key()** (p. 22) function searches property list **plist** from the beginning for a property whose key is **key**. If such a property is found, a sublist of **plist** whose first element is the found one is returned. Otherwise, `NULL` is returned.

If **key** is `Mnil`, it returns a sublist of **plist** whose first element is the last one of **plist**.

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

Find a property of a specific value in a property list.

The **mplist_find_by_value()** (p. 22) function searches property list **plist** from the beginning for a property whose value is **val**. If such a property is found, a sublist of **plist** whose first element is the found one is returned. Otherwise, `NULL` is returned.

2.5.3.13 MPlist∗ mplist_next (MPlist ∗ *plist*)

Return the next sublist of a property list.

The **mplist_next()** (p. 22) function returns a pointer to the sublist of property list **plist**, which begins at the second element in **plist**. If the length of **plist** is zero, it returns `NULL`.

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

Set the first property in a property list.

The **mplist_set()** (p. 22) function sets the key and the value of the first property in property list **plist** to **key** and **value**, respectively. See the documentation of **mplist_add()** (p. 21) for the restriction on **key** and **val**.

Return value:

If the operation was successful, **mplist_set()** (p. 22) returns **plist**. Otherwise, it returns `NULL`.

2.5.3.15 int mplist_length (MPlist ∗ *plist*)

Return the length of a property list.

The **mplist_length()** (p. 23) function returns the number of properties in property list **plist**.

2.5.3.16 MSymbol mplist_key (MPlist ∗ *plist*)

Return the key of the first property in a property list.

The **mplist_key()** (p. 23) function returns the key of the first property in property list **plist**. If the length of **plist** is zero, it returns `Mnil`.

2.5.3.17 void∗ mplist_value (MPlist ∗ *plist*)

Return the value of the first property in a property list.

The **mplist_value()** (p. 23) function returns the value of the first property in property list **plist**. If the length of **plist** is zero, it returns `NULL`.

2.5.4 Variable Documentation

2.5.4.1 MSymbol Minteger

Symbol whose name is "integer".

The symbol `Minteger` has the name `"integer"`. The value of a property whose key is `Minteger` must be an integer.

2.5.4.2 MSymbol Mplist

Symbol whose name is "plist".

The symbol `Mplist` has the name `"plist"`. It is a managing key. A value of a property whose key is `Mplist` must be a plist.

2.5.4.3 MSymbol Mtext

Symbol whose name is "mtext".

The symbol `Mtext` has the name `"mtext"`. It is a managing key. A value of a property whose key is `Mtext` must be an M-text.

2.6 Character

Character objects and API for them.

Variables: Keys of character properties

These symbols are used as keys of character properties.

- **MSymbol Mscript**

  Key for script.

- **MSymbol Mname**

  Key for character name.

- **MSymbol Mcategory**

  Key for general category.

- **MSymbol Mcombining_class**

  Key for canonical combining class.

- **MSymbol Mbidi_category**

  Key for bidi category.

- **MSymbol Msimple_case_folding**

  Key for corresponding single lowercase character.

- **MSymbol Mcomplicated_case_folding**

  Key for corresponding multiple lowercase characters.

- **MSymbol Mcased**

  Key for values used in case operation.

- **MSymbol Msoft_dotted**

  Key for values used in case operation.

- **MSymbol Mcase_mapping**

  Key for values used in case operation.

- **MSymbol Mblock**

  Key for script block name.

Defines

- #define **MCHAR_MAX**

  Maximum character code.

Functions

- **MSymbol mchar_define_property** (const char ∗name, **MSymbol** type)
  Define a character property.
- void ∗ **mchar_get_prop** (int c, **MSymbol** key)
  Get the value of a character property.
- int **mchar_put_prop** (int c, **MSymbol** key, void ∗val)
  Set the value of a character property.
- **MCharTable** ∗ **mchar_get_prop_table** (**MSymbol** key, **MSymbol** ∗type)
  Get the char-table for a character property.

2.6.1 Detailed Description

Character objects and API for them.

The m17n library represents a *character* by a character code (an integer). The minimum character code is `0`. The maximum character code is defined by the macro **MCHAR_MAX** (p. 25). It is assured that **MCHAR_MAX** (p. 25) is not smaller than `0x3FFFFF` (22 bits).

Characters `0` to `0x10FFFF` are equivalent to the Unicode characters of the same code values.

A character can have zero or more properties called *character properties*. A character property consists of a *key* and a *value*, where key is a symbol and value is anything that can be cast to `(void *)`. "The character property that belongs to character C and whose key is K" may be shortened to "the K property of C".

2.6.2 Define Documentation

2.6.2.1 #define MCHAR_MAX

Maximum character code.

The macro **MCHAR_MAX** (p. 25) gives the maximum character code.

2.6.3 Function Documentation

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

Define a character property.

The **mchar_define_property()** (p. 25) function searches the m17n database for a data whose tags are <**Mchar_table** (p. 32), **type**, **sym** >. Here, **sym** is a symbol whose name is **name**. **type** must be **Mstring** (p. 17), **Mtext** (p. 23), **Msymbol** (p. 17), **Minteger** (p. 23), or **Mplist** (p. 23).

Return value:

If the operation was successful, **mchar_define_property()** (p. 25) returns **sym**. Otherwise it returns **Mnil** (p. 17).

Errors:

`MERROR_DB`

See Also:

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

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

Get the value of a character property.

The **mchar_get_prop()** (p. 26) function searches character **c** for the character property whose key is **key**.

Return value:

If the operation was successful, **mchar_get_prop()** (p. 26) returns the value of the character property. Otherwise it returns `NULL`.

Errors:

`MERROR_SYMBOL`, `MERROR_DB`

See Also:

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

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

Set the value of a character property.

The **mchar_put_prop()** (p. 26) function searches character **c** for the character property whose key is **key** and assigns **val** to the value of the found property.

Return value:

If the operation was successful, **mchar_put_prop()** (p. 26) returns 0. Otherwise, it returns -1.

Errors:

`MERROR_SYMBOL`, `MERROR_DB`

See Also:

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

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

Get the char-table for a character property.

The **mchar_get_prop_table()** (p. 26) function returns a char-table that contains the character property whose key is **key**. If **type** is not NULL, this function stores the type of the property in the place pointed by **type**. See **mchar_define_property()** (p. 25) for types of character property.

Return value:

If **key** is a valid character property key, this function returns a char-table. Otherwise NULL is retuned.

2.6.4 Variable Documentation

2.6.4.1 MSymbol Mscript

Key for script.

The symbol **Mscript** (p. 26) has the name `"script"` and is used as the key of a character property. The value of such a property is a symbol representing the script to which the character belongs.

Each symbol that represents a script has one of the names listed in the *Unicode Technical Report #24*.

2.6.4.2 MSymbol Mname

Key for character name.

The symbol **Mname** (p. 27) has the name `"name"` and is used as the key of a character property. The value of such a property is a C-string representing the name of the character.

2.6.4.3 MSymbol Mcategory

Key for general category.

The symbol **Mcategory** (p. 27) has the name `"category"` and is used as the key of a character property. The value of such a property is a symbol representing the *general category* of the character.

Each symbol that represents a general category has one of the names listed as abbreviations for *General Category* in Unicode.

2.6.4.4 MSymbol Mcombining_class

Key for canonical combining class.

The symbol **Mcombining_class** (p. 27) has the name `"combining-class"` and is used as the key of a character property. The value of such a property is an integer that represents the *canonical combining class* of the character.

The meaning of each integer that represents a canonical combining class is identical to the one defined in Unicode.

2.6.4.5 MSymbol Mbidi_category

Key for bidi category.

The symbol **Mbidi_category** (p. 27) has the name `"bidi-category"` and is used as the key of a character property. The value of such a property is a symbol that represents the *bidirectional category* of the character.

Each symbol that represents a bidirectional category has one of the names listed as types of *Bidirectional Category* in Unicode.

2.6.4.6 MSymbol Msimple_case_folding

Key for corresponding single lowercase character.

The symbol **Msimple_case_folding** (p. 27) has the name `"simple-case-folding"` and is used as the key of a character property. The value of such a property is the corresponding single lowercase character that is used when comparing M-texts ignoring cases.

If a character requires a complicated comparison (i.e. cannot be compared by simply mapping to another single character), the value of such a property is `0xFFFF`. In this case, the character has another property whose key is **Mcomplicated_case_folding** (p. 27).

2.6.4.7 MSymbol Mcomplicated_case_folding

Key for corresponding multiple lowercase characters.

The symbol **Mcomplicated_case_folding** (p. 27) has the name `"complicated-case-folding"` and is used as the key of a character property. The value of such a property is the corresponding M-text that contains a sequence of lowercase characters to be used for comparing M-texts ignoring case.

2.6.4.8 MSymbol Mcased

Key for values used in case operation.

The symbol **Mcased** (p. 28) has the name `"cased"` and is used as the key of charater property. The value of such a property is an integer value 1, 2, or 3 representing "cased", "case-ignorable", and both of them respective. See the Unicode Standard 5.0 (Section 3.13 Default Case Algorithm) for the detail.

2.6.4.9 MSymbol Msoft_dotted

Key for values used in case operation.

The symbol **Msoft_dotted** (p. 28) has the name `"soft-dotted"` and is used as the key of charater property. The value of such a property is **Mt** (p. 17) if a character has "Soft_Dotted" property, and **Mnil** (p. 17) otherwise. See the Unicode Standard 5.0 (Section 3.13 Default Case Algorithm) for the detail.

2.6.4.10 MSymbol Mcase_mapping

Key for values used in case operation.

The symbol **Mcase_mapping** (p. 28) has the name `"case-mapping"` and is used as the key of charater property. The value of such a property is a plist of three M-Texts; lower, title, and upper of the corresponding character. See the Unicode Standard 5.0 (Section 5.18 Case Mappings) for the detail.

2.6.4.11 MSymbol Mblock

Key for script block name.

The symbol **Mblock** (p. 28) the name `"block"` and is used as the key of charater property. The value of such a property is a symbol representing a script block of the corresponding character.

2.7 Chartable

Chartable objects and API for them.

Typedefs

- typedef struct **MCharTable MCharTable**

  Type of chartables.

Functions

- **MCharTable** ∗ **mchartable** (**MSymbol** key, void ∗default_value)

  Create a new chartable.

- int **mchartable_min_char** (**MCharTable** ∗table)

  Return the minimum character whose value is set in a chartabe.

- int **mchartable_max_char** (**MCharTable** ∗table)

  Return the maximum character whose value is set in a chartabe.

- void ∗ **mchartable_lookup** (**MCharTable** ∗table, int c)

  Return the assigned value of a character in a chartable.

- int **mchartable_set** (**MCharTable** ∗table, int c, void ∗val)

  Assign a value to a character in a chartable.

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

  Assign a value to the characters in the specified range.

- void **mchartable_range** (**MCharTable** ∗table, int ∗from, int ∗to)

  Search for characters that have non-default value.

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

  Call a function for characters in a chartable.

Variables

- **MSymbol Mchar_table**

  Symbol whose name is "char-table".

2.7.1 Detailed Description

Chartable objects and API for them.

The m17n library supports enormous number of characters. Thus, if attributes of each character are to be stored in a simple array, such an array would be impractically big. The attributes usually used, however, are often assigned only to a range of characters. Even when all characters have attributes, characters of consecutive character code tend to have the same attribute values.

The m17n library utilizes this tendency to store characters and their attribute values efficiently in an object called *Chartable*. Although a chartable object is not a simple array, application programs can handle a chartable as if it is an array. Attribute values of a character can be obtained by accessing a Chartable for the attribute with the character code of the specified character.

A chartable is a managed object.

2.7.2 Typedef Documentation

2.7.2.1 typedef struct MCharTable MCharTable

Type of chartables.

The type **MCharTable** (p. 30) is for a *chartable* objects. Its internal structure is concealed from application programs.

2.7.3 Function Documentation

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

Create a new chartable.

The **mchartable()** (p. 30) function creates a new chartable object with symbol **key** and the default value **default_value**. If **key** is a managing key, the elements of the table (including the default value) are managed objects or NULL.

Return value:

If the operation was successful, **mchartable()** (p. 30) returns a pointer to the created chartable. Otherwise it returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

2.7.3.2 int mchartable_min_char (MCharTable ∗ *table*)

Return the minimum character whose value is set in a chartabe.

The **mchartable_min_char()** (p. 30) function return the minimum character whose value is set in chartable **table**. No character is set its value, the function returns -1.

2.7.3.3 int mchartable_max_char (MCharTable ∗ *table*)

Return the maximum character whose value is set in a chartabe.

The **mchartable_max_char()** (p. 30) function return the maximum character whose value is set in chartable **table**. No character is set its value, the function returns -1.

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

Return the assigned value of a character in a chartable.

The **mchartable_lookup()** (p. 30) function returns the value assigned to character **c** in chartable **table**. If no value has been set for **c** explicitly, the default value of **table** is returned. If **c** is not a valid character, **mchartable_lookup()** (p. 30) returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

MERROR_CHAR

See Also:

mchartable_set() (p. 31)

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

Assign a value to a character in a chartable.

The **mchartable_set()** (p. 31) function sets the value of character **c** in chartable **table** to **val**.

Return value:

If the operation was successful, **mchartable_set()** (p. 31) returns 0. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

MERROR_CHAR

See Also:

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*)

Assign a value to the characters in the specified range.

The **mchartable_set_range()** (p. 31) function assigns value **val** to the characters from **from** to **to** (both inclusive) in chartable **table**.

Return value:

If the operation was successful, **mchartable_set_range()** (p. 31) returns 0. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156). If **from** is greater than **to**, **mchartable_set_range()** (p. 31) returns immediately without an error.

Errors:

MERROR_CHAR

See Also:

mchartable_set() (p. 31)

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

Search for characters that have non-default value.

The **mchartable_range()** (p. 31) function searches chartable **table** for the first and the last character codes that do not have the default value of **table**, and set **from** and **to** to them, respectively. If all characters have the default value, both **from** and **to** are set to -1.

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

Call a function for characters in a chartable.

The **mchartable_map()** (p. 32) function calls function **func** for characters in chartable **table**. No function call occurs for characters that have value **ignore** in **table**. Comparison of **ignore** and character value is done with the operator ==. Be careful when you use string literals or pointers.

Instead of calling **func** for each character, **mchartable_map()** (p. 32) tries to optimize the number of function calls, i.e. it makes a single function call for a chunk of characters when those consecutive characters have the same value.

No matter how long the character chunk is, **func** is called with four arguments; **from**, **to**, **val**, and **arg**. **from** and **to** (both inclusive) defines the range of characters that have value **val**. **arg** is the same as **func_arg**.

Return value:

This function always returns 0.

2.7.4 Variable Documentation

2.7.4.1 MSymbol Mchar_table

Symbol whose name is "char-table".

The symbol `Mchar_table` has the name `"char-table"`.

2.8 M-text

M-text objects and API for them.

Variables: Default Endian of UTF-16 and UTF-32

- enum **MTextFormat** **MTEXT_FORMAT_UTF_16**

  Variable of value MTEXT_FORMAT_UTF_16LE or MTEXT_FORMAT_UTF_16BE.

- const int **MTEXT_FORMAT_UTF_32**

  Variable of value MTEXT_FORMAT_UTF_32LE or MTEXT_FORMAT_UTF_32BE.

Typedefs

- typedef struct **MText** **MText**

  Type of M-texts.

Enumerations

- 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 }

  Enumeration for specifying the format of an M-text.

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

  Enumeration for specifying a set of line breaking option.

Functions

- int **mtext_line_break** (**MText** ∗mt, int pos, int option, int ∗after)

  Find a linebreak postion of an M-text.

- **MText** ∗ **mtext** ()

  Allocate a new M-text.

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

  Allocate a new M-text with specified data.

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

  Get information about the text data in M-text.

- int **mtext_len** (**MText** ∗mt)

  Number of characters in M-text.

- int **mtext_ref_char** (**MText** ∗mt, int pos)

  Return the character at the specified position in an M-text.

- int **mtext_set_char** (**MText** ∗mt, int pos, int c)

  Store a character into an M-text.

- **MText** ∗ **mtext_cat_char** (**MText** ∗mt, int c)

  Append a character to an M-text.

- **MText** ∗ **mtext_dup** (**MText** ∗mt)

  Create a copy of an M-text.

- **MText** ∗ **mtext_cat** (**MText** ∗mt1, **MText** ∗mt2)

  Append an M-text to another.

- **MText** ∗ **mtext_ncat** (**MText** ∗mt1, **MText** ∗mt2, int n)

  Append a part of an M-text to another.

- **MText** ∗ **mtext_cpy** (**MText** ∗mt1, **MText** ∗mt2)

  Copy an M-text to another.

- **MText** ∗ **mtext_ncpy** (**MText** ∗mt1, **MText** ∗mt2, int n)

  Copy the first some characters in an M-text to another.

- **MText** ∗ **mtext_duplicate** (**MText** ∗mt, int from, int to)

  Create a new M-text from a part of an existing M-text.

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

  Copy characters in the specified range into an M-text.

- int **mtext_del** (**MText** ∗mt, int from, int to)

  Delete characters in the specified range destructively.

- int **mtext_ins** (**MText** ∗mt1, int pos, **MText** ∗mt2)

  Insert an M-text into another M-text.

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

  Insert sub-text of an M-text into another M-text.

- int **mtext_ins_char** (**MText** ∗mt, int pos, int c, int n)

  Insert a character into an M-text.

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

Replace sub-text of M-text with another.

- int **mtext_character** (**MText** ∗mt, int from, int to, int c)

  Search a character in an M-text.

- int **mtext_chr** (**MText** ∗mt, int c)

  Return the position of the first occurrence of a character in an M-text.

- int **mtext_rchr** (**MText** ∗mt, int c)

  Return the position of the last occurrence of a character in an M-text.

- int **mtext_cmp** (**MText** ∗mt1, **MText** ∗mt2)

  Compare two M-texts character-by-character.

- int **mtext_ncmp** (**MText** ∗mt1, **MText** ∗mt2, int n)

  Compare initial parts of two M-texts character-by-character.

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

  Compare specified regions of two M-texts.

- int **mtext_spn** (**MText** ∗mt, **MText** ∗accept)

  Search an M-text for a set of characters.

- int **mtext_cspn** (**MText** ∗mt, **MText** ∗reject)

  Search an M-text for the complement of a set of characters.

- int **mtext_pbrk** (**MText** ∗mt, **MText** ∗accept)

  Search an M-text for any of a set of characters.

- **MText** ∗ **mtext_tok** (**MText** ∗mt, **MText** ∗delim, int ∗pos)

  Look for a token in an M-text.

- int **mtext_text** (**MText** ∗mt1, int pos, **MText** ∗mt2)

  Locate an M-text in another.

- int **mtext_search** (**MText** ∗mt1, int from, int to, **MText** ∗mt2)

  Locate an M-text in a specific range of another.

- int **mtext_casecmp** (**MText** ∗mt1, **MText** ∗mt2)

  Compare two M-texts ignoring cases.

- int **mtext_ncasecmp** (**MText** ∗mt1, **MText** ∗mt2, int n)

  Compare initial parts of two M-texts ignoring cases.

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

  Compare specified regions of two M-texts ignoring cases.

- int **mtext_lowercase** (**MText** ∗mt)

  Lowercase an M-text.

- int **mtext_titlecase** (**MText** ∗mt)

  Titlecase an M-text.

- int **mtext_uppercase** (**MText** ∗mt)
  Uppercase an M-text.

Variables

- **MSymbol Mlanguage**

2.8.1 Detailed Description

M-text objects and API for them.

In the m17n library, text is represented as an object called *M-text* rather than as a C-string (`char *` or `unsigned char *`). An M-text is a sequence of characters whose length is equals to or more than 0, and can be coined from various character sources, e.g. C-strings, files, character codes, etc.

M-texts are more useful than C-strings in the following points.

- M-texts can handle mixture of characters of various scripts, including all Unicode characters and more. This is an indispensable facility when handling multilingual text.
- Each character in an M-text can have properties called *text properties*. Text properties store various kinds of information attached to parts of an M-text to provide application programs with a unified view of those information. As rich information can be stored in M-texts in the form of text properties, functions in application programs can be simple.

In addition, the library provides many functions to manipulate an M-text just the same way as a C-string.

2.8.2 Typedef Documentation

2.8.2.1 typedef struct MText MText

Type of *M-texts*.

The type **MText** (p. 36) is for an *M-text* object. Its internal structure is concealed from application programs.

2.8.3 Enumeration Type Documentation

2.8.3.1 enum MTextFormat

Enumeration for specifying the format of an M-text.

The enum **MTextFormat** (p. 36) is used as an argument of the **mtext_from_data()** (p. 37) function to specify the format of data from which an M-text is created.

Enumerator:

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

Enumeration for specifying a set of line breaking option.

The enum **MTextLineBreakOption** (p. 37) is to control the line breaking algorithm of the function **mtext_line_break**() (p. 37) by specifying logical-or of the members in the arg *option*.

Enumerator:

MTEXT_LBO_SP_CM Specify the legacy support for space character as base for combining marks. See the section 8.3 of UAX#14.

MTEXT_LBO_KOREAN_SP Specify to use space characters for line breaking Korean text.

MTEXT_LBO_AI_AS_ID Specify to treat characters of ambiguous line-breaking class as of ideographic line-breaking class.

MTEXT_LBO_MAX

2.8.4 Function Documentation

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

Find a linebreak postion of an M-text.

The **mtext_line_break**() (p. 37) function checks if position **pos** is a proper linebreak position of an M-text **mt** according to the algorithm of The Unicode Standard 4.0 UAX#14. It so, it returns **pos**. Otherwise, it returns a proper linebreak position before **pos**.

If **option** is nonzero, it controls the algorithm by logical-or of the members of **MTextLineBreakOption** (p. 37).

If **after** is not NULL, a proper linebreak position after **pos** is stored there.

2.8.4.2 MText∗ mtext ()

Allocate a new M-text.

The **mtext**() (p. 37) function allocates a new M-text of length 0 and returns a pointer to it. The allocated M-text will not be freed unless the user explicitly does so with the **m17n_object_unref**() (p. 12) function.

See Also:

m17n_object_unref() (p. 12)

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

Allocate a new M-text with specified data.

The **mtext_from_data**() (p. 37) function allocates a new M-text whose character sequence is specified by array **data** of **nitems** elements. **format** specifies the format of **data**.

When **format** is either **MTEXT_FORMAT_US_ASCII** (p. 36) or **MTEXT_FORMAT_UTF_8** (p. 36), the contents of **data** must be of the type `unsigned char`, and **nitems** counts by byte.

When **format** is either **MTEXT_FORMAT_UTF_16LE** (p. 36) or **MTEXT_FORMAT_UTF_16BE** (p. 36), the contents of **data** must be of the type `unsigned short`, and **nitems** counts by unsigned short.

When **format** is either **MTEXT_FORMAT_UTF_32LE** (p. 36) or **MTEXT_FORMAT_UTF_32BE** (p. 36), the contents of **data** must be of the type `unsigned`, and **nitems** counts by unsigned.

The character sequence of the M-text is not modifiable. The contents of **data** must not be modified while the M-text is alive.

The allocated M-text will not be freed unless the user explicitly does so with the **m17n_object_unref()** (p. 12) function. Even in that case, **data** is not freed.

Return value:

If the operation was successful, **mtext_from_data()** (p. 37) returns a pointer to the allocated M-text. Otherwise it returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_MTEXT`

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

Get information about the text data in M-text.

The **mtext_data()** (p. 38) function returns a pointer to the text data of M-text **mt**. If **fmt** is not NULL, the format of the text data is stored in it. If **nunits** is not NULL, the number of units of the text data is stored in it.

If **pos_idx** is not NULL and it points to a non-negative number, what it points to is a character position. In this case, the return value is a pointer to the text data of a character at that position.

Otherwise, if **unit_idx** is not NULL, it points to a unit position. In this case, the return value is a pointer to the text data of a character containing that unit.

The character position and unit position of the return value are stored in **pos_idx** and **unit_dix** respectively if they are not NULL.

- If the format of the text data is MTEXT_FORMAT_US_ASCII or MTEXT_FORMAT_UTF_8, one unit is unsigned char.
- If the format is MTEXT_FORMAT_UTF_16LE or MTEXT_FORMAT_UTF_16BE, one unit is unsigned short.
- If the format is MTEXT_FORMAT_UTF_32LE or MTEXT_FORMAT_UTF_32BE, one unit is unsigned int.

2.8.4.5 int mtext_len (MText ∗ *mt*)

Number of characters in M-text.

The **mtext_len()** (p. 38) function returns the number of characters in M-text **mt**.

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

Return the character at the specified position in an M-text.

The **mtext_ref_char()** (p. 38) function returns the character at **pos** in M-text **mt**. If an error is detected, it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`

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

Store a character into an M-text.

The **mtext_set_char()** (p. 39) function sets character **c**, which has no text properties, at **pos** in M-text **mt**.

Return value:

If the operation was successful, **mtext_set_char()** (p. 39) returns 0. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`

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

Append a character to an M-text.

The **mtext_cat_char()** (p. 39) function appends character **c**, which has no text properties, to the end of M-text **mt**.

Return value:

This function returns a pointer to the resulting M-text **mt**. If **c** is an invalid character, it returns `NULL`.

See Also:

mtext_cat() (p. 39), **mtext_ncat()** (p. 40)

2.8.4.9 MText∗ mtext_dup (MText ∗ *mt*)

Create a copy of an M-text.

The **mtext_dup()** (p. 39) function creates a copy of M-text **mt** while inheriting all the text properties of **mt**.

Return value:

This function returns a pointer to the created copy.

See Also:

mtext_duplicate() (p. 40)

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

Append an M-text to another.

The **mtext_cat()** (p. 39) function appends M-text **mt2** to the end of M-text **mt1** while inheriting all the text properties. **mt2** itself is not modified.

Return value:

This function returns a pointer to the resulting M-text **mt1**.

See Also:

mtext_ncat() (p. 40), **mtext_cat_char()** (p. 39)

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

Append a part of an M-text to another.

The **mtext_ncat()** (p. 40) function appends the first **n** characters of M-text **mt2** to the end of M-text **mt1** while inheriting all the text properties. If the length of **mt2** is less than **n**, all characters are copied. **mt2** is not modified.

Return value:

If the operation was successful, **mtext_ncat()** (p. 40) returns a pointer to the resulting M-text **mt1**. If an error is detected, it returns `NULL` and assigns an error code to the global variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`

See Also:

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

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

Copy an M-text to another.

The **mtext_cpy()** (p. 40) function copies M-text **mt2** to M-text **mt1** while inheriting all the text properties. The old text in **mt1** is overwritten and the length of **mt1** is extended if necessary. **mt2** is not modified.

Return value:

This function returns a pointer to the resulting M-text **mt1**.

See Also:

mtext_ncpy() (p. 40), **mtext_copy()** (p. 41)

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

Copy the first some characters in an M-text to another.

The **mtext_ncpy()** (p. 40) function copies the first **n** characters of M-text **mt2** to M-text **mt1** while inheriting all the text properties. If the length of **mt2** is less than **n**, all characters of **mt2** are copied. The old text in **mt1** is overwritten and the length of **mt1** is extended if necessary. **mt2** is not modified.

Return value:

If the operation was successful, **mtext_ncpy()** (p. 40) returns a pointer to the resulting M-text **mt1**. If an error is detected, it returns `NULL` and assigns an error code to the global variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`

See Also:

mtext_cpy() (p. 40), **mtext_copy()** (p. 41)

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

Create a new M-text from a part of an existing M-text.

The **mtext_duplicate()** (p. 40) function creates a copy of sub-text of M-text **mt**, starting at **from** (inclusive) and ending at **to** (exclusive) while inheriting all the text properties of **mt**. **mt** itself is not modified.

Return value:

If the operation was successful, **mtext_duplicate()** (p. 40) returns a pointer to the created M-text. If an error is detected, it returns NULL and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`

See Also:

mtext_dup() (p. 39)

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

Copy characters in the specified range into an M-text.

The **mtext_copy()** (p. 41) function copies the text between **from** (inclusive) and **to** (exclusive) in M-text **mt2** to the region starting at **pos** in M-text **mt1** while inheriting the text properties. The old text in **mt1** is overwritten and the length of **mt1** is extended if necessary. **mt2** is not modified.

Return value:

If the operation was successful, **mtext_copy()** (p. 41) returns a pointer to the modified **mt1**. Otherwise, it returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`

See Also:

mtext_cpy() (p. 40), **mtext_ncpy()** (p. 40)

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

Delete characters in the specified range destructively.

The **mtext_del()** (p. 41) function deletes the characters in the range **from** (inclusive) and **to** (exclusive) from M-text **mt** destructively. As a result, the length of **mt** shrinks by (**to** - **from**) characters.

Return value:

If the operation was successful, **mtext_del()** (p. 41) returns 0. Otherwise, it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`

See Also:

mtext_ins() (p. 41)

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

Insert an M-text into another M-text.

The **mtext_ins()** (p. 41) function inserts M-text **mt2** into M-text **mt1**, at position **pos**. As a result, **mt1** is lengthen by the length of **mt2**. On insertion, all the text properties of **mt2** are inherited. The original **mt2** is not modified.

Return value:

If the operation was successful, **mtext_ins()** (p. 41) returns 0. Otherwise, it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE` , `MERROR_MTEXT`

See Also:

mtext_del() (p. 41) , **mtext_insert()** (p. 42)

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

Insert sub-text of an M-text into another M-text.

The **mtext_insert()** (p. 42) function inserts sub-text of M-text **mt2** between **from** (inclusive) and **to** (exclusive) into M-text **mt1**, at position **pos**. As a result, **mt1** is lengthen by (**to** - **from**). On insertion, all the text properties of the sub-text of **mt2** are inherited.

Return value:

If the operation was successful, **mtext_insert()** (p. 42) returns 0. Otherwise, it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_MTEXT` , `MERROR_RANGE`

See Also:

mtext_ins() (p. 41)

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

Insert a character into an M-text.

The **mtext_ins_char()** (p. 42) function inserts **n** copies of character **c** into M-text **mt** at position **pos**. As a result, **mt** is lengthen by **n**.

Return value:

If the operation was successful, **mtext_ins()** (p. 41) returns 0. Otherwise, it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`

See Also:

mtext_ins, **mtext_del()** (p. 41)

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

Replace sub-text of M-text with another.

The **mtext_replace()** (p. 42) function replaces sub-text of M-text **mt1** between **from1** (inclusive) and **to1** (exclusive) with the sub-text of M-text **mt2** between **from2** (inclusive) and **to2** (exclusive). The new sub-text inherits text properties of the old sub-text.

Return value:

If the operation was successful, **mtext_replace()** (p. 42) returns 0. Otherwise, it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_MTEXT` , `MERROR_RANGE`

See Also:

mtext_insert() (p. 42)

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

Search a character in an M-text.

The **mtext_character()** (p. 43) function searches M-text **mt** for character **c**. If **from** is less than **to**, the search begins at position **from** and goes forward but does not exceed (**to** - 1). Otherwise, the search begins at position (**from** - 1) and goes backward but does not exceed **to**. An invalid position specification is regarded as both **from** and **to** being 0.

Return value:

If **c** is found, **mtext_character()** (p. 43) returns the position of its first occurrence. Otherwise it returns -1 without changing the external variable **merror_code** (p. 156). If an error is detected, it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

See Also:

mtext_chr() (p. 43), **mtext_rchr()** (p. 43)

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

Return the position of the first occurrence of a character in an M-text.

The **mtext_chr()** (p. 43) function searches M-text **mt** for character **c**. The search starts from the beginning of **mt** and goes toward the end.

Return value:

If **c** is found, **mtext_chr()** (p. 43) returns its position; otherwise it returns -1.

Errors:

`MERROR_RANGE`

See Also:

mtext_rchr() (p. 43), **mtext_character()** (p. 43)

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

Return the position of the last occurrence of a character in an M-text.

The **mtext_rchr()** (p. 43) function searches M-text **mt** for character **c**. The search starts from the end of **mt** and goes backwardly toward the beginning.

Return value:

If **c** is found, **mtext_rchr()** (p. 43) returns its position; otherwise it returns -1.

Errors:

`MERROR_RANGE`

See Also:

mtext_chr() (p. 43), **mtext_character()** (p. 43)

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

Compare two M-texts character-by-character.

The **mtext_cmp()** (p. 44) function compares M-texts **mt1** and **mt2** character by character.

Return value:

This function returns 1, 0, or -1 if **mt1** is found greater than, equal to, or less than **mt2**, respectively. Comparison is based on character codes.

See Also:

mtext_ncmp() (p. 44), **mtext_casecmp()** (p. 46), **mtext_ncasecmp()** (p. 46), **mtext_compare()** (p. 44), **mtext_case_compare()** (p. 46)

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

Compare initial parts of two M-texts character-by-character.

The **mtext_ncmp()** (p. 44) function is similar to **mtext_cmp()** (p. 44), but compares at most **n** characters from the beginning.

Return value:

This function returns 1, 0, or -1 if **mt1** is found greater than, equal to, or less than **mt2**, respectively.

See Also:

mtext_cmp() (p. 44), **mtext_casecmp()** (p. 46), **mtext_ncasecmp()** (p. 46) **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*)

Compare specified regions of two M-texts.

The **mtext_compare()** (p. 44) function compares two M-texts **mt1** and **mt2**, character-by-character. The compared regions are between **from1** and **to1** in **mt1** and **from2** to **to2** in MT2. **from1** and **from2** are inclusive, **to1** and **to2** are exclusive. **from1** being equal to **to1** (or **from2** being equal to **to2**) means an M-text of length zero. An invalid region specification is regarded as both **from1** and **to1** (or **from2** and **to2**) being 0.

Return value:

This function returns 1, 0, or -1 if **mt1** is found greater than, equal to, or less than **mt2**, respectively. Comparison is based on character codes.

See Also:

mtext_cmp() (p. 44), **mtext_ncmp()** (p. 44), **mtext_casecmp()** (p. 46), **mtext_ncasecmp()** (p. 46), **mtext_case_compare()** (p. 46)

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

Search an M-text for a set of characters.

The **mtext_spn()** (p. 45) function returns the length of the initial segment of M-text **mt1** that consists entirely of characters in M-text **mt2**.

See Also:

mtext_cspn() (p. 45)

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

Search an M-text for the complement of a set of characters.

The **mtext_cspn()** (p. 45) returns the length of the initial segment of M-text **mt1** that consists entirely of characters not in M-text **mt2**.

See Also:

mtext_spn() (p. 45)

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

Search an M-text for any of a set of characters.

The **mtext_pbrk()** (p. 45) function locates the first occurrence in M-text **mt1** of any of the characters in M-text **mt2**.

Return value:

This function returns the position in **mt1** of the found character. If no such character is found, it returns -1.

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

Look for a token in an M-text.

The **mtext_tok()** (p. 45) function searches a token that firstly occurs after position **pos** in M-text **mt**. Here, a token means a substring each of which does not appear in M-text **delim**. Note that the type of **pos** is not `int` but pointer to `int`.

Return value:

If a token is found, **mtext_tok()** (p. 45) copies the corresponding part of **mt** and returns a pointer to the copy. In this case, **pos** is set to the end of the found token. If no token is found, it returns `NULL` without changing the external variable **merror_code** (p. 156). If an error is detected, it returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`

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

Locate an M-text in another.

The **mtext_text()** (p. 45) function finds the first occurrence of M-text **mt2** in M-text **mt1** after the position **pos** while ignoring difference of the text properties.

Return value:

If **mt2** is found in **mt1**, **mtext_text()** (p. 45) returns the position of it first occurrence. Otherwise it returns -1. If **mt2** is empty, it returns 0.

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

Locate an M-text in a specific range of another.

The **mtext_search()** (p. 46) function searches for the first occurrence of M-text **mt2** in M-text **mt1** in the region **from** and **to** while ignoring difference of the text properties. If **from** is less than **to**, the forward search starts from **from**, otherwise the backward search starts from **to**.

Return value:

If **mt2** is found in **mt1**, **mtext_search()** (p. 46) returns the position of the first occurrence. Otherwise it returns -1. If **mt2** is empty, it returns 0.

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

Compare two M-texts ignoring cases.

The **mtext_casecmp()** (p. 46) function is similar to **mtext_cmp()** (p. 44), but ignores cases on comparison.

Return value:

This function returns 1, 0, or -1 if **mt1** is found greater than, equal to, or less than **mt2**, respectively.

See Also:

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

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

Compare initial parts of two M-texts ignoring cases.

The **mtext_ncasecmp()** (p. 46) function is similar to **mtext_casecmp()** (p. 46), but compares at most **n** characters from the beginning.

Return value:

This function returns 1, 0, or -1 if **mt1** is found greater than, equal to, or less than **mt2**, respectively.

See Also:

mtext_cmp() (p. 44), **mtext_casecmp()** (p. 46), **mtext_casecmp()** (p. 46) **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*)

Compare specified regions of two M-texts ignoring cases.

The **mtext_case_compare()** (p. 46) function compares two M-texts **mt1** and **mt2**, character-by-character, ignoring cases. The compared regions are between **from1** and **to1** in **mt1** and **from2** to **to2** in MT2. **from1** and **from2** are inclusive, **to1** and **to2** are exclusive. **from1** being equal to **to1** (or **from2** being equal to **to2**) means an M-text of length zero. An invalid region specification is regarded as both **from1** and **to1** (or **from2** and **to2**) being 0.

Return value:

This function returns 1, 0, or -1 if **mt1** is found greater than, equal to, or less than **mt2**, respectively. Comparison is based on character codes.

See Also:

mtext_cmp() (p. 44), **mtext_ncmp()** (p. 44), **mtext_casecmp()** (p. 46), **mtext_ncasecmp()** (p. 46), **mtext_compare()** (p. 44)

2.8.4.36 int mtext_lowercase (MText ∗ *mt*)

Lowercase an M-text.

The **mtext_lowercase()** (p. 47) function destructively converts each character in M-text **mt** to lowercase. Adjacent characters in **mt** may affect the case conversion. If the Mlanguage text property is attached to **mt**, it may also affect the conversion. The length of **mt** may change. Characters that cannot be converted to lowercase is left unchanged. All the text properties are inherited.

Return value:

This function returns the length of the updated **mt**.

See Also:

mtext_titlecase() (p. 47), **mtext_uppercase()** (p. 47)

2.8.4.37 int mtext_titlecase (MText ∗ *mt*)

Titlecase an M-text.

The **mtext_titlecase()** (p. 47) function destructively converts the first character with the cased property in M-text **mt** to titlecase and the others to lowercase. The length of **mt** may change. If the character cannot be converted to titlecase, it is left unchanged. All the text properties are inherited.

Return value:

This function returns the length of the updated **mt**.

See Also:

mtext_lowercase() (p. 47), **mtext_uppercase()** (p. 47)

2.8.4.38 int mtext_uppercase (MText ∗ *mt*)

Uppercase an M-text.

The **mtext_uppercase()** (p. 47) function destructively converts each character in M-text **mt** to uppercase. Adjacent characters in **mt** may affect the case conversion. If the Mlanguage text property is attached to **mt**, it may also affect the conversion. The length of **mt** may change. Characters that cannot be converted to uppercase is left unchanged. All the text properties are inherited.

Return value:

This function returns the length of the updated **mt**.

See Also:

mtext_lowercase() (p. 47), **mtext_titlecase()** (p. 47)

2.8.5 Variable Documentation

2.8.5.1 enum MTextFormat MTEXT_FORMAT_UTF_16

Variable of value MTEXT_FORMAT_UTF_16LE or MTEXT_FORMAT_UTF_16BE.

The global variable **MTEXT_FORMAT_UTF_16** (p. 48) is initialized to **MTEXT_FORMAT_UTF_16LE** (p. 36) on a "Little Endian" system (storing words with the least significant byte first), and to **MTEXT_FORMAT_UTF_16BE** (p. 36) on a "Big Endian" system (storing words with the most significant byte first).

See Also:
mtext_from_data() (p. 37)

2.8.5.2 const int MTEXT_FORMAT_UTF_32

Variable of value MTEXT_FORMAT_UTF_32LE or MTEXT_FORMAT_UTF_32BE.

The global variable **MTEXT_FORMAT_UTF_32** (p. 48) is initialized to **MTEXT_FORMAT_UTF_32LE** (p. 36) on a "Little Endian" system (storing words with the least significant byte first), and to **MTEXT_FORMAT_UTF_32BE** (p. 36) on a "Big Endian" system (storing words with the most significant byte first).

See Also:
mtext_from_data() (p. 37)

2.8.5.3 MSymbol Mlanguage

The symbol whose name is "language".

2.9 Text Property

Function to handle text properties.

Typedefs

- typedef **MPlist** ∗(∗ **MTextPropSerializeFunc**)(void ∗val)

  Type of serializer functions.

- typedef void ∗(∗ **MTextPropDeserializeFunc**)(**MPlist** ∗plist)

  Type of deserializer functions.

- typedef struct **MTextProperty** **MTextProperty**

  Type of text properties.

Enumerations

- 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 }

  Flag bits to control text property.

Functions

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

  Get the value of the topmost text property.

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

  Get multiple values of a text property.

- int **mtext_get_prop_keys** (**MText** ∗mt, int pos, **MSymbol** ∗∗keys)

  Get a list of text property keys at a position of an M-text.

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

  Set a text property.

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

  Set multiple text properties with the same key.

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

  Push a text property.

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

Pop a text property.

- int **mtext_prop_range** (**MText** ∗mt, **MSymbol** key, int pos, int ∗from, int ∗to, int deeper)
  Find the range where the value of a text property is the same.
- **MTextProperty** ∗ **mtext_property** (**MSymbol** key, void ∗val, int control_bits)
  Create a text property.
- **MText** ∗ **mtext_property_mtext** (**MTextProperty** ∗prop)
  Return the M-text of a text property.
- **MSymbol** **mtext_property_key** (**MTextProperty** ∗prop)
  Return the key of a text property.
- void ∗ **mtext_property_value** (**MTextProperty** ∗prop)
  Return the value of a text property.
- int **mtext_property_start** (**MTextProperty** ∗prop)
  Return the start position of a text property.
- int **mtext_property_end** (**MTextProperty** ∗prop)
  Return the end position of a text property.
- **MTextProperty** ∗ **mtext_get_property** (**MText** ∗mt, int pos, **MSymbol** key)
  Get the topmost text property.
- int **mtext_get_properties** (**MText** ∗mt, int pos, **MSymbol** key, **MTextProperty** ∗∗props, int num)
  Get multiple text properties.
- int **mtext_attach_property** (**MText** ∗mt, int from, int to, **MTextProperty** ∗prop)
  Attach a text property to an M-text.
- int **mtext_detach_property** (**MTextProperty** ∗prop)
  Detach a text property from an M-text.
- int **mtext_push_property** (**MText** ∗mt, int from, int to, **MTextProperty** ∗prop)
  Push a text property onto an M-text.
- **MText** ∗ **mtext_serialize** (**MText** ∗mt, int from, int to, **MPlist** ∗property_list)
  Serialize text properties in an M-text.
- **MText** ∗ **mtext_deserialize** (**MText** ∗mt)
  Deserialize text properties in an M-text.

Variables

- **MSymbol** **Mtext_prop_serializer**
  Symbol for specifying serializer functions.
- **MSymbol** **Mtext_prop_deserializer**
  Symbol for specifying deserializer functions.

2.9.1 Detailed Description

Function to handle text properties.

Each character in an M-text can have properties called *text properties*. Text properties store various kinds of information attached to parts of an M-text to provide application programs with a unified view of those information. As rich information can be stored in M-texts in the form of text properties, functions in application programs can be simple.

A text property consists of a *key* and *values*, where key is a symbol and values are anything that can be cast to `(void *)` . Unlike other types of properties, a text property can have multiple values. "The text property whose key is K" may be shortened to "K property".

2.9.2 Typedef Documentation

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

Type of serializer functions.

This is the type of serializer functions. If the key of a symbol property is **Mtext_prop_serializer** (p. 59), the value must be of this type.

See Also:
mtext_serialize() (p. 57), **Mtext_prop_serializer** (p. 59)

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

Type of deserializer functions.

This is the type of deserializer functions. If the key of a symbol property is **Mtext_prop_deserializer** (p. 59), the value must be of this type.

See Also:
mtext_deserialize() (p. 58), **Mtext_prop_deserializer** (p. 59)

2.9.2.3 typedef struct MTextProperty MTextProperty

Type of text properties.

The type **MTextProperty** (p. 51) is for a *text property* objects. Its internal structure is concealed from application programs.

2.9.3 Enumeration Type Documentation

2.9.3.1 enum MTextPropertyControl

Flag bits to control text property.

The **mtext_property()** (p. 56) function accepts logical OR of these flag bits as an argument. They control the behaviour of the created text property as described in the documentation of each flag bit.

Enumerator:

MTEXTPROP_FRONT_STICKY If this flag bit is on, an M-text inserted at the start position or at the middle of the text property inherits the text property.

MTEXTPROP_REAR_STICKY If this flag bit is on, an M-text inserted at the end position or at the middle of the text property inherits the text property.

MTEXTPROP_VOLATILE_WEAK If this flag bit is on, the text property is removed if a text in its region is modified.

MTEXTPROP_VOLATILE_STRONG If this flag bit is on, the text property is removed if a text or the other text property in its region is modified.

MTEXTPROP_NO_MERGE If this flag bit is on, the text property is not automatically merged with the others.

MTEXTPROP_CONTROL_MAX

2.9.4 Function Documentation

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

Get the value of the topmost text property.

The **mtext_get_prop**() (p. 52) function searches the character at **pos** in M-text **mt** for the text property whose key is **key**.

Return value:

If a text property is found, **mtext_get_prop()** (p. 52) returns the value of the property. If the property has multiple values, it returns the topmost one. If no such property is found, it returns `NULL` without changing the external variable **merror_code** (p. 156).

If an error is detected, **mtext_get_prop()** (p. 52) returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

Note:

If `NULL` is returned without an error, there are two possibilities:

- the character at **pos** does not have a property whose key is **key**, or
- the character does have such a property and its value is `NULL`.

If you need to distinguish these two cases, use the **mtext_get_prop_values()** (p. 52) function instead.

Errors:

`MERROR_RANGE`, `MERROR_SYMBOL`

See Also:

mtext_get_prop_values() (p. 52), **mtext_put_prop**() (p. 53), **mtext_put_prop_values**() (p. 54), **mtext_push_prop**() (p. 54), **mtext_pop_prop**() (p. 55), **mtext_prop_range**() (p. 55)

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

Get multiple values of a text property.

The **mtext_get_prop_values**() (p. 52) function searches the character at **pos** in M-text **mt** for the property whose key is **key**. If such a property is found, its values are stored in the memory area pointed to by **values**. **num** limits the maximum number of stored values.

Return value:

If the operation was successful, **mtext_get_prop_values()** (p. 52) returns the number of actually stored values. If the character at **pos** does not have a property whose key is **key**, the return value is 0. If an error is detected, **mtext_get_prop_values()** (p. 52) returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`, `MERROR_SYMBOL`

See Also:

mtext_get_prop() (p. 52), **mtext_put_prop()** (p. 53), **mtext_put_prop_values()** (p. 54), **mtext_push_prop()** (p. 54), **mtext_pop_prop()** (p. 55), **mtext_prop_range()** (p. 55)

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

Get a list of text property keys at a position of an M-text.

The **mtext_get_prop_keys()** (p. 53) function creates an array whose elements are the keys of text properties found at position **pos** in M-text **mt**, and sets ∗**keys** to the address of the created array. The user is responsible to free the memory allocated for the array.

Return value:

If the operation was successful, **mtext_get_prop_keys()** (p. 53) returns the length of the key list. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`

See Also:

mtext_get_prop() (p. 52), **mtext_put_prop()** (p. 53), **mtext_put_prop_values()** (p. 54), **mtext_get_prop_values()** (p. 52), **mtext_push_prop()** (p. 54), **mtext_pop_prop()** (p. 55)

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

Set a text property.

The **mtext_put_prop()** (p. 53) function sets a text property to the characters between **from** (inclusive) and **to** (exclusive) in M-text **mt**. **key** and **val** specify the key and the value of the text property. With this function,

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

becomes

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

Return value:

If the operation was successful, **mtext_put_prop()** (p. 53) returns 0. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`, `MERROR_SYMBOL`

See Also:

mtext_put_prop_values() (p. 54), **mtext_get_prop()** (p. 52), **mtext_get_prop_values()** (p. 52), **mtext_push_prop()** (p. 54), **mtext_pop_prop()** (p. 55), **mtext_prop_range()** (p. 55)

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

Set multiple text properties with the same key.

The **mtext_put_prop_values()** (p. 54) function sets a text property to the characters between **from** (inclusive) and **to** (exclusive) in M-text **mt**. **key** and **values** specify the key and the values of the text property. **num** specifies the number of property values to be set.

Return value:

If the operation was successful, **mtext_put_prop_values()** (p. 54) returns 0. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`, `MERROR_SYMBOL`

See Also:

mtext_put_prop() (p. 53), **mtext_get_prop()** (p. 52), **mtext_get_prop_values()** (p. 52), **mtext_push_prop()** (p. 54), **mtext_pop_prop()** (p. 55), **mtext_prop_range()** (p. 55)

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

Push a text property.

The **mtext_push_prop()** (p. 54) function pushes a text property whose key is **key** and value is **val** to the characters between **from** (inclusive) and **to** (exclusive) in M-text **mt**. With this function,

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

becomes

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

Return value:

If the operation was successful, **mtext_push_prop()** (p. 54) returns 0. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`, `MERROR_SYMBOL`

See Also:

mtext_put_prop() (p. 53), **mtext_put_prop_values()** (p. 54), **mtext_get_prop()** (p. 52), **mtext_get_prop_values()** (p. 52), **mtext_pop_prop()** (p. 55), **mtext_prop_range()** (p. 55)

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

Pop a text property.

The **mtext_pop_prop()** (p. 55) function removes the topmost text property whose key is **key** from the characters between **from** (inclusive) and and **to** (exclusive) in **mt**.

This function does nothing if characters in the region have no such text property. With this function,

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

becomes

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

Return value:
If the operation was successful, **mtext_pop_prop()** (p. 55) return 0. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:
`MERROR_RANGE`, `MERROR_SYMBOL`

See Also:
mtext_put_prop() (p. 53), **mtext_put_prop_values()** (p. 54), **mtext_get_prop()** (p. 52), **mtext_get_prop_values()** (p. 52), **mtext_push_prop()** (p. 54), **mtext_prop_range()** (p. 55)

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

Find the range where the value of a text property is the same.

The **mtext_prop_range()** (p. 55) function investigates the extent where all characters have the same value for a text property. It first finds the value of the property specified by **key** of the character at **pos** in M-text **mt**. Then it checks if adjacent characters have the same value for the property **key**. The beginning and the end of the found range are stored to the variable pointed to by **from** and **to**. The character position stored in **from** is inclusive but that in **to** is exclusive; this fashion is compatible with the range specification in the **mtext_put_prop()** (p. 53) function, etc.

If **deeper** is not 0, not only the topmost but also all the stacked properties whose key is **key** are compared.

If **from** is `NULL`, the beginning of range is not searched for. If **to** is `NULL`, the end of range is not searched for.

Return value:

If the operation was successful, **mtext_prop_range()** (p. 55) returns the number of values the property **key** has at pos. Otherwise it returns -1 and assigns an error code to the external variable `merror_code`.

Errors:
`MERROR_RANGE`, `MERROR_SYMBOL`

See Also:
mtext_put_prop() (p. 53), **mtext_put_prop_values()** (p. 54), **mtext_get_prop()** (p. 52), **mtext_get_prop_values()** (p. 52), **mtext_pop_prop()** (p. 55), **mtext_push_prop()** (p. 54)

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

Create a text property.

The **mtext_property()** (p. 56) function returns a newly allocated text property whose key is **key** and value is **val**. The created text property is not attached to any M-text, i.e. it is detached.

control_bits must be 0 or logical OR of `enum MTextPropertyControl`.

2.9.4.10 MText∗ mtext_property_mtext (MTextProperty ∗ *prop*)

Return the M-text of a text property.

The **mtext_property_mtext()** (p. 56) function returns the M-text to which text property **prop** is attached. If **prop** is currently detached, NULL is returned.

2.9.4.11 MSymbol mtext_property_key (MTextProperty ∗ *prop*)

Return the key of a text property.

The **mtext_property_key()** (p. 56) function returns the key (symbol) of text property **prop**.

2.9.4.12 void∗ mtext_property_value (MTextProperty ∗ *prop*)

Return the value of a text property.

The **mtext_property_value()** (p. 56) function returns the value of text property **prop**.

2.9.4.13 int mtext_property_start (MTextProperty ∗ *prop*)

Return the start position of a text property.

The **mtext_property_start()** (p. 56) function returns the start position of text property **prop**. The start position is a character position of an M-text where **prop** begins. If **prop** is detached, it returns -1.

2.9.4.14 int mtext_property_end (MTextProperty ∗ *prop*)

Return the end position of a text property.

The **mtext_property_end()** (p. 56) function returns the end position of text property **prop**. The end position is a character position of an M-text where **prop** ends. If **prop** is detached, it returns -1.

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

Get the topmost text property.

The **mtext_get_property()** (p. 56) function searches the character at position **pos** in M-text **mt** for a text property whose key is **key**.

Return value:

> If a text property is found, **mtext_get_property()** (p. 56) returns it. If there are multiple text properties, it returns the topmost one. If no such property is found, it returns `NULL` without changing the external variable **merror_code** (p. 156).

If an error is detected, **mtext_get_property()** (p. 56) returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

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

Get multiple text properties.

The **mtext_get_properties()** (p. 57) function searches the character at **pos** in M-text **mt** for properties whose key is **key**. If such properties are found, they are stored in the memory area pointed to by **props**. **num** limits the maximum number of stored properties.

Return value:

If the operation was successful, **mtext_get_properties()** (p. 57) returns the number of actually stored properties. If the character at **pos** does not have a property whose key is **key**, the return value is 0. If an error is detected, **mtext_get_properties()** (p. 57) returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

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

Attach a text property to an M-text.

The **mtext_attach_property()** (p. 57) function attaches text property **prop** to the range between **from** and **to** in M-text **mt**. If **prop** is already attached to an M-text, it is detached before attached to **mt**.

Return value:

If the operation was successful, **mtext_attach_property()** (p. 57) returns 0. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

2.9.4.18 int mtext_detach_property (MTextProperty ∗ *prop*)

Detach a text property from an M-text.

The **mtext_detach_property()** (p. 57) function makes text property **prop** detached.

Return value:

This function always returns 0.

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

Push a text property onto an M-text.

The **mtext_push_property()** (p. 57) function pushes text property **prop** to the characters between **from** (inclusive) and **to** (exclusive) in M-text **mt**.

Return value:

If the operation was successful, **mtext_push_property()** (p. 57) returns 0. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

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

Serialize text properties in an M-text.

The **mtext_serialize()** (p. 57) function serializes the text between **from** and **to** in M-text **mt**. The serialized result is an M-text in a form of XML. **property_list** limits the text properties to be serialized. Only those text properties whose key

- appears as the value of an element in **property_list**, and
- has the symbol property **Mtext_prop_serializer** (p. 59)

are serialized as a "property" element in the resulting XML representation.

The DTD of the generated XML is as follows:

```
<!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>
 ]>
```

This function depends on the libxml2 library. If the m17n library is configured without libxml2, this function always fails.

Return value:

If the operation was successful, **mtext_serialize()** (p. 57) returns an M-text in the form of XML. Otherwise it returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

See Also:

mtext_deserialize() (p. 58), **Mtext_prop_serializer** (p. 59)

2.9.4.21 MText* mtext_deserialize (MText * *mt*)

Deserialize text properties in an M-text.

The **mtext_deserialize()** (p. 58) function deserializes M-text **mt**. **mt** must be an XML having the following DTD.

```
<!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>
 ]>
```

This function depends on the libxml2 library. If the m17n library is configured without libxml2, this function always fail.

Return value:

If the operation was successful, **mtext_deserialize()** (p. 58) returns the resulting M-text. Otherwise it returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

See Also:

mtext_serialize() (p. 57), **Mtext_prop_deserializer** (p. 59)

2.9.5 Variable Documentation

2.9.5.1 MSymbol Mtext_prop_serializer

Symbol for specifying serializer functions.

To serialize a text property, the user must supply a serializer function for that text property. This is done by giving a symbol property whose key is **Mtext_prop_serializer** (p. 59) and value is a pointer to an appropriate serializer function.

See Also:

mtext_serialize() (p. 57), **MTextPropSerializeFunc** (p. 51)

2.9.5.2 MSymbol Mtext_prop_deserializer

Symbol for specifying deserializer functions.

To deserialize a text property, the user must supply a deserializer function for that text property. This is done by giving a symbol property whose key is **Mtext_prop_deserializer** (p. 59) and value is a pointer to an appropriate deserializer function.

See Also:

mtext_deserialize() (p. 58), **MTextPropSerializeFunc** (p. 51)

2.10 Database

The m17n database and API for it.

Typedefs

- typedef struct **MDatabase** **MDatabase**

  Type of database.

Functions

- **MDatabase** ∗ **mdatabase_find** (**MSymbol** tag0, **MSymbol** tag1, **MSymbol** tag2, **MSymbol** tag3)

  Look for a data in the database.

- **MPlist** ∗ **mdatabase_list** (**MSymbol** tag0, **MSymbol** tag1, **MSymbol** tag2, **MSymbol** tag3)

  Return a data list of the m17n database.

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

  Define a data of the m17n database.

- void ∗ **mdatabase_load** (**MDatabase** ∗mdb)

  Load a data from the database.

- **MSymbol** ∗ **mdatabase_tag** (**MDatabase** ∗mdb)

  Get tags of a data.

Variables

- char ∗ **mdatabase_dir**

  Directory for application specific data.

2.10.1 Detailed Description

The m17n database and API for it.

The m17n library acquires various kinds of information from data in the *m17n database* on demand. Application programs can also add/load their original data to/from the m17n database by setting the variable **mdatabase_dir** (p. 62) to an application-specific directory and storing data in it. Users can overwrite those data by storing preferable data in the directory specified by the environment variable "M17NDIR", or if it is not set, in the directory "∼/.m17n.d".

The m17n database contains multiple heterogeneous data, and each data is identified by four tags; TAG0, TAG1, TAG2, TAG3. Each tag must be a symbol.

TAG0 specifies the type of data stored in the database as below.

- If TAG0 is **Mchar_table** (p. 32), the data is of the *chartable type* and provides information about each character. In this case, TAG1 specifies the type of the information and must be **Msymbol** (p. 17), **Minteger** (p. 23), **Mstring** (p. 17), **Mtext** (p. 23), or **Mplist** (p. 23). TAG2 and TAG3 can be any symbols.

- If TAG0 is **Mcharset** (p. 71), the data is of the *charset type* and provides a decode/encode mapping table for a charset. In this case, TAG1 must be a symbol representing a charset. TAG2 and TAG3 can be any symbols.

- If TAG0 is neither **Mchar_table** (p. 32) nor **Mcharset** (p. 71), the data is of the *plist type*. See the documentation of the **mdatabase_load()** (p. 62) function for the details. In this case, TAG1, TAG2, and TAG3 can be any symbols.

The notation <TAG0, TAG1, TAG2, TAG3> means a data with those tags.

Application programs first calls the **mdatabase_find()** (p. 61) function to get a pointer to an object of the type **MDatabase** (p. 61). That object holds information about the specified data. When it is successfully returned, the **mdatabase_load()** (p. 62) function loads the data. The implementation of the structure **MDatabase** (p. 61) is concealed from application programs.

2.10.2 Typedef Documentation

2.10.2.1 typedef struct MDatabase MDatabase

Type of database.

The type **MDatabase** (p. 61) is for a database object. Its internal structure is concealed from an application program.

2.10.3 Function Documentation

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

Look for a data in the database.

The **mdatabase_find()** (p. 61) function searches the m17n database for a data who has tags **tag0** through **tag3**, and returns a pointer to the data. If such a data is not found, it returns `NULL`.

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

Return a data list of the m17n database.

The **mdatabase_list()** (p. 61) function searches the m17n database for data who have tags **tag0** through **tag3**, and returns their list by a plist. The value **Mnil** (p. 17) in **tagn** means a wild card that matches any tag. Each element of the plist has key **Mt** (p. 17) and value a pointer to type **MDatabase** (p. 61).

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

Define a data of the m17n database.

The **mdatabase_define()** (p. 61) function defines a data that has tags **tag0** through **tag3** and additional information **extra_info**.

loader is a pointer to a function that loads the data from the database. This function is called from the **mdatabase_load()** (p. 62) function with the two arguments **tags** and **extra_info**. Here, **tags** is the array of **tag0** through **tag3**.

If **loader** is `NULL`, the default loader of the m17n library is used. In this case, **extra_info** must be a string specifying a filename that contains the data.

Return value:

If the operation was successful, **mdatabase_define()** (p. 61) returns a pointer to the defined data, which can be used as an argument to **mdatabase_load()** (p. 62). Otherwise, it returns `NULL`.

See Also:

mdatabase_load() (p. 62), **mdatabase_define()** (p. 61)

2.10.3.4 void∗ mdatabase_load (MDatabase ∗ *mdb*)

Load a data from the database.

The **mdatabase_load()** (p. 62) function loads a data specified in **mdb** and returns the contents. The type of contents depends on the type of the data.

If the data is of the *plist type*, this function returns a pointer to *plist*.

If the database is of the *chartable type*, it returns a chartable. The default value of the chartable is set according to the second tag of the data as below:

- If the tag is **Msymbol** (p. 17), the default value is **Mnil** (p. 17).
- If the tag is **Minteger** (p. 23), the default value is -1.
- Otherwise, the default value is `NULL`.

If the data is of the *charset type*, it returns a plist of length 2 (keys are both **Mt** (p. 17)). The value of the first element is an array of integers that maps code points to the corresponding character codes. The value of the second element is a chartable of integers that does the reverse mapping. The charset must be defined in advance.

See Also:

mdatabase_load() (p. 62), **mdatabase_define()** (p. 61)

2.10.3.5 MSymbol∗ mdatabase_tag (MDatabase ∗ *mdb*)

Get tags of a data.

The **mdatabase_tag()** (p. 62) function returns an array of tags (symbols) that identify the data in **mdb**. The length of the array is four.

2.10.4 Variable Documentation

2.10.4.1 char∗ mdatabase_dir

Directory for application specific data.

If an application program wants to provide a data specific to the program or a data overriding what supplied by the m17n database, it must set this variable to a name of directory that contains the data files before it calls the macro **M17N_INIT()** (p. 7). The directory may contain a file "mdb.dir" which contains a list of data definitions in the format described in **mdbDir(5)** (p. 211).

The default value is NULL.

2.11 SHELL API

API provided by libm17n.so.

Modules

- **Charset**
  Charset objects and API for them.
- **Code Conversion**
  Coding system objects and API for them.
- **Locale**
  Locale objects and API for them.
- **Input Method (basic)**
  API for Input method.

2.11.1 Detailed Description

API provided by libm17n.so.

2.12 Charset

Charset objects and API for them.

Variables: Symbols representing a charset.

Each of the following symbols represents a predefined charset.

- **MSymbol Mcharset_ascii**
  Symbol representing the charset ASCII.
- **MSymbol Mcharset_iso_8859_1**
  Symbol representing the charset ISO/IEC 8859/1.
- **MSymbol Mcharset_unicode**
  Symbol representing the charset Unicode.
- **MSymbol Mcharset_m17n**
  Symbol representing the largest charset.
- **MSymbol Mcharset_binary**
  Symbol representing the charset for ill-decoded characters.

Variables: Parameter keys for mchar_define_charset().

These are the predefined symbols to use as parameter keys for the function **mchar_define_charset**() (p. 66) (which see).

- **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**

Variables: Symbols representing charset methods.

These are the predefined symbols that can be a value of the **Mmethod** parameter of a charset used in an argument to the **mchar_define_charset()** (p. 66) function.

A method specifies how code-points and character codes are converted. See the documentation of the **mchar_define_charset()** (p. 66) function for the details.

- **MSymbol Moffset**

  Symbol for the offset type method of charset.

- **MSymbol Mmap**

  Symbol for the map type method of charset.

- **MSymbol Munify**

  Symbol for the unify type method of charset.

- **MSymbol Msubset**

  Symbol for the subset type method of charset.

- **MSymbol Msuperset**

  Symbol for the superset type method of charset.

Defines

- #define **MCHAR_INVALID_CODE**

  Invalid code-point.

Functions

- **MSymbol mchar_define_charset** (const char ∗name, **MPlist** ∗plist)

  Define a charset.

- **MSymbol mchar_resolve_charset** (**MSymbol** symbol)

  Resolve charset name.

- int **mchar_list_charset** (**MSymbol** ∗∗symbols)

  List symbols representing charsets.

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

  Decode a code-point.

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

  Encode a character code.

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

  Call a function for all the characters in a specified charset.

Variables

- **MSymbol Mcharset**

  The symbol `Mcharset`.

2.12.1 Detailed Description

Charset objects and API for them.

The m17n library uses *charset* objects to represent a coded character sets (CCS). The m17n library supports many predefined coded character sets. Moreover, application programs can add other charsets. A character can belong to multiple charsets.

The m17n library distinguishes the following three concepts:

- A *code-point* is a number assigned by the CCS to each character. Code-points may or may not be continuous. The type `unsigned` is used to represent a code-point. An invalid code-point is represented by the macro `MCHAR_INVALID_CODE`.

- A *character index* is the canonical index of a character in a CCS. The character that has the character index N occupies the Nth position when all the characters in the current CCS are sorted by their code-points. Character indices in a CCS are continuous and start with 0.

- A *character code* is the internal representation in the m17n library of a character. A character code is a signed integer of 21 bits or longer.

Each charset object defines how characters are converted between code-points and character codes. To *encode* means converting code-points to character codes and to *decode* means converting character codes to code-points.

2.12.2 Define Documentation

2.12.2.1 #define MCHAR_INVALID_CODE

Invalid code-point.

The macro **MCHAR_INVALID_CODE** (p. 66) gives the invalid code-point.

2.12.3 Function Documentation

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

Define a charset.

The **mchar_define_charset()** (p. 66) function defines a new charset and makes it accessible via a symbol whose name is **name**. **plist** specifies parameters of the charset as below:

- Key is **Mmethod**, value is a symbol.

  The value specifies the method for decoding/encoding code-points in the charset. It must be **Moffset** (p. 70), **Mmap** (p. 70) (default), **Munify** (p. 70), **Msubset** (p. 71), or **Msuperset** (p. 71).

- Key is **Mdimension**, value is an integer

  The value specifies the dimension of code-points of the charset. It must be 1 (default), 2, 3, or 4.

- Key is **Mmin_range**, value is an unsigned integer

  The value specifies the minimum range of a code-point, which means that the Nth byte of the value is the minimum Nth byte of code-points of the charset. The default value is 0.

- Key is **Mmax_range**, value is an unsigned integer

  The value specifies the maximum range of a code-point, which means that the Nth byte of the value is the maximum Nth byte of code-points of the charset. The default value is 0xFF, 0xFFFF, 0xFFFFFF, or 0xFFFFFFFF if the dimension is 1, 2, 3, or 4 respectively.

- Key is **Mmin_code**, value is an unsigned integer

  The value specifies the minimum code-point of the charset. The default value is the minimum range.

- Key is **Mmax_code**, value is an unsigned integer

  The value specifies the maximum code-point of the charset. The default value is the maximum range.

- Key is **Mascii_compatible**, value is a symbol

  The value specifies whether the charset is ASCII compatible or not. If the value is **Mnil** (p. 17) (default), it is not ASCII compatible, else compatible.

- Key is **Mfinal_byte**, value is an integer

  The value specifies the *final byte* of the charset registered in The International Registry. It must be 0 (default) or 32..127. The value 0 means that the charset is not in the registry.

- Key is **Mrevision**, value is an integer

  The value specifies the *revision number* of the charset registered in The International Registry. It must be 0..127. If the charset is not in The International Registry, the value is ignored. The value 0 means that the charset has no revision number.

- Key is **Mmin_char**, value is an integer

  The value specifies the minimum character code of the charset. The default value is 0.

- Key is **Mmapfile**, value is an M-text

  If the method is **Mmap** (p. 70) or **Munify** (p. 70), a data that contains mapping information is added to the m17n database by calling the function **mdatabase_define()** (p. 61) with the value as an argument **extra_info**, i.e. the value is used as a file name of the data.

  Otherwise, this parameter is ignored.

- Key is **Mparents**, value is a plist

  If the method is **Msubset** (p. 71), the value must is a plist of length 1, and the value of the plist must be a symbol representing a parent charset.

  If the method is **Msuperset** (p. 71), the value must be a plist of length less than 9, and the values of the plist must be symbols representing subset charsets.

  Otherwise, this parameter is ignored.

- Key is **Mdefine_coding**, value is a symbol

  If the dimension of the charset is 1, the value specifies whether or not to define a coding system of the same name whose type is **Mcharset** (p. 71). A coding system is defined if the value is not **Mnil** (p. 17).

  Otherwise, this parameter is ignored.

Return value:

If the operation was successful, **mchar_define_charset()** (p. 66) returns a symbol whose name is **name**. Otherwise it returns **Mnil** (p. 17) and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_CHARSET`

2.12.3.2 MSymbol mchar_resolve_charset (MSymbol *symbol*)

Resolve charset name.

The **mchar_resolve_charset()** (p. 67) function returns **symbol** if it represents a charset. Otherwise, canonicalize **symbol** as to a charset name, and if the canonicalized name represents a charset, return it. Otherwise, return **Mnil** (p. 17).

2.12.3.3 int mchar_list_charset (MSymbol ∗∗ *symbols*)

List symbols representing charsets.

The mchar_list_charsets() function makes an array of symbols representing a charset, stores the pointer to the array in a place pointed to by **symbols**, and returns the length of the array.

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

Decode a code-point.

The **mchar_decode()** (p. 68) function decodes code-point **code** in the charset represented by the symbol **charset_name** to get a character code.

Return value:

If decoding was successful, **mchar_decode()** (p. 68) returns the decoded character code. Otherwise it returns -1.

See Also:

mchar_encode() (p. 68)

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

Encode a character code.

The **mchar_encode()** (p. 68) function encodes character code **c** to get a code-point in the charset represented by the symbol **charset_name**.

Return value:

If encoding was successful, **mchar_encode()** (p. 68) returns the encoded code-point. Otherwise it returns **MCHAR_INVALID_CODE** (p. 66).

See Also:

mchar_decode() (p. 68)

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

Call a function for all the characters in a specified charset.

The mcharset_map_chars() function calls **func** for all the characters in the charset named **charset_name**. A call is done for a chunk of consecutive characters rather than character by character.

func receives three arguments: **from**, **to**, and **arg**. **from** and **to** specify the range of character codes in **charset**. **arg** is the same as **func_arg**.

Return value:

If the operation was successful, mcharset_map_chars() returns 0. Otherwise, it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_CHARSET`

2.12.4 Variable Documentation

2.12.4.1 MSymbol Mcharset_ascii

Symbol representing the charset ASCII.

The symbol **Mcharset_ascii** (p. 69) has name `"ascii"` and represents the charset ISO 646, USA Version X3.4-1968 (ISO-IR-6).

2.12.4.2 MSymbol Mcharset_iso_8859_1

Symbol representing the charset ISO/IEC 8859/1.

The symbol **Mcharset_iso_8859_1** (p. 69) has name `"iso-8859-1"` and represents the charset ISO/IEC 8859-1:1998.

2.12.4.3 MSymbol Mcharset_unicode

Symbol representing the charset Unicode.

The symbol **Mcharset_unicode** (p. 69) has name `"unicode"` and represents the charset Unicode.

2.12.4.4 MSymbol Mcharset_m17n

Symbol representing the largest charset.

The symbol **Mcharset_m17n** (p. 69) has name `"m17n"` and represents the charset that contains all characters supported by the m17n library.

2.12.4.5 MSymbol Mcharset_binary

Symbol representing the charset for ill-decoded characters.

The symbol **Mcharset_binary** (p. 69) has name `"binary"` and represents the fake charset which the decoding functions put to an M-text as a text property when they encounter an invalid byte (sequence).

See **Code Conversion** (p. 72) for more details.

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

Symbol for the offset type method of charset.

The symbol **Moffset** (p. 70) has the name `"offset"` and, when used as a value of **Mmethod** parameter of a charset, it means that the conversion of code-points and character codes of the charset is done by this calculation:

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

where, MIN-CODE is a value of **Mmin_code** parameter of the charset, and MIN-CHAR is a value of **Mmin_char** parameter.

2.12.4.22 MSymbol Mmap

Symbol for the map type method of charset.

The symbol **Mmap** (p. 70) has the name `"map"` and, when used as a value of **Mmethod** parameter of a charset, it means that the conversion of code-points and character codes of the charset is done by map looking up. The map must be given by **Mmapfile** parameter.

2.12.4.23 MSymbol Munify

Symbol for the unify type method of charset.

The symbol **Munify** (p. 70) has the name `"unify"` and, when used as a value of **Mmethod** parameter of a charset, it means that the conversion of code-points and character codes of the charset is done by map looking up and offsetting. The map must be given by **Mmapfile** parameter. For this kind of charset, a unique continuous character code space for all characters is assigned.

If the map has an entry for a code-point, the conversion is done by looking up the map. Otherwise, the conversion is done by this calculation:

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

where, MIN-CODE is a value of **Mmin_code** parameter of the charset, and LOWEST-CHAR-CODE is the lowest character code of the assigned code space.

2.12.4.24 MSymbol Msubset

Symbol for the subset type method of charset.

The symbol **Msubset** (p. 71) has the name `"subset"` and, when used as a value of **Mmethod** parameter of a charset, it means that the charset is a subset of a parent charset. The parent charset must be given by **Mparents** parameter. The conversion of code-points and character codes of the charset is done conceptually by this calculation:

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

where, PARENT-CODE is a pseudo function that returns a character code of CODE-POINT in the parent charset, and SUBSET-OFFSET is a value given by **Msubset_offset** parameter.

2.12.4.25 MSymbol Msuperset

Symbol for the superset type method of charset.

The symbol **Msuperset** (p. 71) has the name `"superset"` and, when used as a value of **Mmethod** parameter of a charset, it means that the charset is a superset of parent charsets. The parent charsets must be given by **Mparents** parameter.

2.12.4.26 MSymbol Mcharset

The symbol `Mcharset`.

Any decoded M-text has a text property whose key is the predefined symbol `Mcharset`. The name of `Mcharset` is `"charset"`.

2.13 Code Conversion

Coding system objects and API for them.

Data Structures

- struct **MConverter**

  Structure to be used in code conversion.

- struct **MCodingInfoISO2022**

  *Structure for a coding system of type **MCODING_TYPE_ISO_2022** (*p. *76).*

- struct **MCodingInfoUTF**

  *Structure for extra information about a coding system of type **MCODING_TYPE_UTF** (*p. *76).*

Variables: Symbols representing coding systems

- **MSymbol Mcoding_us_ascii**

  Symbol for the coding system US-ASCII.

- **MSymbol Mcoding_iso_8859_1**

  Symbol for the coding system ISO-8859-1.

- **MSymbol Mcoding_utf_8**

  Symbol for the coding system UTF-8.

- **MSymbol Mcoding_utf_8_full**

  Symbol for the coding system UTF-8-FULL.

- **MSymbol Mcoding_utf_16**

  Symbol for the coding system UTF-16.

- **MSymbol Mcoding_utf_16be**

  Symbol for the coding system UTF-16BE.

- **MSymbol Mcoding_utf_16le**

  Symbol for the coding system UTF-16LE.

- **MSymbol Mcoding_utf_32**

  Symbol for the coding system UTF-32.

- **MSymbol Mcoding_utf_32be**

  Symbol for the coding system UTF-32BE.

- **MSymbol Mcoding_utf_32le**

  Symbol for the coding system UTF-32LE.

- **MSymbol Mcoding_sjis**

  Symbol for the coding system SJIS.

Variables: Parameter keys for mconv_define_coding().

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

Variables: Symbols representing coding system types.

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

Variables: Symbols appearing in the value of Mflags parameter.

Symbols that can be a value of the **Mflags** parameter of a coding system used in an argument to the **mconv_define_coding()** (p. 77) function (which see).

- **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**

Variables: Others

Remaining variables.

- **MSymbol Mmaybe**

  Symbol whose name is "maybe".

- **MSymbol Mcoding**

  The symbol `Mcoding`.

Enumerations

- 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 }

  Codes that represent the result of code conversion.

- enum **MCodingType** {
  MCODING_TYPE_CHARSET,
  MCODING_TYPE_UTF,
  MCODING_TYPE_ISO_2022,
  MCODING_TYPE_MISC }

  Types of coding system.

- 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 }

  Bit-masks to specify the detail of coding system whose type is MCODING_TYPE_ISO_2022.

Functions

- **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)

  Define a coding system.

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

Resolve coding system name.

- int **mconv_list_codings** (**MSymbol** ∗∗symbols)

  List symbols representing coding systems.

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

  Create a code converter bound to a buffer.

- **MConverter** ∗ **mconv_stream_converter** (**MSymbol** name, FILE ∗fp)

  Create a code converter bound to a stream.

- int **mconv_reset_converter** (**MConverter** ∗converter)

  Reset a code converter.

- void **mconv_free_converter** (**MConverter** ∗converter)

  Free a code converter.

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

  Bind a buffer to a code converter.

- **MConverter** ∗ **mconv_rebind_stream** (**MConverter** ∗converter, FILE ∗fp)

  Bind a stream to a code converter.

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

  Decode a byte sequence into an M-text.

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

  Decode a buffer area based on a coding system.

- **MText** ∗ **mconv_decode_stream** (**MSymbol** name, FILE ∗fp)

  Decode a stream input based on a coding system.

- int **mconv_encode** (**MConverter** ∗converter, **MText** ∗mt)

  Encode an M-text into a byte sequence.

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

  Encode a part of an M-text.

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

  Encode an M-text into a buffer area.

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

  Encode an M-text to write to a stream.

- int **mconv_getc** (**MConverter** ∗converter)

  Read a character via a code converter.

- int **mconv_ungetc** (**MConverter** ∗converter, int c)

  Push a character back to a code converter.

- int **mconv_putc** (**MConverter** ∗converter, int c)

  Write a character via a code converter.

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

  Read a line using a code converter.

2.13.1 Detailed Description

Coding system objects and API for them.

The m17n library represents a character encoding scheme (CES) of coded character sets (CCS) as an object called *coding system*. Application programs can add original coding systems.

To *encode* means converting code-points to character codes and to *decode* means converting character codes back to code-points.

Application programs can decode a byte sequence with a specified coding system into an M-text, and inversely, can encode an M-text into a byte sequence.

2.13.2 Enumeration Type Documentation

2.13.2.1 enum MConversionResult

Codes that represent the result of code conversion.

One of these values is set in `MConverter->result`.

Enumerator:

MCONVERSION_RESULT_SUCCESS Code conversion is successful.

MCONVERSION_RESULT_INVALID_BYTE On decoding, the source contains an invalid byte.

MCONVERSION_RESULT_INVALID_CHAR On encoding, the source contains a character that cannot be encoded by the specified coding system.

MCONVERSION_RESULT_INSUFFICIENT_SRC On decoding, the source ends with an incomplete byte sequence.

MCONVERSION_RESULT_INSUFFICIENT_DST On encoding, the destination is too short to store the result.

MCONVERSION_RESULT_IO_ERROR An I/O error occurred in the conversion.

2.13.2.2 enum MCodingType

Types of coding system.

Enumerator:

MCODING_TYPE_CHARSET A coding system of this type supports charsets directly. The dimension of each charset defines the length of bytes to represent a single character of the charset, and a byte sequence directly represents the code-point of a character. The m17n library provides the default decoding and encoding routines of this type.

MCODING_TYPE_UTF A coding system of this type supports byte sequences of a UTF (UTF-8, UTF-16, UTF-32) like structure. The m17n library provides the default decoding and encoding routines of this type.

MCODING_TYPE_ISO_2022 A coding system of this type supports byte sequences of an ISO-2022 like structure. The details of each structure are specified by **`MCodingInfoISO2022`** `(p. 162)` . The m17n library provides decoding and encoding routines of this type.

MCODING_TYPE_MISC A coding system of this type is for byte sequences of miscellaneous structures. The m17n library does not provide decoding and encoding routines of this type. They must be provided by the application program.

2.13.2.3 enum MCodingFlagISO2022

Bit-masks to specify the detail of coding system whose type is MCODING_TYPE_ISO_2022.

Enumerator:

MCODING_ISO_RESET_AT_EOL On encoding, reset the invocation and designation status to initial at end of line.

MCODING_ISO_RESET_AT_CNTL On encoding, reset the invocation and designation status to initial before any control codes.

MCODING_ISO_EIGHT_BIT Use the right graphic plane.

MCODING_ISO_LONG_FORM Use the non-standard 4 bytes format for designation sequence for charsets JISX0208-1978, GB2312, and JISX0208-1983.

MCODING_ISO_DESIGNATION_G0 On encoding, unless explicitly specified, designate charsets to G0.

MCODING_ISO_DESIGNATION_G1 On encoding, unless explicitly specified, designate charsets except for ASCII to G1.

MCODING_ISO_DESIGNATION_CTEXT On encoding, unless explicitly specified, designate 94-chars charsets to G0, 96-chars charsets to G1.

MCODING_ISO_DESIGNATION_CTEXT_EXT On encoding, encode such charsets not conforming to ISO-2022 by ESC % / ..., and encode non-supported Unicode characters by ESC % G ... ESC % @ . On decoding, handle those escape sequences.

MCODING_ISO_LOCKING_SHIFT Use locking shift.

MCODING_ISO_SINGLE_SHIFT Use single shift (SS2 (0x8E or ESC N), SS3 (0x8F or ESC O)).

MCODING_ISO_SINGLE_SHIFT_7 Use 7-bit single shift 2 (SS2 (0x19)).

MCODING_ISO_EUC_TW_SHIFT Use EUC-TW like special shifting.

MCODING_ISO_ISO6429 Use ISO-6429 escape sequences to indicate direction. Not yet implemented.

MCODING_ISO_REVISION_NUMBER On encoding, if a charset has revision number, produce escape sequences to specify the number.

MCODING_ISO_FULL_SUPPORT Support all ISO-2022 charsets.

MCODING_ISO_FLAG_MAX

2.13.3 Function Documentation

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*)

Define a coding system.

The **mconv_define_coding()** (p. 77) function defines a new coding system and makes it accessible via a symbol whose name is **name**. **plist** specifies parameters of the coding system as below:

- Key is `Mtype`, value is a symbol

  The value specifies the type of the coding system. It must be **Mcharset**, **Mutf**, **Miso_2022**, or **Mnil**.

  If the type is **Mcharset**, **extra_info** is ignored.

If the type is **Mutf**, **extra_info** must be a pointer to **MCodingInfoUTF** (p. 163).

If the type is **Miso_2022**, **extra_info** must be a pointer to **MCodingInfoISO2022** (p. 162).

If the type is **Mnil** (p. 17), the argument **resetter**, **decoder**, and **encoder** must be supplied. **extra_info** is ignored. Otherwise, they can be `NULL` and the m17n library provides proper defaults.

- Key is **Mcharsets**, value is a plist

  The value specifies a list charsets supported by the coding system. The keys of the plist must be **Msymbol** (p. 17), and the values must be symbols representing charsets.

- Key is **Mflags**, value is a plist

  If the type is **Miso_2022**, the values specifies flags to control the ISO 2022 interpreter. The keys of the plist must e **Msymbol** (p. 17), and values must be one of the following.

  - **Mreset_at_eol**
    If this flag exists, designation and invocation status is reset to the initial state at the end of line.
  - **Mreset_at_cntl**
    If this flag exists, designation and invocation status is reset to the initial state at a control character.
  - **Meight_bit**
    If this flag exists, the graphic plane right is used.
  - **Mlong_form**
    If this flag exists, the over-long escape sequences (ESC '$' '(' <final_byte>) are used for designating the CCS JISX0208.1978, GB2312, and JISX0208.
  - **Mdesignation_g0**
    If this flag and **Mfull_support** exists, designates charsets not listed in the charset list to the graphic register G0.
  - **Mdesignation_g1**
    If this flag and **Mfull_support** exists, designates charsets not listed in the charset list to the graphic register G1.
  - **Mdesignation_ctext**
    If this flag and **Mfull_support** exists, designates charsets not listed in the charset list to a graphic register G0 or G1 based on the criteria of the Compound Text.
  - **Mdesignation_ctext_ext**
    If this flag and **Mfull_support** exists, designates charsets not listed in the charset list to a graphic register G0 or G1, or use extended segment for such charsets based on the criteria of the Compound Text.
  - **Mlocking_shift**
    If this flag exists, use locking shift.
  - **Msingle_shift**
    If this flag exists, use single shift.
  - **Msingle_shift_7**
    If this flag exists, use 7-bit single shift code (0x19).
  - **Meuc_tw_shift**
    If this flag exists, use a special shifting according to EUC-TW.
  - **Miso_6429**
    This flag is currently ignored.
  - **Mrevision_number**
    If this flag exists, use a revision number escape sequence to designate a charset that has a revision number.

  - **Mfull_support**

    If this flag exists, support all charsets registered in the International Registry.

- Key is **Mdesignation**, value is a plist

  If the type is **Miso_2022**, the value specifies how to designate each supported characters. The keys of the plist must be **Minteger** (p. 23), and the values must be numbers indicating a graphic registers. The Nth element value is for the Nth charset of the charset list. The value 0..3 means that it is assumed that a charset is already designated to the graphic register 0..3. The negative value G (-4..-1) means that a charset is not designated to any register at first, and if necessary, is designated to the (G+4) graphic register.

- Key is **Minvocation**, value is a plist

  If the type is **Miso_2022**, the value specifies how to invocate each graphic registers. The plist length must be one or two. The keys of the plist must be **Minteger** (p. 23), and the values must be numbers indicating a graphic register. The value of the first element specifies which graphic register is invocated to the graphic plane left. If the length is one, no graphic register is invocated to the graphic plane right. Otherwise, the value of the second element specifies which graphic register is invocated to the graphic plane right.

- Key is **Mcode_unit**, value is an integer

  If the type is **Mutf**, the value specifies the bit length of a code-unit. It must be 8, 16, or 32.

- Key is **Mbom**, value is a symbol

  If the type is **Mutf** and the code-unit bit length is 16 or 32, it specifies whether or not to use BOM (Byte Order Mark). If the value is **Mnil** (p. 17) (default), BOM is not used, else if the value is **Mmaybe** (p. 88), the existence of BOM is detected at decoding time, else BOM is used.

- Key is **Mlittle_endian**, value is a symbol

  If the type is **Mutf** and the code-unit bit length is 16 or 32, it specifies whether or not the encoding is little endian. If the value is **Mnil** (p. 17) (default), it is big endian, else it is little endian.

resetter is a pointer to a function that resets a converter for the coding system to the initial status. The pointed function is called with one argument, a pointer to a converter object.

decoder is a pointer to a function that decodes a byte sequence according to the coding system. The pointed function is called with four arguments:

- A pointer to the byte sequence to decode.
- The number of bytes to decode.
- A pointer to an M-text to which the decoded characters are appended.
- A pointer to a converter object.

decoder must return 0 if it succeeds. Otherwise it must return -1.

encoder is a pointer to a function that encodes an M-text according to the coding system. The pointed function is called with six arguments:

- A pointer to the M-text to encode.
- The starting position of the encoding.
- The ending position of the encoding.
- A pointer to a memory area where the produced bytes are stored.
- The size of the memory area.

- A pointer to a converter object.

encoder must return 0 if it succeeds. Otherwise it must return -1.

extra_info is a pointer to a data structure that contains extra information about the coding system. The type of the data structure depends on **type**.

Return value:

If the operation was successful, **mconv_define_coding()** (p. 77) returns a symbol whose name is **name**. If an error is detected, it returns **Mnil** (p. 17) and assigns an error code to the external variable **merror_code** (p. 156).

Errors:
`MERROR_CODING`

2.13.3.2 MSymbol mconv_resolve_coding (MSymbol *symbol*)

Resolve coding system name.

The **mconv_resolve_coding()** (p. 80) function returns **symbol** if it represents a coding system. Otherwise, canonicalize **symbol** as to a coding system name, and if the canonicalized name represents a coding system, return it. Otherwise, return **Mnil** (p. 17).

2.13.3.3 int mconv_list_codings (MSymbol ** *symbols*)

List symbols representing coding systems.

The **mconv_list_codings()** (p. 80) function makes an array of symbols representing a coding system, stores the pointer to the array in a place pointed to by **symbols**, and returns the length of the array.

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

Create a code converter bound to a buffer.

The **mconv_buffer_converter()** (p. 80) function creates a pointer to a code converter for coding system **name**. The code converter is bound to buffer area of **n** bytes pointed to by **buf**. Subsequent decodings and encodings are done to/from this buffer area.

name can be **Mnil** (p. 17). In this case, a coding system associated with the current locale (LC_CTYPE) is used.

Return value:
If the operation was successful, **mconv_buffer_converter()** (p. 80) returns the created code converter. Otherwise it returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

Errors:
`MERROR_SYMBOL`, `MERROR_CODING`

See Also:
mconv_stream_converter() (p. 81)

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

Create a code converter bound to a stream.

The **mconv_stream_converter()** (p. 81) function creates a pointer to a code converter for coding system **name**. The code converter is bound to stream **fp**. Subsequent decodings and encodings are done to/from this stream.

name can be **Mnil** (p. 17). In this case, a coding system associated with the current locale (LC_CTYPE) is used.

Return value:

If the operation was successful, **mconv_stream_converter()** (p. 81) returns the created code converter. Otherwise it returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_SYMBOL`, `MERROR_CODING`

See Also:

mconv_buffer_converter() (p. 80)

2.13.3.6 int mconv_reset_converter (MConverter ∗ *converter*)

Reset a code converter.

The **mconv_reset_converter()** (p. 81) function resets code converter **converter** to the initial state.

Return value:

If **converter->coding** has its own reseter function, **mconv_reset_converter()** (p. 81) returns the result of that function applied to **converter**. Otherwise it returns 0.

2.13.3.7 void mconv_free_converter (MConverter ∗ *converter*)

Free a code converter.

The **mconv_free_converter()** (p. 81) function frees the code converter **converter**.

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

Bind a buffer to a code converter.

The **mconv_rebind_buffer()** (p. 81) function binds buffer area of **n** bytes pointed to by **buf** to code converter **converter**. Subsequent decodings and encodings are done to/from this newly bound buffer area.

Return value:

This function always returns **converter**.

See Also:

mconv_rebind_stream() (p. 81)

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

Bind a stream to a code converter.

The **mconv_rebind_stream()** (p. 81) function binds stream **fp** to code converter **converter**. Following decodings and encodings are done to/from this newly bound stream.

Return value:

This function always returns **converter**.

See Also:

mconv_rebind_buffer() (p. 81)

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

Decode a byte sequence into an M-text.

The **mconv_decode()** (p. 82) function decodes a byte sequence and appends the result at the end of M-text **mt**. The source byte sequence is taken from either the buffer area or the stream that is currently bound to **converter**.

Return value:

If the operation was successful, **mconv_decode()** (p. 82) returns updated **mt**. Otherwise it returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_IO`, `MERROR_CODING`

See Also:

mconv_rebind_buffer() (p. 81), **mconv_rebind_stream()** (p. 81), **mconv_encode()** (p. 83), **mconv_encode_range()** (p. 83), **mconv_decode_buffer()** (p. 82), **mconv_decode_stream()** (p. 82)

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

Decode a buffer area based on a coding system.

The **mconv_decode_buffer()** (p. 82) function decodes **n** bytes of the buffer area pointed to by **buf** based on the coding system **name**. A temporary code converter for decoding is automatically created and freed.

Return value:

If the operation was successful, **mconv_decode_buffer()** (p. 82) returns the resulting M-text. Otherwise it returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_IO`, `MERROR_CODING`

See Also:

mconv_decode() (p. 82), **mconv_decode_stream()** (p. 82)

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

Decode a stream input based on a coding system.

The **mconv_decode_stream()** (p. 82) function decodes the entire byte sequence read in from stream **fp** based on the coding system **name**. A code converter for decoding is automatically created and freed.

Return value:

If the operation was successful, **mconv_decode_stream()** (p. 82) returns the resulting M-text. Otherwise it returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_IO`, `MERROR_CODING`

See Also:

mconv_decode() (p. 82), **mconv_decode_buffer()** (p. 82)

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

Encode an M-text into a byte sequence.

The **mconv_encode()** (p. 83) function encodes M-text **mt** and writes the resulting byte sequence into the buffer area or the stream that is currently bound to code converter **converter**.

Return value:

If the operation was successful, **mconv_encode()** (p. 83) returns the number of written bytes. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_IO`, `MERROR_CODING`

See Also:

mconv_rebind_buffer() (p. 81), **mconv_rebind_stream()** (p. 81), **mconv_decode()** (p. 82), **mconv_encode_range()** (p. 83)

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

Encode a part of an M-text.

The **mconv_encode_range()** (p. 83) function encodes the text between **from** (inclusive) and **to** (exclusive) in M-text **mt** and writes the resulting byte sequence into the buffer area or the stream that is currently bound to code converter **converter**.

Return value:

If the operation was successful, **mconv_encode_range()** (p. 83) returns the number of written bytes. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`, `MERROR_IO`, `MERROR_CODING`

See Also:

mconv_rebind_buffer() (p. 81), **mconv_rebind_stream()** (p. 81), **mconv_decode()** (p. 82), **mconv_encode()** (p. 83)

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

Encode an M-text into a buffer area.

The **mconv_encode_buffer()** (p. 83) function encodes M-text **mt** based on coding system **name** and writes the resulting byte sequence into the buffer area pointed to by **buf**. At most **n** bytes are written. A temporary code converter for encoding is automatically created and freed.

Return value:

If the operation was successful, **mconv_encode_buffer()** (p. 83) returns the number of written bytes. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:
`MERROR_IO`, `MERROR_CODING`

See Also:
mconv_encode() (p. 83), **mconv_encode_stream()** (p. 84)

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

Encode an M-text to write to a stream.

The **mconv_encode_stream()** (p. 84) function encodes M-text **mt** based on coding system **name** and writes the resulting byte sequence to stream **fp**. A temporary code converter for encoding is automatically created and freed.

Return value:
If the operation was successful, **mconv_encode_stream()** (p. 84) returns the number of written bytes. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:
`MERROR_IO`, `MERROR_CODING`

See Also:
mconv_encode() (p. 83), **mconv_encode_buffer()** (p. 83), mconv_encode_file()

2.13.3.17 int mconv_getc (MConverter ∗ *converter*)

Read a character via a code converter.

The **mconv_getc()** (p. 84) function reads one character from the buffer area or the stream that is currently bound to code converter **converter**. The decoder of **converter** is used to decode the byte sequence. The internal status of **converter** is updated appropriately.

Return value:
If the operation was successful, **mconv_getc()** (p. 84) returns the character read in. If the input source reaches EOF, it returns `EOF` without changing the external variable **merror_code** (p. 156). If an error is detected, it returns `EOF` and assigns an error code to **merror_code** (p. 156).

Errors:
`MERROR_CODING`

See Also:
mconv_ungetc() (p. 84), **mconv_putc()** (p. 85), **mconv_gets()** (p. 85)

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

Push a character back to a code converter.

The **mconv_ungetc()** (p. 84) function pushes character **c** back to code converter **converter**. Any number of characters can be pushed back. The lastly pushed back character is firstly read by the subsequent **mconv_getc()** (p. 84) call. The characters pushed back are registered only in **converter**; they are not written to the input source. The internal status of **converter** is updated appropriately.

Return value:
If the operation was successful, **mconv_ungetc()** (p. 84) returns **c**. Otherwise it returns `EOF` and assigns an error code to the external variable **merror_code** (p. 156).

Errors:
`MERROR_CODING`, `MERROR_CHAR`

See Also:
mconv_getc() (p. 84), **mconv_putc()** (p. 85), **mconv_gets()** (p. 85)

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

Write a character via a code converter.

The **mconv_putc()** (p. 85) function writes character **c** to the buffer area or the stream that is currently bound to code converter **converter**. The encoder of **converter** is used to encode the character. The number of bytes actually written is set to the `nbytes` member of **converter**. The internal status of **converter** is updated appropriately.

Return value:
If the operation was successful, **mconv_putc()** (p. 85) returns **c**. If an error is detected, it returns `EOF` and assigns an error code to the external variable **merror_code** (p. 156).

Errors:
`MERROR_CODING`, `MERROR_IO`, `MERROR_CHAR`

See Also:
mconv_getc() (p. 84), **mconv_ungetc()** (p. 84), **mconv_gets()** (p. 85)

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

Read a line using a code converter.

The **mconv_gets()** (p. 85) function reads one line from the buffer area or the stream that is currently bound to code converter **converter**. The decoder of **converter** is used for decoding. The decoded character sequence is appended at the end of M-text **mt**. The final newline character in the original byte sequence is not appended. The internal status of **converter** is updated appropriately.

Return value:
If the operation was successful, **mconv_gets()** (p. 85) returns the modified **mt**. If it encounters EOF without reading a single character, it returns **mt** without changing it. If an error is detected, it returns `NULL` and assigns an error code to **merror_code** (p. 156).

Errors:
`MERROR_CODING`

See Also:
mconv_getc() (p. 84), **mconv_ungetc()** (p. 84), **mconv_putc()** (p. 85)

2.13.4 Variable Documentation

2.13.4.1 MSymbol Mcoding_us_ascii

Symbol for the coding system US-ASCII.

The symbol **Mcoding_us_ascii** (p. 85) has name `"us-ascii"` and represents a coding system for the CES US-ASCII.

2.13.4.2 MSymbol Mcoding_iso_8859_1

Symbol for the coding system ISO-8859-1.

The symbol **Mcoding_iso_8859_1** (p. 86) has name `"iso-8859-1"` and represents a coding system for the CES ISO-8859-1.

2.13.4.3 MSymbol Mcoding_utf_8

Symbol for the coding system UTF-8.

The symbol **Mcoding_utf_8** (p. 86) has name `"utf-8"` and represents a coding system for the CES UTF-8.

2.13.4.4 MSymbol Mcoding_utf_8_full

Symbol for the coding system UTF-8-FULL.

The symbol **Mcoding_utf_8_full** (p. 86) has name `"utf-8-full"` and represents a coding system that is a extension of UTF-8. This coding system uses the same encoding algorithm as UTF-8 but is not limited to the Unicode characters. It can encode all characters supported by the m17n library.

2.13.4.5 MSymbol Mcoding_utf_16

Symbol for the coding system UTF-16.

The symbol **Mcoding_utf_16** (p. 86) has name `"utf-16"` and represents a coding system for the CES UTF-16 (RFC 2279).

2.13.4.6 MSymbol Mcoding_utf_16be

Symbol for the coding system UTF-16BE.

The symbol **Mcoding_utf_16be** (p. 86) has name `"utf-16be"` and represents a coding system for the CES UTF-16BE (RFC 2279).

2.13.4.7 MSymbol Mcoding_utf_16le

Symbol for the coding system UTF-16LE.

The symbol **Mcoding_utf_16le** (p. 86) has name `"utf-16le"` and represents a coding system for the CES UTF-16LE (RFC 2279).

2.13.4.8 MSymbol Mcoding_utf_32

Symbol for the coding system UTF-32.

The symbol **Mcoding_utf_32** (p. 86) has name `"utf-32"` and represents a coding system for the CES UTF-32 (RFC 2279).

2.13.4.9 MSymbol Mcoding_utf_32be

Symbol for the coding system UTF-32BE.

The symbol **Mcoding_utf_32be** (p. 86) has name `"utf-32be"` and represents a coding system for the CES UTF-32BE (RFC 2279).

2.13.4.10 MSymbol Mcoding_utf_32le

Symbol for the coding system UTF-32LE.

The symbol **Mcoding_utf_32le** (p. 87) has name `"utf-32le"` and represents a coding system for the CES UTF-32LE (RFC 2279).

2.13.4.11 MSymbol Mcoding_sjis

Symbol for the coding system SJIS.

The symbol **Mcoding_sjis** (p. 87) has name `"sjis"` and represents a coding system for the CES Shift-JIS.

2.13.4.12 MSymbol Mtype

Parameter key for **mconv_define_coding()** (p. 77) (which see).

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

Symbol that can be a value of the **Mtype** (p. 87) parameter of a coding system used in an argument to the **mconv_define_coding()** (p. 77) function (which see).

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

Symbol whose name is "maybe".

The variable **Mmaybe** (p. 88) is a symbol of name `"maybe"`. It is used a value of **Mbom** parameter of the function **mconv_define_coding()** (p. 77) (which see).

2.13.4.38 MSymbol Mcoding

The symbol `Mcoding`.

Any decoded M-text has a text property whose key is the predefined symbol `Mcoding`. The name of `Mcoding` is `"coding"`.

2.14 Locale

Locale objects and API for them.

Typedefs

- typedef struct **MLocale** **MLocale**

  `struct MLocale`.

Functions

- **MLocale** ∗ **mlocale_set** (int category, const char ∗name)

  Set the current locale.

- **MSymbol** **mlocale_get_prop** (**MLocale** ∗locale, **MSymbol** key)

  Get the value of a locale property.

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

  Format date and time.

- **MText** ∗ **mtext_getenv** (const char ∗name)

  Get an environment variable.

- int **mtext_putenv** (**MText** ∗mt)

  Change or add an environment variable.

- int **mtext_coll** (**MText** ∗mt1, **MText** ∗mt2)

  Compare two M-texts using the current locale.

Variables

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

2.14.1 Detailed Description

Locale objects and API for them.

The m17n library represents locale related information as objects of type **MLocale** (p. 89).

2.14.2 Typedef Documentation

2.14.2.1 typedef struct MLocale MLocale

`struct MLocale`.

The structure `MLocale` is used to hold information about name, language, territory, modifier, codeset, and the corresponding coding system of locales.

The contents of this structure are implementation dependent. Its internal structure is concealed from application programs.

See Also:

mlocale_get_prop() (p. 90)

2.14.3 Function Documentation

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

Set the current locale.

The **mlocale_set()** (p. 90) function sets or query a part of the current locale. The part is specified by **category** which must be a valid first argument to `setlocale()`.

If **locale** is not NULL, the locale of the specified part is set to **locale**. If **locale** is not supported by the system, the current locale is not changed.

If **locale** is NULL, the current locale of the specified part is queried.

Return value:

If the call is successful, **mlocale_set()** (p. 90) returns an opaque locale object that corresponds to the locale. The name of the locale can be acquired by the function **mlocale_get_prop()** (p. 90). Otherwise, it returns NULL.

Errors:

`MERROR_LOCALE`

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

Get the value of a locale property.

The **mlocale_get_prop()** (p. 90) function returns the value of a property **key** of local **locale**. **key** must be **Mname** (p. 27), **Mlanguage** (p. 48), **Mterritory** (p. 91), **Mcodeset** (p. 91), **Mmodifier** (p. 91), or **Mcoding** (p. 88).

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

Format date and time.

The **mtext_ftime()** (p. 90) function formats the broken-down time **tm** according to the format specification **format** and append the result to the M-text **mt**. The formating is done according to the locale **locale** (if not NULL) or the current locale (LC_TIME).

The meaning of the arguments **tm** and **format** are the same as those of strftime().

See Also:

strftime().

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

Get an environment variable.

The **mtext_getenv()** (p. 90) function searches the environment variable list for a string that matches the string pointed to by **name**.

If there is a match, the function decodes the value according to the current locale (LC_CTYPE) into an M-text, and return that M-text.

If there is no match, the function returns NULL.

2.14.3.5 int mtext_putenv (MText * *mt*)

Change or add an environment variable.

The **mtext_putenv()** (p. 91) function changes or adds the value of environment variables according to M-text **mt**. It calls the function `putenv` with an argument generated by encoding **mt** according to the current locale (LC_CTYPE).

Return value:
This function returns zero on success, or -1 if an error occurs.

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

Compare two M-texts using the current locale.

The **mtext_coll()** (p. 91) function compares the two M-texts **mt1** and **mt2**. It returns an integer less than, equal to, or greater than zero if **mt1** is found, respectively, to be less than, to match, or to be greater than **mt2**. The comparison is based on texts as appropriate for the current locale (LC_COLLATE).

This function makes use of information that is automatically cached in the M-texts as a text property. So, the second call of this function with **mt1** or **mt2** finishes faster than the first call.

2.14.4 Variable Documentation

2.14.4.1 MSymbol Mterritory

The symbol whose name is "territory".

2.14.4.2 MSymbol Mmodifier

The symbol whose name is "modifier".

2.14.4.3 MSymbol Mcodeset

The symbol whose name is "codeset".

2.15 Input Method (basic)

API for Input method.

Data Structures

- struct **MInputDriver**

  Structure of input method driver.

- struct **MInputMethod**

  Structure of input method.

- struct **MInputContext**

  Structure of input context.

Variables: Predefined symbols for callback commands.

These are the predefined symbols that are used as the `COMMAND` argument of callback functions of an input method driver (see **MInputDriver::callback_list** (p. 191)).

Most of them do not require extra argument nor return any value; exceptions are these:

Minput_get_surrounding_text: When a callback function assigned for this command is called, the first element of **MInputContext::plist** (p. 189) has key **Minteger** (p. 23) and the value specifies which portion of the surrounding text should be retrieved. If the value is positive, it specifies the number of characters following the current cursor position. If the value is negative, the absolute value specifies the number of characters preceding the current cursor position. If the value is zero, it means that the caller just wants to know if the surrounding text is currently supported or not.

If the surrounding text is currently supported, the callback function must set the key of this element to **Mtext** (p. 23) and the value to the retrieved M-text. The length of the M-text may be shorter than the requested number of characters, if the available text is not that long. The length can be zero in the worst case. Or, the length may be longer if an application thinks it is more efficient to return that length.

If the surrounding text is not currently supported, the callback function should return without changing the first element of **MInputContext::plist** (p. 189).

Minput_delete_surrounding_text: When a callback function assigned for this command is called, the first element of **MInputContext::plist** (p. 189) has key **Minteger** (p. 23) and the value specifies which portion of the surrounding text should be deleted in the same way as the case of Minput_get_surrounding_text. The callback function must delete the specified text. It should not alter **MInputContext::plist** (p. 189).

- **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**

Variables: Predefined symbols for special input events.

These are the predefined symbols that are used as the `KEY` argument of **minput_filter()** (p. 97).

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

Variables: Predefined symbols used in input method information.

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

Functions

- **MInputMethod** ∗ **minput_open_im** (**MSymbol** language, **MSymbol** name, void ∗arg)

  Open an input method.

- void **minput_close_im** (**MInputMethod** ∗im)

  Close an input method.

- **MInputContext** ∗ **minput_create_ic** (**MInputMethod** ∗im, void ∗arg)

  Create an input context.

- void **minput_destroy_ic** (**MInputContext** ∗ic)

  Destroy an input context.

- int **minput_filter** (**MInputContext** ∗ic, **MSymbol** key, void ∗arg)

  Filter an input key.

- int **minput_lookup** (**MInputContext** ∗ic, **MSymbol** key, void ∗arg, **MText** ∗mt)

  Look up a text produced in the input context.

- void **minput_set_spot** (**MInputContext** ∗ic, int x, int y, int ascent, int descent, int fontsize, **MText** ∗mt, int pos)

  Set the spot of the input context.

- void **minput_toggle** (**MInputContext** ∗ic)

  Toggle input method.

- void **minput_reset_ic** (**MInputContext** ∗ic)

  Reset an input context.

- **MPlist** ∗ **minput_get_title_icon** (**MSymbol** language, **MSymbol** name)

  Get title and icon filename of an input method.

- **MText** ∗ **minput_get_description** (**MSymbol** language, **MSymbol** name)

Get description text of an input method.

- **MPlist** ∗ **minput_get_command** (**MSymbol** language, **MSymbol** name, **MSymbol** command)

  Get information about input method command(s).

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

  Configure the key sequence of an input method command.

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

  Get information about input method variable(s).

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

  Configure the value of an input method variable.

- char ∗ **minput_config_file** ()

  Get the name of per-user customization file.

- int **minput_save_config** (void)

  Save configurations in per-user customization file.

Obsolete functions

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

  Get a list of variables of an input method (obsolete).

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

  Set the initial value of an input method variable.

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

  Get information about input method commands.

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

  Assign a key sequence to an input method command (obsolete).

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

  Call a callback function.

Typedefs

- typedef struct **MInputMethod MInputMethod**

  *See struct **MInputMethod** (*p. *193).*

- typedef struct **MInputContext MInputContext**

  *See struct **MInputContext** (*p. *186).*

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

  Type of input method callback functions.

Enumerations

- enum **MInputCandidatesChanged** {
  MINPUT_CANDIDATES_LIST_CHANGED = 1,
  MINPUT_CANDIDATES_INDEX_CHANGED = 2,
  MINPUT_CANDIDATES_SHOW_CHANGED = 4,
  MINPUT_CANDIDATES_CHANGED_MAX }

  Bit-masks to specify how candidates of input method is changed.

Variables

- **MSymbol Minput_method**

  Symbol whose name is "input-method".

- **MInputDriver minput_default_driver**

  The default driver for internal input methods.

- **MInputDriver ∗ minput_driver**

  The driver for internal input methods.

- **MSymbol Minput_driver**

2.15.1 Detailed Description

API for Input method.

An input method is an object to enable inputting various characters. An input method is identified by a pair of symbols, LANGUAGE and NAME. This pair decides an input method driver of the input method. An input method driver is a set of functions for handling the input method. There are two kinds of input methods; internal one and foreign one.

- Internal Input Method

  An internal input method has non `Mnil` LANGUAGE, and its body is defined in the m17n database by the tag <Minput_method, LANGUAGE, NAME>. For this kind of input methods, the m17n library uses two predefined input method drivers, one for CUI use and the other for GUI use. Those drivers utilize the input processing engine provided by the m17n library itself. The m17n database may provide input methods that are not limited to a specific language. The database uses `Mt` as LANGUAGE of those input methods.

  An internal input method accepts an input key which is a symbol associated with an input event. As there is no way for the `m17n library` to know how input events are represented in an application program, an application programmer has to convert an input event to an input key by himself. See the documentation of the function **minput_event_to_key**() (p. 150) for the detail.

- Foreign Input Method

  A foreign input method has `Mnil` LANGUAGE, and its body is defined in an external resource (e.g. XIM of X Window System). For this kind of input methods, the symbol NAME must have a property of key **Minput_driver** (p. 107), and the value must be a pointer to an input method driver. Therefore, by preparing a proper driver, any kind of input method can be treated in the framework of the `m17n library`.

  For convenience, the m17n-X library provides an input method driver that enables the input style of OverTheSpot for XIM, and stores **Minput_driver** (p. 107) property of the symbol `Mxim` with a pointer to the driver. See the documentation of m17n GUI API for the detail.

PROCESSING FLOW

The typical processing flow of handling an input method is:

- open an input method
- create an input context for the input method
- filter an input key
- look up a produced text in the input context

2.15.2 Typedef Documentation

2.15.2.1 typedef struct MInputMethod MInputMethod

See struct **MInputMethod** (p. 193).

2.15.2.2 typedef struct MInputContext MInputContext

See struct **MInputContext** (p. 186).

2.15.2.3 typedef void(∗ MInputCallbackFunc)(MInputContext ∗ic, MSymbol command)

Type of input method callback functions.

This is the type of callback functions called from input method drivers. **ic** is a pointer to an input context, **command** is a name of callback for which the function is called.

2.15.3 Enumeration Type Documentation

2.15.3.1 enum MInputCandidatesChanged

Bit-masks to specify how candidates of input method is changed.

Enumerator:

MINPUT_CANDIDATES_LIST_CHANGED

MINPUT_CANDIDATES_INDEX_CHANGED

MINPUT_CANDIDATES_SHOW_CHANGED

MINPUT_CANDIDATES_CHANGED_MAX

2.15.4 Function Documentation

2.15.4.1 MInputMethod∗ minput_open_im (MSymbol *language*, MSymbol *name*, void ∗ *arg*)

Open an input method.

The **minput_open_im()** (p. 96) function opens an input method whose language and name match **language** and **name**, and returns a pointer to the input method object newly allocated.

This function at first decides a driver for the input method as described below.

If **language** is not **Mnil** (p. 17), the driver pointed by the variable **minput_driver** (p. 107) is used.

If **language** is **Mnil** (p. 17) and **name** has the property **Minput_driver** (p. 107), the driver pointed to by the property value is used to open the input method. If **name** has no such a property, `NULL` is returned.

Then, the member **MInputDriver::open_im()** (p. 190) of the driver is called.

arg is set in the member `arg` of the structure **MInputMethod** (p. 193) so that the driver can refer to it.

2.15.4.2 void minput_close_im (MInputMethod ∗ *im*)

Close an input method.

The **minput_close_im()** (p. 97) function closes the input method **im**, which must have been created by **minput_open_im()** (p. 96).

2.15.4.3 MInputContext∗ minput_create_ic (MInputMethod ∗ *im*, void ∗ *arg*)

Create an input context.

The **minput_create_ic()** (p. 97) function creates an input context object associated with input method **im**, and calls callback functions corresponding to **Minput_preedit_start**, **Minput_status_start**, and **Minput_status_draw** in this order.

Return value:

If an input context is successfully created, **minput_create_ic()** (p. 97) returns a pointer to it. Otherwise it returns `NULL`.

2.15.4.4 void minput_destroy_ic (MInputContext ∗ *ic*)

Destroy an input context.

The **minput_destroy_ic()** (p. 97) function destroys the input context **ic**, which must have been created by **minput_create_ic()** (p. 97). It calls callback functions corresponding to **Minput_preedit_done**, **Minput_status_done**, and **Minput_candidates_done** in this order.

2.15.4.5 int minput_filter (MInputContext ∗ *ic*, MSymbol *key*, void ∗ *arg*)

Filter an input key.

The **minput_filter()** (p. 97) function filters input key **key** according to input context **ic**, and calls callback functions corresponding to **Minput_preedit_draw**, **Minput_status_draw**, and **Minput_candidates_draw** if the preedit text, the status, and the current candidate are changed respectively.

To make the input method commit the current preedit text (if any) and shift to the initial state, call this function with **Mnil** (p. 17) as **key**.

To inform the input method about the focus-out event, call this function with **Minput_focus_out** as **key**.

To inform the input method about the focus-in event, call this function with **Minput_focus_in** as **key**.

To inform the input method about the focus-move event (i.e. input spot change within the same input context), call this function with **Minput_focus_move** as **key**.

Return value:

If **key** is filtered out, this function returns 1. In that case, the caller should discard the key. Otherwise, it returns 0, and the caller should handle the key, for instance, by calling the function **minput_lookup()** (p. 98) with the same key.

2.15.4.6 int minput_lookup (MInputContext ∗ *ic*, MSymbol *key*, void ∗ *arg*, MText ∗ *mt*)

Look up a text produced in the input context.

The **minput_lookup()** (p. 98) function looks up a text in the input context **ic**. **key** must be identical to the one that was used in the previous call of **minput_filter()** (p. 97).

If a text was produced by the input method, it is concatenated to M-text **mt**.

This function calls **MInputDriver::lookup** (p. 191) .

Return value:

If **key** was correctly handled by the input method, this function returns 0. Otherwise, it returns -1, even though some text might be produced in **mt**.

2.15.4.7 void minput_set_spot (MInputContext ∗ *ic*, int *x*, int *y*, int *ascent*, int *descent*, int *fontsize*, MText ∗ *mt*, int *pos*)

Set the spot of the input context.

The **minput_set_spot()** (p. 98) function sets the spot of input context **ic** to coordinate (**x**, **y**) with the height specified by **ascent** and **descent** . The semantics of these values depends on the input method driver.

For instance, a driver designed to work in a CUI environment may use **x** and **y** as the column- and row numbers, and may ignore **ascent** and **descent** . A driver designed to work in a window system may interpret **x** and **y** as the pixel offsets relative to the origin of the client window, and may interpret **ascent** and **descent** as the ascent- and descent pixels of the line at (**x** . **y**).

fontsize specifies the fontsize of preedit text in 1/10 point.

mt and **pos** are the M-text and the character position at the spot. **mt** may be `NULL`, in which case, the input method cannot get information about the text around the spot.

2.15.4.8 void minput_toggle (MInputContext ∗ *ic*)

Toggle input method.

The **minput_toggle()** (p. 98) function toggles the input method associated with input context **ic**.

2.15.4.9 void minput_reset_ic (MInputContext ∗ *ic*)

Reset an input context.

The **minput_reset_ic()** (p. 98) function resets input context **ic** by calling a callback function corresponding to **Minput_reset**. It resets the status of **ic** to its initial one. As the current preedit text is deleted without commitment, if necessary, call **minput_filter()** (p. 97) with the arg **key** **Mnil** (p. 17) to force the input method to commit the preedit in advance.

2.15.4.10 MPlist∗ minput_get_title_icon (MSymbol *language*, MSymbol *name*)

Get title and icon filename of an input method.

The **minput_get_title_icon()** (p. 98) function returns a plist containing a title and icon filename (if any) of an input method specified by **language** and **name**.

The first element of the plist has key **Mtext** (p. 23) and the value is an M-text of the title for identifying the input method. The second element (if any) has key **Mtext** (p. 23) and the value is an M-text of the icon image (absolute) filename for the same purpose.

Return value:

If there exists a specified input method and it defines an title, a plist is returned. Otherwise, NULL is returned. The caller must free the plist by **m17n_object_unref()** (p. 12).

2.15.4.11 MText∗ minput_get_description (MSymbol *language*, MSymbol *name*)

Get description text of an input method.

The **minput_get_description()** (p. 99) function returns an M-text that describes the input method specified by **language** and **name**.

Return value:

If the specified input method has a description text, a pointer to **MText** (p. 36) is returned. The caller has to free it by **m17n_object_unref()** (p. 12). If the input method does not have a description text, `NULL` is returned.

2.15.4.12 MPlist∗ minput_get_command (MSymbol *language*, MSymbol *name*, MSymbol *command*)

Get information about input method command(s).

The **minput_get_command()** (p. 99) function returns information about the command **command** of the input method specified by **language** and **name**. An input method command is a pseudo key event to which one or more actual input key sequences are assigned.

There are two kinds of commands, global and local. A global command has a global definition, and the description and the key assignment may be inherited by a local command. Each input method defines a local command which has a local key assignment. It may also declare a local command that inherits the definition of a global command of the same name.

If **language** is **Mt** (p. 17) and **name** is **Mnil** (p. 17), this function returns information about a global command. Otherwise information about a local command is returned.

If **command** is **Mnil** (p. 17), information about all commands is returned.

The return value is a *well-formed* plist (**Property List** (p. 18)) of this format:

```
((NAME DESCRIPTION STATUS [KEYSEQ ...]) ...)
```

`NAME` is a symbol representing the command name.

`DESCRIPTION` is an M-text describing the command, or **Mnil** (p. 17) if the command has no description.

`STATUS` is a symbol representing how the key assignment is decided. The value is **Mnil** (p. 17) (the default key assignment), **Mcustomized** (the key assignment is customized by per-user customization file), or **Mconfigured** (the key assignment is set by the call of **minput_config_command()** (p. 100)). For a local command only, it may also be **Minherited** (the key assignment is inherited from the corresponding global command).

`KEYSEQ` is a plist of one or more symbols representing a key sequence assigned to the command. If there's no KEYSEQ, the command is currently disabled (i.e. no key sequence can trigger actions of the command).

If **command** is not **Mnil** (p. 17), the first element of the returned plist contains the information about **command**.

Return value:

If the requested information was found, a pointer to a non-empty plist is returned. As the plist is kept in the library, the caller must not modify nor free it.

Otherwise (the specified input method or the specified command does not exist), `NULL` is returned.

Example:

```
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*)

Configure the key sequence of an input method command.

The **minput_config_command()** (p. 100) function assigns a list of key sequences **keyseqlist** to the command **command** of the input method specified by **language** and **name**.

If **keyseqlist** is a non-empty plist, it must be a list of key sequences, and each key sequence must be a plist of symbols.

If **keyseqlist** is an empty plist, any configuration and customization of the command are cancelled, and default key sequences become effective.

If **keyseqlist** is NULL, the configuration of the command is canceled, and the original key sequences (what saved in per-user customization file, or the default one) become effective.

In the latter two cases, **command** can be **Mnil** (p. 17) to make all the commands of the input method the target of the operation.

If **name** is **Mnil** (p. 17), this function configures the key assignment of a global command, not that of a specific input method.

The configuration takes effect for input methods opened or re-opened later in the current session. In order to make the configuration take effect for the future session, it must be saved in a per-user customization file by the function **minput_save_config()** (p. 103).

Return value:

If the operation was successful, this function returns 0, otherwise returns -1. The operation fails in these cases:

- **keyseqlist** is not in a valid form.
- **command** is not available for the input method.
- **language** and **name** do not specify an existing input method.

See Also:

minput_get_commands() (p. 104), **minput_save_config()** (p. 103).

Example:

```
/* 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*)

Get information about input method variable(s).

The **minput_get_variable()** (p. 101) function returns information about variable **variable** of the input method specified by **language** and **name**. An input method variable controls behavior of an input method.

There are two kinds of variables, global and local. A global variable has a global definition, and the description and the value may be inherited by a local variable. Each input method defines a local variable which has local value. It may also declare a local variable that inherits definition of a global variable of the same name.

If **language** is **Mt** (p. 17) and **name** is **Mnil** (p. 17), information about a global variable is returned. Otherwise information about a local variable is returned.

If **variable** is **Mnil** (p. 17), information about all variables is returned.

The return value is a *well-formed* plist (**Property List** (p. 18)) of this format:

```
((NAME DESCRIPTION STATUS VALUE [VALID-VALUE ...]) ...)
```

`NAME` is a symbol representing the variable name.

`DESCRIPTION` is an M-text describing the variable, or **Mnil** (p. 17) if the variable has no description.

`STATUS` is a symbol representing how the value is decided. The value is **Mnil** (p. 17) (the default value), **Mcustomized** (the value is customized by per-user customization file), or **Mconfigured** (the value is set by the call of **minput_config_variable()** (p. 102)). For a local variable only, it may also be **Minherited** (the value is inherited from the corresponding global variable).

`VALUE` is the initial value of the variable. If the key of this element is **Mt** (p. 17), the variable has no initial value. Otherwise, the key is **Minteger** (p. 23), **Msymbol** (p. 17), or **Mtext** (p. 23) and the value is of the corresponding type.

`VALID-VALUEs` (if any) specify which values the variable can have. They have the same type (i.e. having the same key) as `VALUE` except for the case that VALUE is an integer. In that case, `VALID-VALUE` may be a plist of two integers specifying the range of possible values.

If there no `VALID-VALUE`, the variable can have any value as long as the type is the same as `VALUE`.

If **variable** is not **Mnil** (p. 17), the first element of the returned plist contains the information about **variable**.

Return value:

If the requested information was found, a pointer to a non-empty plist is returned. As the plist is kept in the library, the caller must not modify nor free it.

Otherwise (the specified input method or the specified variable does not exist), `NULL` is returned.

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

Configure the value of an input method variable.

The **minput_config_variable()** (p. 102) function assigns **value** to the variable **variable** of the input method specified by **language** and **name**.

If **value** is a non-empty plist, it must be a plist of one element whose key is **Minteger** (p. 23), **Msymbol** (p. 17), or **Mtext** (p. 23), and the value is of the corresponding type. That value is assigned to the variable.

If **value** is an empty plist, any configuration and customization of the variable are canceled, and the default value is assigned to the variable.

If **value** is NULL, the configuration of the variable is canceled, and the original value (what saved in per-user customization file, or the default value) is assigned to the variable.

In the latter two cases, **variable** can be **Mnil** (p. 17) to make all the variables of the input method the target of the operation.

If **name** is **Mnil** (p. 17), this function configures the value of global variable, not that of a specific input method.

The configuration takes effect for input methods opened or re-opened later in the current session. To make the configuration take effect for the future session, it must be saved in a per-user customization file by the function **minput_save_config()** (p. 103).

Return value:

If the operation was successful, this function returns 0, otherwise returns -1. The operation fails in these cases:

- **value** is not in a valid form, the type does not match the definition, or the value is our of range.
- **variable** is not available for the input method.
- **language** and **name** do not specify an existing input method.

See Also:
minput_get_variable() (p. 101), **minput_save_config()** (p. 103).

2.15.4.16 char∗ minput_config_file (void)

Get the name of per-user customization file.

The **minput_config_file()** (p. 102) function returns the absolute path name of per-user customization file into which **minput_save_config()** (p. 103) save configurations. It is usually `config.mic` under the directory `${HOME}/.m17n.d` (${HOME} is user's home directory). It is not assured that the file of the returned name exists nor is readable/writable. If **minput_save_config()** (p. 103) fails and returns -1, an application program might check the file, make it writable (if possible), and try **minput_save_config()** (p. 103) again.

Return value:

This function returns a string. As the string is kept in the library, the caller must not modify nor free it.

See Also:

minput_save_config() (p. 103)

2.15.4.17 int minput_save_config (void)

Save configurations in per-user customization file.

The **minput_save_config()** (p. 103) function saves the configurations done so far in the current session into the per-user customization file.

Return value:

If the operation was successful, 1 is returned. If the per-user customization file is currently locked, 0 is returned. In that case, the caller may wait for a while and try again. If the configuration file is not writable, -1 is returned. In that case, the caller may check the name of the file by calling **minput_config_file()** (p. 102), make it writable if possible, and try again.

See Also:

minput_config_file() (p. 102)

2.15.4.18 MPlist∗ minput_get_variables (MSymbol *language*, MSymbol *name*)

Get a list of variables of an input method (obsolete).

This function is obsolete. Use **minput_get_variable()** (p. 101) instead.

The **minput_get_variables()** (p. 103) function returns a plist (**MPlist** (p. 19)) of variables used to control the behavior of the input method specified by **language** and **name**. The plist is *well-formed* (**Property List** (p. 18)) of the following format:

```
(VARNAME (DOC-MTEXT DEFAULT-VALUE [ VALUE ... ] )
 VARNAME (DOC-MTEXT DEFAULT-VALUE [ VALUE ... ] )
 ...)
```

`VARNAME` is a symbol representing the variable name.

`DOC-MTEXT` is an M-text describing the variable.

`DEFAULT-VALUE` is the default value of the variable. It is a symbol, integer, or M-text.

`VALUEs` (if any) specifies the possible values of the variable. If `DEFAULT-VALUE` is an integer, `VALUE` may be a plist (`FROM TO`), where `FROM` and `TO` specifies a range of possible values.

For instance, suppose an input method has the variables:

- name:intvar, description:"value is an integer", initial value:0, value-range:0..3,10,20
- name:symvar, description:"value is a symbol", initial value:nil, value-range:a, b, c, nil
- name:txtvar, description:"value is an M-text", initial value:empty text, no value-range (i.e. any text)

Then, the returned plist is as follows.

```
(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" ""))
```

Return value:

If the input method uses any variables, a pointer to **MPlist** (p. 19) is returned. As the plist is kept in the library, the caller must not modify nor free it. If the input method does not use any variable, `NULL` is returned.

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

Set the initial value of an input method variable.

The **minput_set_variable()** (p. 104) function sets the initial value of input method variable **variable** to **value** for the input method specified by **language** and **name**.

By default, the initial value is 0.

This setting gets effective in a newly opened input method.

Return value:

If the operation was successful, 0 is returned. Otherwise -1 is returned, and **merror_code** (p. 156) is set to `MERROR_IM`.

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

Get information about input method commands.

The **minput_get_commands()** (p. 104) function returns information about input method commands of the input method specified by **language** and **name**. An input method command is a pseudo key event to which one or more actual input key sequences are assigned.

There are two kinds of commands, global and local. Global commands are used by multiple input methods for the same purpose, and have global key assignments. Local commands are used only by a specific input method, and have only local key assignments.

Each input method may locally change key assignments for global commands. The global key assignment for a global command is effective only when the current input method does not have local key assignments for that command.

If **name** is **Mnil** (p. 17), information about global commands is returned. In this case **language** is ignored.

If **name** is not **Mnil** (p. 17), information about those commands that have local key assignments in the input method specified by **language** and **name** is returned.

Return value:

If no input method commands are found, this function returns `NULL`.

Otherwise, a pointer to a plist is returned. The key of each element in the plist is a symbol representing a command, and the value is a plist of the form COMMAND-INFO described below.

The first element of COMMAND-INFO has the key **Mtext** (p. 23), and the value is an M-text describing the command.

If there are no more elements, that means no key sequences are assigned to the command. Otherwise, each of the remaining elements has the key **Mplist** (p. 23), and the value is a plist whose keys are **Msymbol** (p. 17) and values are symbols representing input keys, which are currently assigned to the command.

As the returned plist is kept in the library, the caller must not modify nor free it.

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

Assign a key sequence to an input method command (obsolete).

This function is obsolete. Use **minput_config_command()** (p. 100) instead.

The **minput_assign_command_keys()** (p. 105) function assigns input key sequence **keyseq** to input method command **command** for the input method specified by **language** and **name**. If **name** is **Mnil** (p. 17), the key sequence is assigned globally no matter what **language** is. Otherwise the key sequence is assigned locally.

Each element of **keyseq** must have the key **msymbol** and the value must be a symbol representing an input key.

keyseq may be `NULL`, in which case, all assignments are deleted globally or locally.

This assignment gets effective in a newly opened input method.

Return value:

If the operation was successful, 0 is returned. Otherwise -1 is returned, and **merror_code** (p. 156) is set to `MERROR_IM`.

2.15.4.22 int minput_callback (MInputContext ∗ *ic*, MSymbol *command*)

Call a callback function.

The **minput_callback()** (p. 105) functions calls a callback function **command** assigned for the input context **ic**. The caller must set specific elements in **ic->plist** if the callback function requires.

Return value:

If there exists a specified callback function, 0 is returned. Otherwise -1 is returned. By side effects, **ic->plist** may be modified.

2.15.5 Variable Documentation

2.15.5.1 MSymbol Minput_method

Symbol whose name is "input-method".

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

These are the predefined symbols describing status of input method command and variable, and are used in a return value of **minput_get_command()** (p. 99) and **minput_get_variable()** (p. 101).

2.15.5.20 MSymbol Mcustomized

2.15.5.21 MSymbol Mconfigured

2.15.5.22 MInputDriver minput_default_driver

The default driver for internal input methods.

The variable **minput_default_driver** (p. 106) is the default driver for internal input methods.

The member **MInputDriver::open_im()** (p. 190) searches the m17n database for an input method that matches the tag < **Minput_method** (p. 105), **language**, **name**> and loads it.

The member **MInputDriver::callback_list()** (p. 191) is `NULL`. Thus, it is programmers responsibility to set it to a plist of proper callback functions. Otherwise, no feedback information (e.g. preedit text) can be shown to users.

The macro **M17N_INIT()** (p. 7) sets the variable **minput_driver** (p. 107) to the pointer to this driver so that all internal input methods use it.

Therefore, unless `minput_driver` is set differently, the driver dependent arguments **arg** of the functions whose name begins with "minput_" are all ignored.

2.15.5.23 MInputDriver∗ minput_driver

The driver for internal input methods.

The variable **minput_driver** (p. 107) is a pointer to the input method driver that is used by internal input methods. The macro **M17N_INIT()** (p. 7) initializes it to a pointer to **minput_default_driver** (p. 106) if <m17n.h> is included.

2.15.5.24 MSymbol Minput_driver

The variable **Minput_driver** (p. 107) is a symbol for a foreign input method. See **foreign input method** (p. 95) for the detail.

2.16 FLT API

API provided by libm17n-flt.so.

Data Structures

- struct **MFLTGlyph**

  Type of information about a glyph.

- struct **MFLTGlyphAdjustment**

  Type of information about a glyph position adjustment.

- struct **MFLTGlyphString**

  Type of information about a glyph sequence.

- struct **MFLTOtfSpec**

  Type of specification of GSUB and GPOS OpenType tables.

- struct **MFLTFont**

  Type of font to be used by the FLT driver.

Typedefs

- typedef struct _MFLT **MFLT**

  Type of FLT (Font Layout Table).

Functions

- **MFLT** ∗ **mflt_get** (**MSymbol** name)

  Return an FLT object that has a specified name.

- **MFLT** ∗ **mflt_find** (int c, **MFLTFont** ∗font)

  Find an FLT suitable for the specified character and font.

- const char ∗ **mflt_name** (**MFLT** ∗flt)

  Return the name of an FLT.

- **MCharTable** ∗ **mflt_coverage** (**MFLT** ∗flt)

  Return a coverage of a FLT.

- int **mflt_run** (**MFLTGlyphString** ∗gstring, int from, int to, **MFLTFont** ∗font, **MFLT** ∗flt)

  Layout characters with an FLT.

- **MFLT** ∗ **mdebug_dump_flt** (**MFLT** ∗flt, int indent)

  Dump a Font Layout Table.

Variables

- **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 Detailed Description

API provided by libm17n-flt.so.

FLT support for a window system.

This section defines the m17n FLT API concerning character layouting facility using FLT (Font Layout Table). The format of FLT is described in **Font Layout Table** (p. 211).

2.16.2 Typedef Documentation

2.16.2.1 typedef struct _MFLT MFLT

Type of FLT (Font Layout Table).

The type **MFLT** (p. 109) is for an FLT object. Its internal structure is concealed from application programs.

2.16.3 Function Documentation

2.16.3.1 MFLT ∗ mflt_get (MSymbol *name*)

Return an FLT object that has a specified name.

The **mflt_get()** (p. 109) function returns an FLT object whose name is **name**.

Return value:

If the operation was successful, **mflt_get()** (p. 109) returns a pointer to the found FLT object. Otherwise, it returns `NULL`.

2.16.3.2 MFLT ∗ mflt_find (int *c*, MFLTFont ∗ *font*)

Find an FLT suitable for the specified character and font.

The **mflt_find()** (p. 109) function returns the most appropriate FLT for layouting character **c** with font **font**.

Return value:

If the operation was successful, **mflt_find()** (p. 109) returns a pointer to the found FLT object. Otherwise, it returns `NULL`.

2.16.3.3 const char ∗ mflt_name (MFLT ∗ *flt*)

Return the name of an FLT.

The **mflt_name()** (p. 109) function returns the name of **flt**.

2.16.3.4 MCharTable ∗ mflt_coverage (MFLT ∗ *flt*)

Return a coverage of a FLT.

The **mflt_coverage()** (p. 110) function returns a char-table that contains nonzero values for characters supported by **flt**.

2.16.3.5 int mflt_run (MFLTGlyphString ∗ *gstring*, int *from*, int *to*, MFLTFont ∗ *font*, MFLT ∗ *flt*)

Layout characters with an FLT.

The **mflt_run()** (p. 110) function layouts characters in **gstring** between **from** (inclusive) and **to** (exclusive) with **font**. If **flt** is nonzero, it is used for all the charaters. Otherwise, appropriate FLTs are automatically chosen.

Return values:

- ***>=0*** The operation was successful. The value is the index to the glyph, which was previously indexed by **to**, in **gstring->glyphs**.
- ***-2*** **gstring->glyphs** is too short to store the result. The caller can call this fucntion again with a longer **gstring->glyphs**.
- ***-1*** Some other error occurred.

2.16.3.6 MFLT∗ mdebug_dump_flt (MFLT ∗ *flt*, int *indent*)

Dump a Font Layout Table.

The **mdebug_dump_flt()** (p. 110) function prints the Font Layout Table **flt** in a human readable way to the stderr. **indent** specifies how many columns to indent the lines but the first one.

Return value:

This function returns **flt**.

2.16.4 Variable Documentation

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

API provided by libm17n-gui.so.

Modules

- **Frame**

  A frame *is an object corresponding to the graphic device.*

- **Font**

  Font object.

- **Fontset**

  A fontset is an object that maps a character to fonts.

- **Face**

  A face is an object to control appearance of M-text.

- **Drawing**

  Drawing M-texts on a window.

- **Input Method (GUI)**

  Input method support on window systems.

2.17.1 Detailed Description

API provided by libm17n-gui.so.

GUI support for a window system.

This section defines the m17n GUI API concerning M-text drawing and inputting under a window system.

All the definitions here are independent of window systems. An actual library file, however, can depend on a specific window system. For instance, the library file m17n-X.so is an example of implementation of the m17n GUI API for the X Window System.

Actually the GUI API is mainly for toolkit libraries or to implement XOM, not for direct use from application programs.

2.18 Frame

A *frame* is an object corresponding to the graphic device.

Variables: Keys of frame parameter

These are the symbols to use in a parameter to create a frame. See the function **mframe()** (p. 113) for details.

Mdevice, **Mdisplay**, **Mscreen**, **Mdrawable**, **Mdepth**, and **Mcolormap** are also keys of a frame property.

- **MSymbol Mdevice**
- **MSymbol Mdisplay**
- **MSymbol Mscreen**
- **MSymbol Mdrawable**
- **MSymbol Mdepth**
- **MSymbol Mcolormap**
- **MSymbol Mwidget**
- **MSymbol Mgd**

Variables: Keys of frame property

These are the symbols to use as an argument to the function **mframe_get_prop()** (p. 114).

- **MSymbol Mfont**
- **MSymbol Mfont_width**
- **MSymbol Mfont_ascent**
- **MSymbol Mfont_descent**

Typedefs

- typedef struct **MFrame MFrame**

  Type of frames.

Functions

- **MFrame** ∗ **mframe** (**MPlist** ∗plist)

  Create a new frame.

- void ∗ **mframe_get_prop** (**MFrame** ∗frame, **MSymbol** key)

  Return property value of frame.

Variables

- **MFrame** ∗ **mframe_default**

  The default frame.

2.18.1 Detailed Description

A *frame* is an object corresponding to the graphic device.

A *frame* is an object of the type **MFrame** (p. 113) to hold various information about each display/input device. Almost all m17n GUI functions require a pointer to a frame as an argument.

2.18.2 Typedef Documentation

2.18.2.1 typedef struct MFrame MFrame

Type of frames.

The type **MFrame** (p. 113) is for a *frame* object. Each frame holds various information about the corresponding physical display/input device.

The internal structure of the type **MFrame** (p. 113) is concealed from an application program, and its contents depend on the window system in use. In the m17n-X library, it contains the information about *display* and *screen* in the X Window System.

2.18.3 Function Documentation

2.18.3.1 MFrame∗ mframe (MPlist ∗ *plist*)

Create a new frame.

The **mframe()** (p. 113) function creates a new frame with parameters listed in **plist** which may be `NULL`.

The recognized keys in **plist** are window system dependent.

The following key is always recognized.

- **Mdevice**, the value must be one of **Mx** (p. 126), **Mgd**, and **Mnil** (p. 17).

  If the value is **Mx** (p. 126), the frame is for X Window System. The argument **MDrawWindow** (p. 144) specified together with the frame must be of type `Window`. The frame is both readable and writable, thus all GUI functions can be used.

  If the value is **Mgd**, the frame is for an image object of GD library. The argument **MDrawWindow** (p. 144) specified together with the frame must be of type `gdImagePtr`. The frame is writable only, thus functions minput_XXX can't be used for the frame.

  If the value is **Mnil** (p. 17), the frame is for a null device. The frame is not writable nor readable, thus functions mdraw_XXX that require the argument **MDrawWindow** (p. 144) and functions minput_XXX can't be used for the frame.

- **Mface** (p. 142), the value must be a pointer to **MFace** (p. 134).

  The value is used as the default face of the frame.

In addition, if the value of the key **Mdevice** is **Mx** (p. 126), the following keys are recognized. They are to specify the root window and the depth of drawables that can be used with the frame.

- **Mdrawable**, the value type must be `Drawable`.

  A parameter of key **Mdisplay** must also be specified. The created frame can be used for drawables whose root window and depth are the same as those of the specified drawable on the specified display.

  When this parameter is specified, the parameter of key **Mscreen** is ignored.

- **Mwidget**, the value type must be `Widget`.

  The created frame can be used for drawables whose root window and depth are the same as those of the specified widget.

  If a parameter of key **Mface** (p. 142) is not specified, the default face is created from the resources of the widget.

  When this parameter is specified, the parameters of key **Mdisplay**, **Mscreen**, **Mdrawable**, **Mdepth** are ignored.

- **Mdepth**, the value type must be `unsigned`.

  The created frame can be used for drawables of the specified depth.

- **Mscreen**, the value type must be `(Screen *)`.

  The created frame can be used for drawables whose root window is the same as the root window of the specified screen, and depth is the same at the default depth of the screen.

  When this parameter is specified, parameter of key **Mdisplay** is ignored.

- **Mdisplay**, the value type must be `(Display *)`.

  The created frame can be used for drawables whose root window is the same as the root window for the default screen of the display, and depth is the same as the default depth of the screen.

- **Mcolormap**, the value type must be `(Colormap)`.

  The created frame uses the specified colormap.

- **Mfont**, the value must be **Mx** (p. 126), **Mfreetype** (p. 126), or **Mxft** (p. 127).

  The created frame uses the specified font backend. The value **Mx** (p. 126) instructs to use X core fonts, **Mfreetype** (p. 126) to use local fonts supported by FreeType fonts, and **Mxft** (p. 127) to use local fonts via Xft library. You can specify this parameter more than once with different values if you want to use multiple font backends. This is ignored if the specified font backend is not supported on the device.

  When this parameter is not specified, all font backend supported on the device are used.

Return value:

If the operation was successful, **mframe()** (p. 113) returns a pointer to a newly created frame. Otherwise, it returns `NULL`.

2.18.3.2 void* mframe_get_prop (MFrame * *frame*, MSymbol *key*)

Return property value of frame.

The **mframe_get_prop()** (p. 114) function returns a value of property **key** of frame **frame**. The valid keys and the corresponding return values are as follows.

key	type of value	meaning of value
Mface	MFace *	The default face.
Mfont	MFont *	The default font.
Mfont_width	int	Width of the default font.
Mfont_ascent	int	Ascent of the default font.
Mfont_descent	int	Descent of the default font.

In the m17n-X library, the followings are also accepted.

```
key             type of value   meaning of value
---             -------------   ----------------
Mdisplay        Display *       Display associated with the frame.

Mscreen         int             Screen number of a screen associated
                                with the frame.

Mcolormap       Colormap        Colormap of the frame.

Mdepth          unsigned        Depth of the frame.
```

2.18.4 Variable Documentation

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

The default frame.

The external variable **mframe_default** (p. 115) contains a pointer to the default frame that is created by the first call of **mframe()** (p. 113).

2.19 Font

Font object.

Variables: Keys of font property.

- **MSymbol Mfoundry**

  Key of font property specifying foundry.

- **MSymbol Mfamily**

  Key of font property specifying family.

- **MSymbol Mweight**

  Key of font property specifying weight.

- **MSymbol Mstyle**

  Key of font property specifying style.

- **MSymbol Mstretch**

  Key of font property specifying stretch.

- **MSymbol Madstyle**

  Key of font property specifying additional style.

- **MSymbol Mspacing**

  Key of font property specifying spacing.

- **MSymbol Mregistry**

  Key of font property specifying registry.

- **MSymbol Msize**

  Key of font property specifying size.

- **MSymbol Motf**

  Key of font property specifying file name.

- **MSymbol Mfontfile**

  Key of font property specifying file name.

- **MSymbol Mresolution**

  Key of font property specifying resolution.

- **MSymbol Mmax_advance**

  Key of font property specifying max advance width.

- **MSymbol Mfontconfig**

  Symbol of name "fontconfig".

- **MSymbol Mx**

  Symbol of name "x".

- **MSymbol Mfreetype**

  Symbol of name "freetype".

- **MSymbol Mxft**

  Symbol of name "xft".

Typedefs

- typedef struct **MFont MFont**

  Type of fonts.

Functions

- **MFont** ∗ **mfont** ()

  Create a new font.

- **MFont** ∗ **mfont_parse_name** (const char ∗name, **MSymbol** format)

  Create a font by parsing a fontname.

- char ∗ **mfont_unparse_name** (**MFont** ∗font, **MSymbol** format)

  Create a fontname from a font.

- **MFont** ∗ **mfont_copy** (**MFont** ∗font)

  Make a copy of a font.

- void ∗ **mfont_get_prop** (**MFont** ∗font, **MSymbol** key)

  Get a property value of a font.

- int **mfont_put_prop** (**MFont** ∗font, **MSymbol** key, void ∗val)

  Put a property value to a font.

- **MSymbol** ∗ **mfont_selection_priority** ()

  Return the font selection priority.

- int **mfont_set_selection_priority** (**MSymbol** ∗keys)

  Set the font selection priority.

- **MFont** ∗ **mfont_find** (**MFrame** ∗frame, **MFont** ∗spec, int ∗score, int max_size)

  Find a font.

- int **mfont_set_encoding** (**MFont** ∗font, **MSymbol** encoding_name, **MSymbol** repertory_name)

  Set encoding of a font.

- char ∗ **mfont_name** (**MFont** ∗font)

  Create a fontname from a font.

- **MFont** ∗ **mfont_from_name** (const char ∗name)

  Create a new font from fontname.

- int **mfont_resize_ratio** (**MFont** ∗font)

  Get resize information of a font.

- **MPlist** ∗ **mfont_list** (**MFrame** ∗frame, **MFont** ∗font, **MSymbol** language, int maxnum)

  Get a list of fonts.

- **MPlist** ∗ **mfont_list_family_names** (**MFrame** ∗frame)

  Get a list of font famiy names.

- int **mfont_check** (**MFrame** ∗frame, **MFontset** ∗fontset, **MSymbol** script, **MSymbol** language, **MFont** ∗font)

  Check the usability of a font.

- int **mfont_match_p** (**MFont** ∗font, **MFont** ∗spec)

  Check is a font matches with a font spec.

- **MFont** ∗ **mfont_open** (**MFrame** ∗frame, **MFont** ∗font)

  Open a font.

- **MFont** ∗ **mfont_encapsulate** (**MFrame** ∗frame, **MSymbol** data_type, void ∗data)

  Encapusulate a font.

- int **mfont_close** (**MFont** ∗font)

  Close a font.

Variables

- **MPlist** ∗ **mfont_freetype_path**

  List of font files and directories that contain font files.

2.19.1 Detailed Description

Font object.

The m17n GUI API represents a font by an object of the type `MFont`. A font can have *font properties*. Like other types of properties, a font property consists of a key and a value. The key of a font property must be one of the following symbols:

`Mfoundry`, `Mfamily`, `Mweight`, `Mstyle`, `Mstretch`, `Madstyle`, `Mregistry`, `Msize`, `Mresolution`, `Mspacing`.

When the key of a font property is `Msize` or `Mresolution`, its value is an integer. Otherwise the value is a symbol.

The notation "xxx property of F" means the font property that belongs to font F and whose key is `Mxxx`.

The value of a foundry property is a symbol representing font foundry information, e.g. adobe, misc, etc.

The value of a family property is a symbol representing font family information, e.g. times, helvetica, etc.

The value of a weight property is a symbol representing weight information, e.g. normal, bold, etc.

The value of a style property is a symbol representing slant information, e.g. normal, italic, etc.

The value of a stretch property is a symbol representing width information, e.g. normal, semicondensed, etc.

The value of an adstyle property is a symbol representing abstract font family information, e.g. serif, sans-serif, etc.

The value of a registry property is a symbol representing registry information, e.g. iso10646-1, iso8895-1, etc.

The value of a size property is an integer representing design size in the unit of 1/10 point.

The value of a resolution property is an integer representing assumed device resolution in the unit of dots per inch (dpi).

The value of a type property is a symbol indicating a font driver; currently Mx or Mfreetype.

The m17n library uses font objects for two purposes: to receive font specification from an application program, and to present available fonts to an application program. When the m17n library presents an available font to an application program, all font properties have a concrete value.

The m17n library supports three kinds of fonts: Window system fonts, FreeType fonts, and OpenType fonts.

- Window system fonts

  The m17n-X library supports all fonts handled by an X server and an X font server. The correspondence between XLFD fields and font properties are shown below.

  ```
  XLFD field                                  property
  ---------------                             --------
  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
  ```

  XLFD fields not listed in the above table are ignored.

- FreeType fonts

  The m17n library, if configured to use the FreeType library, supports all fonts that can be handled by the FreeType library. The variable **mfont_freetype_path** (p. 127) is initialized properly according to the configuration of the m17n library and the environment variable `M17NDIR`. See the documentation of the variable for details.

  If the m17n library is configured to use the fontconfig library, in addition to **mfont_freetype_path** (p. 127), all fonts available via fontconfig are supported.

  The family name of a FreeType font corresponds to the family property. Style names of FreeType fonts correspond to the weight, style, and stretch properties as below.

  ```
  style name          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
  ```

  Style names not listed in the above table are treated as "Regular".

Combination of a platform ID and an encoding ID corresponds to the registry property. For example, if a font has the combination (1 1), the registry property is 1-1. Some frequent combinations have a predefined registry property as below.

```
platform ID         encoding ID     registry property
-----------         -----------     -----------------
0                   3               unicode-bmp
0                   4               unicode-full
1                   0               apple-roman
3                   1               unicode-bmp
3                   1               unicode-full
```

Thus, a font that has two combinations (1 0) and (3 1) corresponds to four font objects whose registries are 1-0, apple-roman, 3-1, and unicode-bmp.

- OpenType fonts

  The m17n library, if configured to use both the FreeType library and the OTF library, supports any OpenType fonts. The list of actually available fonts is created in the same way as in the case of FreeType fonts. If a fontset instructs to use an OpenType font via an FLT (Font Layout Table), and the FLT has an OTF-related command (e.g. otf:deva), the OTF library converts a character sequence to a glyph code sequence according to the OpenType layout tables of the font, and the FreeType library gives a bitmap image for each glyph.

2.19.2 Typedef Documentation

2.19.2.1 typedef struct MFont MFont

Type of fonts.

The type **MFont** (p. 120) is the structure defining fonts. It contains information about the following properties of a font: foundry, family, weight, style, stretch, adstyle, registry, size, and resolution.

This structure is used both for specifying a font in a fontset and for storing information about available system fonts.

The internal structure is concealed from an application program.

See Also:

mfont() (p. 120), **mfont_from_name()** (p. 123), **mfont_find()** (p. 122).

2.19.3 Function Documentation

2.19.3.1 MFont$*$ mfont ()

Create a new font.

The **mfont()** (p. 120) function creates a new font object that has no property.

Return value:

This function returns a pointer to the created font object.

2.19.3.2 MFont$*$ mfont_parse_name (const char $*$ *name*, MSymbol *format*)

Create a font by parsing a fontname.

The **mfont_parse_name()** (p. 120) function creates a new font object. The properties are extracted fontname **name**.

format specifies the format of **name**. If **format** is **Mx** (p. 126), **name** is parsed as XLFD (X Logical Font Description). If **format** is **Mfontconfig** (p. 126), **name** is parsed as Fontconfig's textual representation of font. If **format** is **Mnil** (p. 17), **name** is at first parsed as XLFD, and it it fails, parsed as Fontconfig's representation.

Return value:

If the operation was successful, this function returns a pointer to the created font. Otherwise it returns `NULL`.

2.19.3.3 char∗ mfont_unparse_name (MFont ∗ *font*, MSymbol *format*)

Create a fontname from a font.

The **mfont_unparse_name()** (p. 121) function creates a fontname string from font **font** according to **format**.

format must be **Mx** (p. 126) or **Mfontconfig** (p. 126). If it is **Mx** (p. 126), the fontname is in XLFD (X Logical Font Description) format. If it is **Mfontconfig** (p. 126), the fontname is in the style of Fontconfig's text representation.

Return value:

This function returns a newly allocated fontname string, which is not freed unless the user explicitly does so by free().

2.19.3.4 MFont∗ mfont_copy (MFont ∗ *font*)

Make a copy of a font.

The **mfont_copy()** (p. 121) function returns a new copy of font **font**.

2.19.3.5 void∗ mfont_get_prop (MFont ∗ *font*, MSymbol *key*)

Get a property value of a font.

The **mfont_get_prop()** (p. 121) function gets the value of **key** property of font **font**. **key** must be one of the following symbols:

`Mfoundry`, `Mfamily`, `Mweight`, `Mstyle`, `Mstretch`, `Madstyle`, `Mregistry`, `Msize`, `Mresolution`, `Mspacing`.

If **font** is a return value of **mfont_find()** (p. 122), **key** can also be one of the following symbols:

Mfont_ascent, **Mfont_descent**, **Mmax_advance** (p. 126).

Return value:

If **key** is `Mfoundry`, `Mfamily`, `Mweight`, `Mstyle`, `Mstretch`, `Madstyle`, `Mregistry`, or `Mspacing`, this function returns the corresponding value as a symbol. If the font does not have **key** property, it returns `Mnil`. If **key** is `Msize`, `Mresolution`, **Mfont_ascent**, Mfont_descent, or **Mmax_advance** (p. 126), this function returns the corresponding value as an integer. If the font does not have **key** property, it returns 0. If **key** is something else, it returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

2.19.3.6 int mfont_put_prop (MFont ∗ *font*, MSymbol *key*, void ∗ *val*)

Put a property value to a font.

The **mfont_put_prop()** (p. 121) function puts a font property whose key is **key** and value is **val** to font **font**. **key** must be one of the following symbols:

`Mfoundry`, `Mfamily`, `Mweight`, `Mstyle`, `Mstretch`, `Madstyle`, `Mregistry`, `Msize`, `Mresolution`.

If **key** is `Msize` or `Mresolution`, **val** must be an integer. Otherwise, **val** must be a symbol of a property value name. But, if the name is "nil", a symbol of name "Nil" must be specified.

2.19.3.7 MSymbol* mfont_selection_priority ()

Return the font selection priority.

The **mfont_selection_priority()** (p. 122) function returns a newly created array of six symbols. The elements are the following keys of font properties ordered by priority.

`Mfamily`, `Mweight`, `Mstyle`, `Mstretch`, `Madstyle`, `Msize`.

The m17n library selects the best matching font according to the order of this array. A font that has a different value for a property of lower priority is preferred to a font that has a different value for a property of higher priority.

2.19.3.8 int mfont_set_selection_priority (MSymbol * *keys*)

Set the font selection priority.

The **mfont_set_selection_priority()** (p. 122) function sets font selection priority according to **keys**, which is an array of six symbols. Each element must be one of the below. No two elements must be the same.

`Mfamily`, `Mweight`, `Mstyle`, `Mstretch`, `Madstyle`, `Msize`.

See the documentation of the function **mfont_selection_priority()** (p. 122) for details.

2.19.3.9 MFont* mfont_find (MFrame * *frame*, MFont * *spec*, int * *score*, int *max_size*)

Find a font.

The **mfont_find()** (p. 122) function returns a pointer to the available font that matches best the specification **spec** on frame **frame**.

score, if not NULL, must point to a place to store the score value that indicates how well the found font matches to **spec**. The smaller score means a better match.

2.19.3.10 int mfont_set_encoding (MFont * *font*, MSymbol *encoding_name*, MSymbol *repertory_name*)

Set encoding of a font.

The **mfont_set_encoding()** (p. 122) function sets the encoding information of font **font**.

encoding_name is a symbol representing a charset that has the same encoding as the font.

repertory_name is `Mnil` or a symbol representing a charset that has the same repertory as the font. If it is `Mnil`, whether a specific character is supported by the font is asked to each font driver.

Return value:

If the operation was successful, this function returns 0. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

2.19.3.11 char* mfont_name (MFont * *font*)

Create a fontname from a font.

This function is obsolete. Use mfont_unparse_name instead.

2.19.3.12 MFont∗ mfont_from_name (const char ∗ *name*)

Create a new font from fontname.

This function is obsolete. Use **mfont_parse_name()** (p. 120) instead.

2.19.3.13 int mfont_resize_ratio (MFont ∗ *font*)

Get resize information of a font.

The **mfont_resize_ratio()** (p. 123) function lookups the m17n database <font, reisize> and returns a resizing ratio (in percentage) of FONT. For instance, if the return value is 150, that means that the m17n library uses an 1.5 time bigger font than a specified size.

2.19.3.14 MPlist∗ mfont_list (MFrame ∗ *frame*, MFont ∗ *font*, MSymbol *language*, int *maxnum*)

Get a list of fonts.

The **mfont_list()** (p. 123) functions returns a list of fonts available on frame **frame**. **font**, if not NULL, limits fonts to ones that match with **font**. **language**, if not `Mnil`, limits fonts to ones that support **language**. **maxnum**, if greater than 0, limits the number of fonts.

language argument exists just for backward compatibility, and the use is deprecated. Use **Mlanguage** (p. 48) font property instead. If **font** already has **Mlanguage** (p. 48) property, **language** is ignored.

Return value:

This function returns a plist whose keys are family names and values are pointers to the object MFont. The plist must be freed by **m17n_object_unref()** (p. 12). If no font is found, it returns NULL.

2.19.3.15 MPlist∗ mfont_list_family_names (MFrame ∗ *frame*)

Get a list of font famiy names.

The **mfont_list_family_names()** (p. 123) functions returns a list of font family names available on frame **frame**.

Return value:

This function returns a plist whose keys are **Msymbol** (p. 17) and values are symbols representing font family names. The elements are sorted by alphabetical order. The plist must be freed by **m17n_object_unref()** (p. 12). If not font is found, it returns NULL.

2.19.3.16 int mfont_check (MFrame ∗ *frame*, MFontset ∗ *fontset*, MSymbol *script*, MSymbol *language*, MFont ∗ *font*)

Check the usability of a font.

The **mfont_check()** (p. 123) function checkes if **font** can be used for **script** and **language** in **fontset** on **frame**.

Return value:

If the font is usable, return 1. Otherwise return 0.

2.19.3.17 int mfont_match_p (MFont ∗ *font*, MFont ∗ *spec*)

Check is a font matches with a font spec.

The **mfont_match_p()** (p. 124) function checks if **font** matches with the font-spec **spec**.

Return value:
If the font matches, 1 is returned. Otherwise 0 is returned.

2.19.3.18 MFont∗ mfont_open (MFrame ∗ *frame*, MFont ∗ *font*)

Open a font.

The **mfont_open()** (p. 124) function opens **font** on **frame**, and returns a realized font.

Return value:
If the font was successfully opened, a realized font is returned. Otherwize NULL is returned.

See Also:
mfont_close() (p. 124).

2.19.3.19 MFont∗ mfont_encapsulate (MFrame ∗ *frame*, MSymbol *data_type*, void ∗ *data*)

Encapusulate a font.

The **mfont_encapsulate()** (p. 124) functions realizes a font by encapusulating data **data** or type **data_type** on **frame**. Currently **data_tape** is **Mfontconfig** (p. 126) or **Mfreetype** (p. 126), and **data** points to an object of FcPattern or FT_Face respectively.

Return value:
If the operation was successful, a realized font is returned. Otherwise NULL is return.

See Also:
mfont_close() (p. 124).

2.19.3.20 int mfont_close (MFont ∗ *font*)

Close a font.

The **mfont_close()** (p. 124) function close a realized font **font**. **font** must be opened previously by **mfont_open()** (p. 124) or mfont_encapsulate ().

Return value:
If the operation was successful, 0 is returned. Otherwise, -1 is returned.

See Also:
mfont_open() (p. 124), **mfont_encapsulate()** (p. 124).

2.19.4 Variable Documentation

2.19.4.1 MSymbol Mfoundry

Key of font property specifying foundry.

The variable **Mfoundry** (p. 124) is a symbol of name `"foundry"` and is used as a key of font property and face property. The property value must be a symbol whose name is a foundry name of a font.

2.19.4.2 MSymbol Mfamily

Key of font property specifying family.

The variable **Mfamily** (p. 125) is a symbol of name `"family"` and is used as a key of font property and face property. The property value must be a symbol whose name is a family name of a font.

2.19.4.3 MSymbol Mweight

Key of font property specifying weight.

The variable **Mweight** (p. 125) is a symbol of name `"weight"` and is used as a key of font property and face property. The property value must be a symbol whose name is a weight name of a font (e.g "medium", "bold").

2.19.4.4 MSymbol Mstyle

Key of font property specifying style.

The variable **Mstyle** (p. 125) is a symbol of name `"style"` and is used as a key of font property and face property. The property value must be a symbol whose name is a style name of a font (e.g "r", "i", "o").

2.19.4.5 MSymbol Mstretch

Key of font property specifying stretch.

The variable **Mstretch** (p. 125) is a symbol of name `"stretch"` and is used as a key of font property and face property. The property value must be a symbol whose name is a stretch name of a font (e.g "normal", "condensed").

2.19.4.6 MSymbol Madstyle

Key of font property specifying additional style.

The variable **Madstyle** (p. 125) is a symbol of name `"adstyle"` and is used as a key of font property and face property. The property value must be a symbol whose name is an additional style name of a font (e.g "serif", "", "sans").

2.19.4.7 MSymbol Mspacing

Key of font property specifying spacing.

The variable **Madstyle** (p. 125) is a symbol of name `"spacing"` and is used as a key of font property. The property value must be a symbol whose name specifies the spacing of a font (e.g "p" for proportional, "m" for monospaced).

2.19.4.8 MSymbol Mregistry

Key of font property specifying registry.

The variable **Mregistry** (p. 125) is a symbol of name `"registry"` and is used as a key of font property. The property value must be a symbol whose name is a registry name a font registry (e.g. "iso8859-1", "jisx0208.1983-0").

2.19.4.9 MSymbol Msize

Key of font property specifying size.

The variable **Msize** (p. 126) is a symbol of name `"size"` and is used as a key of font property and face property. The property value must be an integer specifying a font design size in the unit of 1/10 point (on 100 dpi display).

2.19.4.10 MSymbol Motf

Key of font property specifying file name.

The variable **Mfontfile** (p. 126) is a symbol of name `"fontfile"` and is used as a key of font property. The property value must be a symbol whose name is a font file name.

2.19.4.11 MSymbol Mfontfile

Key of font property specifying file name.

The variable **Mfontfile** (p. 126) is a symbol of name `"fontfile"` and is used as a key of font property. The property value must be a symbol whose name is a font file name.

2.19.4.12 MSymbol Mresolution

Key of font property specifying resolution.

The variable **Mresolution** (p. 126) is a symbol of name `"resolution"` and is used as a key of font property and face property. The property value must be an integer to specifying a font resolution in the unit of dots per inch (dpi).

2.19.4.13 MSymbol Mmax_advance

Key of font property specifying max advance width.

The variable **Mmax_advance** (p. 126) is a symbol of name `"max-advance"` and is used as a key of font property. The property value must be an integer specifying a font's max advance value by pixels.

2.19.4.14 MSymbol Mfontconfig

Symbol of name "fontconfig".

The variable **Mfontconfig** (p. 126) is to be used as an argument of the functions **mfont_parse_name()** (p. 120) and **mfont_unparse_name()** (p. 121).

2.19.4.15 MSymbol Mx

Symbol of name "x".

The variable **Mx** (p. 126) is to be used for a value of <type> member of the structure **MDrawGlyph** (p. 170) to specify the type of <fontp> member is actually (XFontStruct ∗).

2.19.4.16 MSymbol Mfreetype

Symbol of name "freetype".

The variable **Mfreetype** (p. 126) is to be used for a value of <type> member of the structure **MDrawGlyph** (p. 170) to specify the type of <fontp> member is actually FT_Face.

2.19.4.17 MSymbol Mxft

Symbol of name "xft".

The variable **Mxft** (p. 127) is to be used for a value of <type> member of the structure **MDrawGlyph** (p. 170) to specify the type of <fontp> member is actually (XftFont ∗).

2.19.4.18 MPlist∗ mfont_freetype_path

List of font files and directories that contain font files.

The variable `mfont_freetype_path` is a plist of FreeType font files and directories that contain FreeType font files. Key of the element is `Mstring`, and the value is a string that represents a font file or a directory.

The macro **M17N_INIT()** (p. 7) sets up this variable to contain the sub-directory "fonts" of the m17n database and the environment variable "M17NDIR". The first call of **mframe()** (p. 113) creates the internal list of the actually available fonts from this variable. Thus, an application program, if necessary, must modify the variable before calling **mframe()** (p. 113). If it is going to add a new element, value must be a string that can be safely freed.

If the m17n library is not configured to use the FreeType library, this variable is not used.

2.20 Fontset

A fontset is an object that maps a character to fonts.

Functions

- **MFontset** ∗ **mfontset** (char ∗name)

  Return a fontset.

- **MSymbol** **mfontset_name** (**MFontset** ∗fontset)

  Return the name of a fontset.

- **MFontset** ∗ **mfontset_copy** (**MFontset** ∗fontset, char ∗name)

  Make a copy of a fontset.

- int **mfontset_modify_entry** (**MFontset** ∗fontset, **MSymbol** script, **MSymbol** language, **MSymbol** charset, **MFont** ∗spec, **MSymbol** layouter_name, int how)

  Modify the contents of a fontset.

- **MPlist** ∗ **mfontset_lookup** (**MFontset** ∗fontset, **MSymbol** script, **MSymbol** language, **MSymbol** charset)

  Lookup a fontset.

2.20.1 Detailed Description

A fontset is an object that maps a character to fonts.

A *fontset* is an object of the type `MFontset`. When drawing an M-text, a fontset provides rules to select a font for each character in the M-text according to the following information.

- The script character property of a character.
- The language text property of a character.
- The charset text property of a character.

The documentation of **mdraw_text()** (p. 145) describes how that information is used.

2.20.2 Function Documentation

2.20.2.1 MFontset ∗ mfontset (char ∗ *name*)

Return a fontset.

The **mfontset()** (p. 128) function returns a pointer to a fontset object of name **name**. If **name** is `NULL`, it returns a pointer to the default fontset.

If no fontset has the name **name**, a new one is created. At that time, if there exists a data <`fontset`, **name**> in the m17n database, the fontset contents are initialized according to the data. If no such data exists, the fontset contents are left vacant.

The macro **M17N_INIT()** (p. 7) creates the default fontset. An application program can modify it before the first call of **mframe()** (p. 113).

Return value:

This function returns a pointer to the found or newly created fontset.

2.20.2.2 MSymbol mfontset_name (MFontset * *fontset*)

Return the name of a fontset.

The **mfontset_name()** (p. 129) function returns the name of fontset **fontset**.

2.20.2.3 MFontset * mfontset_copy (MFontset * *fontset*, char * *name*)

Make a copy of a fontset.

The **mfontset_copy()** (p. 129) function makes a copy of fontset **fontset**, gives it a name **name**, and returns a pointer to the created copy. **name** must not be a name of existing fontset. In such case, this function returns NULL without making a copy.

2.20.2.4 int mfontset_modify_entry (MFontset * *fontset*, MSymbol *script*, MSymbol *language*, MSymbol *charset*, MFont * *spec*, MSymbol *layouter_name*, int *how*)

Modify the contents of a fontset.

The **mfontset_modify_entry()** (p. 129) function associates, in fontset **fontset**, a copy of **font** with the **script / language** pair or with **charset**.

Each font in a fontset is associated with a particular script/language pair, with a particular charset, or with the symbol `Mnil`. The fonts that are associated with the same item make a group.

If **script** is not `Mnil`, it must be a symbol identifying a script. In this case, **language** is either a symbol identifying a language or `Mnil`, and **font** is associated with the **script / language** pair.

If **charset** is not `Mnil`, it must be a symbol representing a charset object. In this case, **font** is associated with that charset.

If both **script** and **charset** are not `Mnil`, two copies of **font** are created. Then one is associated with the **script / language** pair and the other with that charset.

If both **script** and **charset** are `Mnil`, **font** is associated with `Mnil`. This kind of fonts are called *fallback fonts*.

The argument **how** specifies the priority of **font**. If **how** is positive, **font** has the highest priority in the group of fonts that are associated with the same item. If **how** is negative, **font** has the lowest priority. If **how** is zero, **font** becomes the only available font for the associated item; all the other fonts are removed from the group.

If **layouter_name** is not `Mnil`, it must be a symbol representing a **Font Layout Table** (p. 211) (font layout table). In that case, if **font** is selected for drawing an M-text, that font layout table is used to generate a glyph code sequence from a character sequence.

Return value:

If the operation was successful, **mfontset_modify_entry()** (p. 129) returns 0. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_SYMBOL`

2.20.2.5 MPlist * mfontset_lookup (MFontset * *fontset*, MSymbol *script*, MSymbol *language*, MSymbol *charset*)

Lookup a fontset.

The **mfontset_lookup()** (p. 129) function lookups **fontset** and returns a plist that describes the contents of **fontset** corresponding to the specified script, language, and charset.

If **script** is `Mt`, keys of the returned plist are script name symbols for which some fonts are specified and values are NULL.

If **script** is a script name symbol, the returned plist is decided by **language**.

- If **language** is `Mt`, keys of the plist are language name symbols for which some fonts are specified and values are NULL. A key may be `Mt` which means some fallback fonts are specified for the script.
- If **language** is a language name symbol, the plist is a `FONT-GROUP` for the specified script and language. `FONT-GROUP` is a plist whose keys are FLT (FontLayoutTable) name symbols (`Mt` if no FLT is associated with the font) and values are pointers to **MFont** (p. 120).
- If **language** is `Mnil`, the plist is fallback `FONT-GROUP` for the script.

If **script** is `Mnil`, the returned plist is decided as below.

- If **charset** is `Mt`, keys of the returned plist are charset name symbols for which some fonts are specified and values are NULL.
- If **charset** is a charset name symbol, the plist is a `FONT-GROUP` for the charset.
- If **charset** is `Mnil`, the plist is a fallback `FONT-GROUP`.

Return value:

It returns a plist describing the contents of a fontset. The plist should be freed by **m17n_object_unref()** (p. 12).

2.21 Face

A face is an object to control appearance of M-text.

Data Structures

- struct **MFaceHLineProp**

  Type of horizontal line spec of face.

- struct **MFaceBoxProp**

  Type of box spec of face.

Variables: Keys of face property

- **MSymbol Mforeground**

  Key of a face property specifying foreground color.

- **MSymbol Mbackground**

  Key of a face property specifying background color.

- **MSymbol Mvideomode**

  Key of a face property specifying video mode.

- **MSymbol Mratio**

  Key of a face property specifying font size ratio.

- **MSymbol Mhline**

  Key of a face property specifying horizontal line.

- **MSymbol Mbox**

  Key of a face property specifying box.

- **MSymbol Mfontset**

  Key of a face property specifying fontset.

- **MSymbol Mhook_func**

  Key of a face property specifying hook.

- **MSymbol Mhook_arg**

  Key of a face property specifying argument of hook.

Variables: Possible values of #Mvideomode property of face

See the documentation of the variable **Mvideomode** (p. 137).

- **MSymbol Mnormal**
- **MSymbol Mreverse**

Variables: Predefined faces

- **MFace ∗ mface_normal_video**

  Normal video face.

- **MFace ∗ mface_reverse_video**

  Reverse video face.

- **MFace ∗ mface_underline**

  Underline face.

- **MFace ∗ mface_medium**

  Medium face.

- **MFace ∗ mface_bold**

  Bold face.

- **MFace ∗ mface_italic**

  Italic face.

- **MFace ∗ mface_bold_italic**

  Bold italic face.

- **MFace ∗ mface_xx_small**

  Smallest face.

- **MFace ∗ mface_x_small**

  Smaller face.

- **MFace ∗ mface_small**

  Small face.

- **MFace ∗ mface_normalsize**

  Normalsize face.

- **MFace ∗ mface_large**

  Large face.

- **MFace ∗ mface_x_large**

  Larger face.

- **MFace ∗ mface_xx_large**

  Largest face.

- **MFace ∗ mface_black**

  Black face.

- **MFace ∗ mface_white**

  White face.

- **MFace ∗ mface_red**

  Red face.

- **MFace** ∗ **mface_green**

  Green face.

- **MFace** ∗ **mface_blue**

  Blue face.

- **MFace** ∗ **mface_cyan**

  Cyan face.

- **MFace** ∗ **mface_yellow**

  yellow face.

- **MFace** ∗ **mface_magenta**

  Magenta face.

Variables: The other symbols for face handling.

- **MSymbol Mface**

  Key of a text property specifying a face.

Typedefs

- typedef struct **MFace MFace**

  Type of faces.

- typedef void(∗ **MFaceHookFunc**)(**MFace** ∗face, void ∗arg, void ∗info)

  Type of hook function of face.

Functions

- **MFace** ∗ **mface** ()

  Create a new face.

- **MFace** ∗ **mface_copy** (**MFace** ∗face)

  Make a copy of a face.

- int **mface_equal** (**MFace** ∗face1, **MFace** ∗face2)

  Compare faces.

- **MFace** ∗ **mface_merge** (**MFace** ∗dst, **MFace** ∗src)

  Merge faces.

- **MFace** ∗ **mface_from_font** (**MFont** ∗font)

  Make a face from a font.

- void ∗ **mface_get_prop** (**MFace** ∗face, **MSymbol** key)

Get the value of a face property.

- **MFaceHookFunc mface_get_hook** (**MFace** ∗face)

  Get the hook function of a face.

- int **mface_put_prop** (**MFace** ∗face, **MSymbol** key, void ∗val)

  Set a value of a face property.

- int **mface_put_hook** (**MFace** ∗face, **MFaceHookFunc** func)

  Set a hook function to a face.

- void **mface_update** (**MFrame** ∗frame, **MFace** ∗face)

  Update a face.

2.21.1 Detailed Description

A face is an object to control appearance of M-text.

A *face* is an object of the type **MFace** (p. 134) and controls how to draw M-texts. A face has a fixed number of *face properties*. Like other types of properties, a face property consists of a key and a value. A key is one of the following symbols:

Mforeground (p. 137), **Mbackground** (p. 137), **Mvideomode** (p. 137), **Mhline** (p. 137), **Mbox** (p. 138), **Mfoundry** (p. 124), **Mfamily** (p. 125), **Mweight** (p. 125), **Mstyle** (p. 125), **Mstretch** (p. 125), **Madstyle** (p. 125), **Msize** (p. 126), **Mfontset** (p. 138), **Mratio** (p. 137), **Mhook_func** (p. 138), **Mhook_arg** (p. 138)

The notation "xxx property of F" means the face property that belongs to face F and whose key is `Mxxx`.

The M-text drawing functions first search an M-text for the text property whose key is the symbol **Mface** (p. 142), then draw the M-text using the value of that text property. This value must be a pointer to a face object.

If there are multiple text properties whose key is `Mface`, and they are not conflicting one another, properties of those faces are merged and used.

If no faces specify a certain property, the value of the default face is used.

2.21.2 Typedef Documentation

2.21.2.1 typedef struct MFace MFace

Type of faces.

The type **MFace** (p. 134) is the structure of face objects. The internal structure is concealed from an application program.

2.21.2.2 typedef void(∗ MFaceHookFunc)(MFace ∗face, void ∗arg, void ∗info)

Type of hook function of face.

MFaceHookFunc (p. 134) is a type of a hook function of a face.

2.21.3 Function Documentation

2.21.3.1 MFace∗ mface ()

Create a new face.

The **mface()** (p. 135) function creates a new face object that specifies no property.

Return value:
This function returns a pointer to the created face.

2.21.3.2 MFace∗ mface_copy (MFace ∗ *face*)

Make a copy of a face.

The **mface_copy()** (p. 135) function makes a copy of **face** and returns a pointer to the created copy.

2.21.3.3 int mface_equal (MFace ∗ *face1*, MFace ∗ *face2*)

Compare faces.

The **mface_equal()** (p. 135) function compares faces **face1** and **face2**.

Return value:
If two faces have the same property values, return 1. Otherwise return 0.

2.21.3.4 MFace∗ mface_merge (MFace ∗ *dst*, MFace ∗ *src*)

Merge faces.

The **mface_merge()** (p. 135) functions merges the properties of face **src** into **dst**.

Return value:
This function returns **dst**.

2.21.3.5 MFace∗ mface_from_font (MFont ∗ *font*)

Make a face from a font.

The **mface_from_font()** (p. 135) function return a newly created face while reflecting the properties of **font** in its properties.

2.21.3.6 void∗ mface_get_prop (MFace ∗ *face*, MSymbol *key*)

Get the value of a face property.

The **mface_get_prop()** (p. 135) function returns the value of the face property whose key is **key** in face **face**. **key** must be one of the followings:

Mforeground (p. 137), **Mbackground** (p. 137), **Mvideomode** (p. 137), **Mhline** (p. 137), **Mbox** (p. 138), **Mfoundry** (p. 124), **Mfamily** (p. 125), **Mweight** (p. 125), **Mstyle** (p. 125), **Mstretch** (p. 125), **Madstyle** (p. 125), **Msize** (p. 126), **Mfontset** (p. 138), **Mratio** (p. 137), **Mhook_func** (p. 138), **Mhook_arg** (p. 138)

Return value:

The actual type of the returned value depends of **key**. See documentation of the above keys. If an error is detected, it returns `NULL` and assigns an error code to the external variable **merror_code** (p. 156).

See Also:

mface_put_prop() (p. 136), **mface_put_hook()** (p. 136)

Errors:

`MERROR_FACE`

2.21.3.7 MFaceHookFunc mface_get_hook (MFace ∗ *face*)

Get the hook function of a face.

The **mface_get_hook()** (p. 136) function returns the hook function of face **face**.

2.21.3.8 int mface_put_prop (MFace ∗ *face*, MSymbol *key*, void ∗ *val*)

Set a value of a face property.

The **mface_put_prop()** (p. 136) function assigns **val** to the property whose key is **key** in face **face**. **key** must be one the followings:

Mforeground (p. 137), **Mbackground** (p. 137), **Mvideomode** (p. 137), **Mhline** (p. 137), **Mbox** (p. 138), **Mfoundry** (p. 124), **Mfamily** (p. 125), **Mweight** (p. 125), **Mstyle** (p. 125), **Mstretch** (p. 125), **Madstyle** (p. 125), **Msize** (p. 126), **Mfontset** (p. 138), **Mratio** (p. 137), **Mhook_func** (p. 138), **Mhook_arg** (p. 138)

Among them, font related properties (**Mfoundry** (p. 124) through **Msize** (p. 126)) are used as the default values when a font in the fontset of **face** does not specify those values.

The actual type of the returned value depends of **key**. See documentation of the above keys.

Return value:

If the operation was successful, **mface_put_prop()** (p. 136) returns 0. Otherwise it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

See Also:

mface_get_prop() (p. 135)

Errors:

`MERROR_FACE`

2.21.3.9 int mface_put_hook (MFace ∗ *face*, MFaceHookFunc *func*)

Set a hook function to a face.

The mface_set_hook() function sets the hook function of face **face** to **func**.

2.21.3.10 void mface_update (MFrame ∗ *frame*, MFace ∗ *face*)

Update a face.

The **mface_update()** (p. 136) function update face **face** on frame **frame** by calling a hook function of **face** (if any).

2.21.4 Variable Documentation

2.21.4.1 MSymbol Mforeground

Key of a face property specifying foreground color.

The variable **Mforeground** (p. 137) is used as a key of face property. The property value must be a symbol whose name is a color name, or **Mnil** (p. 17).

Mnil (p. 17) means that the face does not specify a foreground color. Otherwise, the foreground of an M-text is drawn by the specified color.

2.21.4.2 MSymbol Mbackground

Key of a face property specifying background color.

The variable **Mbackground** (p. 137) is used as a key of face property. The property value must be a symbol whose name is a color name, or **Mnil** (p. 17).

Mnil (p. 17) means that the face does not specify a background color. Otherwise, the background of an M-text is drawn by the specified color.

2.21.4.3 MSymbol Mvideomode

Key of a face property specifying video mode.

The variable **Mvideomode** (p. 137) is used as a key of face property. The property value must be **Mnormal**, **Mreverse**, or **Mnil** (p. 17).

Mnormal means that an M-text is drawn in normal video mode (i.e. the foreground is drawn by foreground color, the background is drawn by background color).

Mreverse means that an M-text is drawn in reverse video mode (i.e. the foreground is drawn by background color, the background is drawn by foreground color).

Mnil (p. 17) means that the face does not specify a video mode.

2.21.4.4 MSymbol Mratio

Key of a face property specifying font size ratio.

The variable **Mratio** (p. 137) is used as a key of face property. The value RATIO must be an integer.

The value 0 means that the face does not specify a font size ratio. Otherwise, an M-text is drawn by a font of size (FONTSIZE RATIO / 100) where FONTSIZE is a font size specified by the face property **Msize** (p. 126).

2.21.4.5 MSymbol Mhline

Key of a face property specifying horizontal line.

The variable **Mhline** (p. 137) is used as a key of face property. The value must be a pointer to an object of type **MFaceHLineProp** (p. 177), or `NULL`.

The value `NULL` means that the face does not specify this property. Otherwise, an M-text is drawn with a horizontal line by a way specified by the object that the value points to.

2.21.4.6 MSymbol Mbox

Key of a face property specifying box.

The variable **Mbox** (p. 138) is used as a key of face property. The value must be a pointer to an object of type **MFaceBoxProp** (p. 176), or `NULL`.

The value `NULL` means that the face does not specify a box. Otherwise, an M-text is drawn with a surrounding box by a way specified by the object that the value points to.

2.21.4.7 MSymbol Mfontset

Key of a face property specifying fontset.

The variable **Mfontset** (p. 138) is used as a key of face property. The value must be a pointer to an object of type **Mfontset** (p. 138), or `NULL`.

The value `NULL` means that the face does not specify a fontset. Otherwise, an M-text is drawn with a font selected from what specified in the fontset.

2.21.4.8 MSymbol Mhook_func

Key of a face property specifying hook.

The variable **Mhook_func** (p. 138) is used as a key of face property. The value must be a function of type **MFaceHookFunc** (p. 134), or `NULL`.

The value `NULL` means that the face does not specify a hook. Otherwise, the specified function is called before the face is realized.

2.21.4.9 MSymbol Mhook_arg

Key of a face property specifying argument of hook.

The variable **Mhook_arg** (p. 138) is used as a key of face property. The value can be anything that is passed a hook function specified by the face property **Mhook_func** (p. 138).

2.21.4.10 MSymbol Mnormal

2.21.4.11 MSymbol Mreverse

2.21.4.12 MFace∗ mface_normal_video

Normal video face.

The variable **mface_normal_video** (p. 138) points to a face that has the **Mvideomode** (p. 137) property with value **Mnormal**. The other properties are not specified. An M-text drawn with this face appear normal colors (i.e. the foreground is drawn by foreground color, and background is drawn by background color).

2.21.4.13 MFace∗ mface_reverse_video

Reverse video face.

The variable **mface_reverse_video** (p. 138) points to a face that has the **Mvideomode** (p. 137) property with value **Mreverse**. The other properties are not specified. An M-text drawn with this face appear in reversed colors (i.e. the foreground is drawn by background color, and background is drawn by foreground color).

2.21.4.14 MFace∗ mface_underline

Underline face.

The variable **mface_underline** (p. 139) points to a face that has the **Mhline** (p. 137) property with value a pointer to an object of type **MFaceHLineProp** (p. 177). The members of the object are as follows:

```
member  value
-----   -----
type    MFACE_HLINE_UNDER
width   1
color   Mnil
```

The other properties are not specified. An M-text that has this face is drawn with an underline.

2.21.4.15 MFace∗ mface_medium

Medium face.

The variable **mface_medium** (p. 139) points to a face that has the **Mweight** (p. 125) property with value a symbol of name "medium". The other properties are not specified. An M-text that has this face is drawn with a font of medium weight.

2.21.4.16 MFace∗ mface_bold

Bold face.

The variable **mface_bold** (p. 139) points to a face that has the **Mweight** (p. 125) property with value a symbol of name "bold". The other properties are not specified. An M-text that has this face is drawn with a font of bold weight.

2.21.4.17 MFace∗ mface_italic

Italic face.

The variable **mface_italic** (p. 139) points to a face that has the **Mstyle** (p. 125) property with value a symbol of name "italic". The other properties are not specified. An M-text that has this face is drawn with a font of italic style.

2.21.4.18 MFace∗ mface_bold_italic

Bold italic face.

The variable **mface_bold_italic** (p. 139) points to a face that has the **Mweight** (p. 125) property with value a symbol of name "bold", and **Mstyle** (p. 125) property with value a symbol of name "italic". The other properties are not specified. An M-text that has this face is drawn with a font of bold weight and italic style.

2.21.4.19 MFace∗ mface_xx_small

Smallest face.

The variable **mface_xx_small** (p. 139) points to a face that has the **Mratio** (p. 137) property with value 50. The other properties are not specified. An M-text that has this face is drawn with a font whose size is 50% of a normal font.

2.21.4.20 MFace* mface_x_small

Smaller face.

The variable **mface_x_small** (p. 140) points to a face that has the **Mratio** (p. 137) property with value 66. The other properties are not specified. An M-text that has this face is drawn with a font whose size is 66% of a normal font.

2.21.4.21 MFace* mface_small

Small face.

The variable **mface_x_small** (p. 140) points to a face that has the **Mratio** (p. 137) property with value 75. The other properties are not specified. An M-text that has this face is drawn with a font whose size is 75% of a normal font.

2.21.4.22 MFace* mface_normalsize

Normalsize face.

The variable **mface_normalsize** (p. 140) points to a face that has the **Mratio** (p. 137) property with value 100. The other properties are not specified. An M-text that has this face is drawn with a font whose size is the same as a normal font.

2.21.4.23 MFace* mface_large

Large face.

The variable **mface_large** (p. 140) points to a face that has the **Mratio** (p. 137) property with value 120. The other properties are not specified. An M-text that has this face is drawn with a font whose size is 120% of a normal font.

2.21.4.24 MFace* mface_x_large

Larger face.

The variable **mface_x_large** (p. 140) points to a face that has the **Mratio** (p. 137) property with value 150. The other properties are not specified. An M-text that has this face is drawn with a font whose size is 150% of a normal font.

2.21.4.25 MFace* mface_xx_large

Largest face.

The variable **mface_xx_large** (p. 140) points to a face that has the **Mratio** (p. 137) property with value 200. The other properties are not specified. An M-text that has this face is drawn with a font whose size is 200% of a normal font.

2.21.4.26 MFace* mface_black

Black face.

The variable **mface_black** (p. 140) points to a face that has the **Mforeground** (p. 137) property with value a symbol of name "black". The other properties are not specified. An M-text that has this face is drawn with black foreground.

2.21.4.27 MFace∗ mface_white

White face.

The variable **mface_white** (p. 141) points to a face that has the **Mforeground** (p. 137) property with value a symbol of name "white". The other properties are not specified. An M-text that has this face is drawn with white foreground.

2.21.4.28 MFace∗ mface_red

Red face.

The variable **mface_red** (p. 141) points to a face that has the **Mforeground** (p. 137) property with value a symbol of name "red". The other properties are not specified. An M-text that has this face is drawn with red foreground.

2.21.4.29 MFace∗ mface_green

Green face.

The variable **mface_green** (p. 141) points to a face that has the **Mforeground** (p. 137) property with value a symbol of name "green". The other properties are not specified. An M-text that has this face is drawn with green foreground.

2.21.4.30 MFace∗ mface_blue

Blue face.

The variable **mface_blue** (p. 141) points to a face that has the **Mforeground** (p. 137) property with value a symbol of name "blue". The other properties are not specified. An M-text that has this face is drawn with blue foreground.

2.21.4.31 MFace∗ mface_cyan

Cyan face.

The variable **mface_cyan** (p. 141) points to a face that has the **Mforeground** (p. 137) property with value a symbol of name "cyan". The other properties are not specified. An M-text that has this face is drawn with cyan foreground.

2.21.4.32 MFace∗ mface_yellow

yellow face.

The variable **mface_yellow** (p. 141) points to a face that has the **Mforeground** (p. 137) property with value a symbol of name "yellow". The other properties are not specified. An M-text that has this face is drawn with yellow foreground.

2.21.4.33 MFace∗ mface_magenta

Magenta face.

The variable **mface_magenta** (p. 141) points to a face that has the **Mforeground** (p. 137) property with value a symbol of name "magenta". The other properties are not specified. An M-text that has this face is drawn with magenta foreground.

2.21.4.34 MSymbol Mface

Key of a text property specifying a face.

The variable **Mface** (p. 142) is a symbol of name `"face"`. A text property whose key is this symbol must have a pointer to an object of type **MFace** (p. 134). This is a managing key.

2.22 Drawing

Drawing M-texts on a window.

Data Structures

- struct **MDrawControl**

  Type of a text drawing control.

- struct **MDrawMetric**

  Type of metric for glyphs and texts.

- struct **MDrawGlyphInfo**

  Type of information about a glyph.

- struct **MDrawGlyph**

  Type of information about a glyph metric and font.

Typedefs

- typedef void ∗ **MDrawWindow**

  Window system dependent type for a window.

- typedef void ∗ **MDrawRegion**

  Window system dependent type for a region.

Functions

- int **mdraw_text** (**MFrame** ∗frame, **MDrawWindow** win, int x, int y, **MText** ∗mt, int from, int to)

  Draw an M-text on a window.

- int **mdraw_image_text** (**MFrame** ∗frame, **MDrawWindow** win, int x, int y, **MText** ∗mt, int from, int to)

  Draw an M-text on a window as an image.

- int **mdraw_text_with_control** (**MFrame** ∗frame, **MDrawWindow** win, int x, int y, **MText** ∗mt, int from, int to, **MDrawControl** ∗control)

  Draw an M-text on a window with fine control.

- 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)

  Compute text pixel width.

- 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)

  Compute the text dimensions of each character of M-text.

- int **mdraw_coordinates_position** (**MFrame** ∗frame, **MText** ∗mt, int from, int to, int x_offset, int y_offset, **MDrawControl** ∗control)

  Return the character position nearest to the coordinates.

- int **mdraw_glyph_info** (**MFrame** ∗frame, **MText** ∗mt, int from, int pos, **MDrawControl** ∗control, **MDrawGlyphInfo** ∗info)

  Compute information about a glyph.

- int **mdraw_glyph_list** (**MFrame** ∗frame, **MText** ∗mt, int from, int to, **MDrawControl** ∗control, **MDrawGlyph** ∗glyphs, int array_size, int ∗num_glyphs_return)

  Compute information about glyph sequence.

- void **mdraw_text_items** (**MFrame** ∗frame, **MDrawWindow** win, int x, int y, **MDrawTextItem** ∗items, int nitems)

  Draw one or more textitems.

- int **mdraw_default_line_break** (**MText** ∗mt, int pos, int from, int to, int line, int y)

  Calculate a line breaking position.

- void **mdraw_per_char_extents** (**MFrame** ∗frame, **MText** ∗mt, **MDrawMetric** ∗array_return, **MDrawMetric** ∗overall_return)

  Obtain per character dimension information.

- void **mdraw_clear_cache** (**MText** ∗mt)

  clear cached information.

Variables

- int **mdraw_line_break_option**

  Option of line breaking for drawing text.

2.22.1 Detailed Description

Drawing M-texts on a window.

The m17n GUI API provides functions to draw M-texts.

The fonts used for drawing are selected automatically based on the fontset and the properties of a face. A face also specifies the appearance of M-texts, i.e. font size, color, underline, etc.

The drawing format of M-texts can be controlled in a variety of ways, which provides powerful 2-dimensional layout facility.

2.22.2 Typedef Documentation

2.22.2.1 typedef void∗ MDrawWindow

Window system dependent type for a window.

The type **MDrawWindow** (p. 144) is for a window; a rectangular area that works in several ways like a miniature screen.

What it actually points depends on a window system. A program that uses the m17n-X library must coerce the type `Drawable` to this type.

2.22.2.2 typedef void∗ MDrawRegion

Window system dependent type for a region.

The type **MDrawRegion** (p. 145) is for a region; an arbitrary set of pixels on the screen (typically a rectangular area).

What it actually points depends on a window system. A program that uses the m17n-X library must coerce the type `Region` to this type.

2.22.3 Function Documentation

2.22.3.1 int mdraw_text (MFrame ∗ *frame*, MDrawWindow *win*, int *x*, int *y*, MText ∗ *mt*, int *from*, int *to*)

Draw an M-text on a window.

The **mdraw_text()** (p. 145) function draws the text between **from** and **to** of M-text **mt** on window **win** of frame **frame** at coordinate (**x**, **y**).

The appearance of the text (size, style, color, etc) is specified by the value of the text property whose key is `Mface`. If the M-text or a part of the M-text does not have such a text property, the default face of **frame** is used.

The font used to draw a character in the M-text is selected from the value of the fontset property of a face by the following algorithm:

1. Search the text properties given to the character for the one whose key is `Mcharset`; its value should be either a symbol specifying a charset or **Mnil** (p. 17). If the value is **Mnil** (p. 17), proceed to the next step.

   Otherwise, search the mapping table of the fontset for the charset. If no entry is found proceed to the next step.

   If an entry is found, use one of the fonts in the entry that has a glyph for the character and that matches best with the face properties. If no such font exists, proceed to the next step.

2. Get the character property "script" of the character. If it is inherited, get the script property from the previous characters. If there is no previous character, or none of them has the script property other than inherited, proceed to the next step.

   Search the text properties given to the character for the one whose key is `Mlanguage`; its value should be either a symbol specifying a language or `Mnil`.

   Search the mapping table of the fontset for the combination of the script and language. If no entry is found, proceed to the next step.

   If an entry is found, use one of the fonts in the entry that has a glyph for the character and that matches best with the face properties. If no such font exists, proceed to the next step.

3. Search the fall-back table of the fontset for a font that has a glyph of the character. If such a font is found, use that font.

If no font is found by the algorithm above, this function draws an empty box for the character.

This function draws only the glyph foreground. To specify the background color, use **mdraw_image_text()** (p. 146) or **mdraw_text_with_control()** (p. 146).

This function is the counterpart of `XDrawString()`, `XmbDrawString()`, and `XwcDrawString()` functions in the X Window System.

Return value:

If the operation was successful, **mdraw_text()** (p. 145) returns 0. If an error is detected, it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`

See Also:

mdraw_image_text() (p. 146)

2.22.3.2 int mdraw_image_text (MFrame ∗ *frame*, MDrawWindow *win*, int *x*, int *y*, MText ∗ *mt*, int *from*, int *to*)

Draw an M-text on a window as an image.

The **mdraw_image_text()** (p. 146) function draws the text between **from** and **to** of M-text **mt** as image on window **win** of frame **frame** at coordinate (**x**, **y**).

The way to draw a text is the same as in **mdraw_text()** (p. 145) except that this function also draws the background with the color specified by faces.

This function is the counterpart of `XDrawImageString()`, `XmbDrawImageString()`, and `XwcDrawImageString()` functions in the X Window System.

Return value:

If the operation was successful, **mdraw_image_text()** (p. 146) returns 0. If an error is detected, it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`MERROR_RANGE`

See Also:

mdraw_text() (p. 145)

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*)

Draw an M-text on a window with fine control.

The **mdraw_text_with_control()** (p. 146) function draws the text between **from** and **to** of M-text **mt** on windows **win** of frame **frame** at coordinate (**x**, **y**).

The way to draw a text is the same as in **mdraw_text()** (p. 145) except that this function also follows what specified in the drawing control object **control**.

For instance, if <two_dimensional> of **control** is nonzero, this function draw an M-text 2-dimensionally, i.e., newlines in M-text breaks lines and the following characters are drawn in the next line. See the documentation of the structure @ **MDrawControl** (p. 166) for more detail.

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*)

Compute text pixel width.

The **mdraw_text_extents()** (p. 146) function computes the width of text between **from** and **to** of M-text **mt** when it is drawn on a window of frame **frame** using the **mdraw_text_with_control()** (p. 146) function with the drawing control object **control**.

If **overall_ink_return** is not `NULL`, this function also computes the bounding box of character ink of the M-text, and stores the results in the members of the structure pointed to by **overall_ink_return**. If the M-text has a face specifying a surrounding box, the box is included in the bounding box.

If **overall_logical_return** is not `NULL`, this function also computes the bounding box that provides minimum spacing to other graphical features (such as surrounding box) for the M-text, and stores the results in the members of the structure pointed to by **overall_logical_return**.

If **overall_line_return** is not `NULL`, this function also computes the bounding box that provides minimum spacing to the other M-text drawn, and stores the results in the members of the structure pointed to by **overall_line_return**. This is a union of **overall_ink_return** and **overall_logical_return** if the members min_line_ascent, min_line_descent, max_line_ascent, and max_line_descent of **control** are all zero.

Return value:

This function returns the width of the text to be drawn in the unit of pixels. If **control->two_dimensional** is nonzero and the text is drawn in multiple physical lines, it returns the width of the widest line. If an error occurs, it returns -1 and assigns an error code to the external variable **merror_code** (p. 156).

Errors:

`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*)

Compute the text dimensions of each character of M-text.

The **mdraw_text_per_char_extents()** (p. 147) function computes the drawn metric of each character between **from** and **to** of M-text **mt** assuming that they are drawn on a window of frame **frame** using the **mdraw_text_with_control()** (p. 146) function with the drawing control object **control**.

array_size specifies the size of **ink_array_return** and **logical_array_return**. Each successive element of **ink_array_return** and **logical_array_return** are set to the drawn ink and logical metrics of successive characters respectively, relative to the drawing origin of the M-text. The number of elements of **ink_array_return** and **logical_array_return** that have been set is returned to **num_chars_return**.

If **array_size** is too small to return all metrics, the function returns -1 and store the requested size in **num_chars_return**. Otherwise, it returns zero.

If pointer **overall_ink_return** and **overall_logical_return** are not `NULL`, this function also computes the metrics of the overall text and stores the results in the members of the structure pointed to by **overall_ink_return** and **overall_logical_return**.

If **control->two_dimensional** is nonzero, this function computes only the metrics of characters in the first line.

2.22.3.6 int mdraw_coordinates_position (MFrame ∗ *frame*, MText ∗ *mt*, int *from*, int *to*, int *x_offset*, int *y_offset*, MDrawControl ∗ *control*)

Return the character position nearest to the coordinates.

The **mdraw_coordinates_position()** (p. 147) function checks which character is to be drawn at coordinate (**x**, **y**) when the text between **from** and **to** of M-text **mt** is drawn at the coordinate (0, 0) using the **mdraw_text_with_control()** (p. 146) function with the drawing control object **control**. Here, the character position means the number of characters that precede the character in question in **mt**, that is, the character position of the first character is 0.

frame is used only to get the default face information.

Return value:

If the glyph image of a character covers coordinate (**x**, **y**), **mdraw_coordinates_position()** (p. 147) returns the character position of that character.
If **y** is less than the minimum Y-coordinate of the drawn area, it returns **from**.
If **y** is greater than the maximum Y-coordinate of the drawn area, it returns **to**.
If **y** fits in with the drawn area but **x** is less than the minimum X-coordinate, it returns the character position of the first character drawn on the line **y**.
If **y** fits in with the drawn area but **x** is greater than the maximum X-coordinate, it returns the character position of the last character drawn on the line **y**.

2.22.3.7 int mdraw_glyph_info (MFrame ∗ *frame*, MText ∗ *mt*, int *from*, int *pos*, MDrawControl ∗ *control*, MDrawGlyphInfo ∗ *info*)

Compute information about a glyph.

The **mdraw_glyph_info()** (p. 148) function computes information about a glyph that covers a character at position **pos** of the M-text **mt** assuming that the text is drawn from the character at **from** of **mt** on a window of frame **frame** using the **mdraw_text_with_control()** (p. 146) function with the drawing control object **control**.

The information is stored in the members of **info**.

See Also:

MDrawGlyphInfo (p. 172)

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*)

Compute information about glyph sequence.

The **mdraw_glyph_list()** (p. 148) function computes information about glyphs corresponding to the text between **from** and **to** of M-text **mt** when it is drawn on a window of frame **frame** using the **mdraw_text_with_control()** (p. 146) function with the drawing control object **control**. **glyphs** is an array of objects to store the information, and **array_size** is the array size.

If **array_size** is large enough to cover all glyphs, it stores the number of actually filled elements in the place pointed by **num_glyphs_return**, and returns 0.

Otherwise, it stores the required array size in the place pointed by **num_glyphs_return**, and returns -1.

See Also:

MDrawGlyph (p. 170)

2.22.3.9 void mdraw_text_items (MFrame ∗ *frame*, MDrawWindow *win*, int *x*, int *y*, MDrawTextItem ∗ *items*, int *nitems*)

Draw one or more textitems.

The **mdraw_text_items()** (p. 148) function draws one or more M-texts on window **win** of frame **frame** at coordinate (**x**, **y**). **items** is an array of the textitems to be drawn and **nitems** is the number of textitems in the array.

See Also:

MTextItem, **mdraw_text()** (p. 145).

2.22.3.10 int mdraw_default_line_break (MText * *mt*, int *pos*, int *from*, int *to*, int *line*, int *y*)

Calculate a line breaking position.

The function **mdraw_default_line_break()** (p. 149) calculates a line breaking position based on the line number **line** and the coordinate **y**, when a line is too long to fit within the width limit. **pos** is the position of the character next to the last one that fits within the limit. **from** is the position of the first character of the line, and **to** is the position of the last character displayed on the line if there were not width limit. **line** and **y** are reset to 0 when a line is broken by a newline character, and incremented each time when a long line is broken because of the width limit.

Return value:

This function returns a character position to break the line.

2.22.3.11 void mdraw_per_char_extents (MFrame * *frame*, MText * *mt*, MDrawMetric * *array_return*, MDrawMetric * *overall_return*)

Obtain per character dimension information.

The **mdraw_per_char_extents()** (p. 149) function computes the text dimension of each character in M-text **mt**. The faces given as text properties in **mt** and the default face of frame **frame** determine the fonts to draw the text. Each successive element in **array_return** is set to the drawn metrics of successive characters, which is relative to the origin of the drawing, and a rectangle for each character in **mt**. The number of elements of **array_return** must be equal to or greater than the number of characters in **mt**.

If pointer **overall_return** is not `NULL`, this function also computes the extents of the overall text and stores the results in the members of the structure pointed to by **overall_return**.

2.22.3.12 void mdraw_clear_cache (MText * *mt*)

clear cached information.

The **mdraw_clear_cache()** (p. 149) function clear cached information on M-text **mt** that was attached by any of the drawing functions. When the behavior of ‘format’ or ‘line_break’ member functions of **MDrawControl** (p. 166) is changed, the cache must be cleared.

See Also:

MDrawControl (p. 166)

2.22.4 Variable Documentation

2.22.4.1 int mdraw_line_break_option

Option of line breaking for drawing text.

The variable **mdraw_line_break_option** (p. 149) specifies line breaking options by logical-or of the members of **MTextLineBreakOption** (p. 37). It controls the line breaking algorithm of the function **mdraw_default_line_break()** (p. 149).

2.23 Input Method (GUI)

Input method support on window systems.

Data Structures

- struct **MInputGUIArgIC**

  Type of the argument to the function ***minput_create_ic()*** *(*p. *97).*

- struct **MInputXIMArgIM**

  Structure pointed to by the argument **arg** *of the function* ***minput_open_im()*** *(*p. *96).*

- struct **MInputXIMArgIC**

  Structure pointed to by the argument **arg** *of the function* ***minput_create_ic()*** *(*p. *97).*

Functions

- **MSymbol minput_event_to_key** (**MFrame** ∗frame, void ∗event)

  Convert an event to an input key.

Variables

- **MInputDriver minput_gui_driver**

  Input driver for internal input methods on window systems.

- **MSymbol Mxim**

  Symbol of the name "xim".

- **MInputDriver minput_xim_driver**

  Input method driver for XIM.

2.23.1 Detailed Description

Input method support on window systems.

The input driver `minput_gui_driver` is provided for internal input methods that is useful on window systems. It displays preedit text and status text at the inputting spot. See the documentation of `minput_gui_driver` for more details.

In the m17n-X library, the foreign input method of name `Mxim` is provided. It uses XIM (X Input Method) as a background input engine. The symbol `Mxim` has a property `Minput_driver` whose value is a pointer to the input driver `minput_xim_driver`. See the documentation of `minput_xim_driver` for more details.

2.23.2 Function Documentation

2.23.2.1 MSymbol minput_event_to_key (MFrame ∗ *frame*, void ∗ *event*)

Convert an event to an input key.

The **minput_event_to_key()** (p. 150) function returns the input key corresponding to event **event** on **frame** by a window system dependent manner.

In the m17n-X library, **event** must be a pointer to the structure `XKeyEvent`, and it is handled as below.

At first, the keysym name of **event** is acquired by the function `XKeysymToString`. Then, the name is modified as below.

If the name is one of "a" .. "z" and **event** has a Shift modifier, the name is converted to "A" .. "Z" respectively, and the Shift modifier is cleared.

If the name is one byte length and **event** has a Control modifier, the byte is bitwise anded by 0x1F and the Control modifier is cleared.

If **event** still has modifiers, the name is preceded by "S-" (Shift), "C-" (Control), "M-" (Meta), "A-" (Alt), "s-" (Super), and/or "H-" (Hyper) in this order.

For instance, if the keysym name is "a" and the event has Shift, Meta, and Hyper modifiers, the resulting name is "M-H-A".

At last, a symbol who has the name is returned.

2.23.3 Variable Documentation

2.23.3.1 MInputDriver minput_gui_driver

Input driver for internal input methods on window systems.

The input driver `minput_gui_driver` is for internal input methods to be used on window systems.

It creates sub-windows for a preedit text and a status text, and displays them at the input spot set by the function **minput_set_spot()** (p. 98).

The macro **M17N_INIT()** (p. 7) set the variable `minput_driver` to the pointer to this driver so that all internal input methods use it.

Therefore, unless `minput_driver` is changed from the default, the driver dependent arguments to the functions whose name begin with minput_ must are treated as follows.

The argument **arg** of the function **minput_open_im()** (p. 96) is ignored.

The argument **arg** of the function **minput_create_ic()** (p. 97) must be a pointer to the structure **`MInputGUIArgIC`** `(p. 192)`. See the documentation of **`MInputGUIArgIC`** `(p. 192)` for more details.

If the argument **key** of function **minput_filter()** (p. 97) is `Mnil`, the argument **arg** must be a pointer to the object of type `XEvent`. In that case, **key** is generated from **arg**.

The argument **arg** of the function **minput_lookup()** (p. 98) must be the same one as that of the function **minput_filter()** (p. 97).

2.23.3.2 MSymbol Mxim

Symbol of the name "xim".

The variable Mxim is a symbol of name "xim". It is a name of the input method driver **minput_xim_driver** (p. 151).

2.23.3.3 MInputDriver minput_xim_driver

Initial value:

```
{ xim_open_im, xim_close_im, xim_create_ic, xim_destroy_ic,
    xim_filter, xim_lookup, NULL }
```

Input method driver for XIM.

The driver **minput_xim_driver** (p. 151) is for the foreign input method of name **Mxim** (p. 151). It uses XIM (X Input Methods) as a background input engine.

As the symbol **Mxim** (p. 151) has property **Minput_driver** (p. 107) whose value is a pointer to this driver, the input method of language **Mnil** (p. 17) and name **Mxim** (p. 151) uses this driver.

Therefore, for such input methods, the driver dependent arguments to the functions whose name begin with minput_ must be as follows.

The argument **arg** of the function **minput_open_im()** (p. 96) must be a pointer to the structure **MInputXIMArgIM** (p. 195). See the documentation of **MInputXIMArgIM** (p. 195) for more details.

The argument **arg** of the function **minput_create_ic()** (p. 97) must be a pointer to the structure **MInputXIMArgIC** (p. 194). See the documentation of **MInputXIMArgIC** (p. 194) for more details.

The argument **arg** of the function **minput_filter()** (p. 97) must be a pointer to the structure `XEvent`. The argument **key** is ignored.

The argument **arg** of the function **minput_lookup()** (p. 98) must be the same one as that of the function **minput_filter()** (p. 97). The argument **key** is ignored.

2.24 MISC API

Miscellaneous API.

Modules

- **Error Handling**

  Error handling of the m17n library.

- **Debugging**

  Support for m17n library users to debug their programs.

2.24.1 Detailed Description

Miscellaneous API.

2.25 Error Handling

Error handling of the m17n library.

Enumerations

- 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 }

  Enumeration for error code of the m17n library.

Variables

- int **merror_code**

  External variable to hold error code of the m17n library.

- void($*$ **m17n_memory_full_handler**)(enum **MErrorCode** err)

  Memory allocation error handler.

2.25.1 Detailed Description

Error handling of the m17n library.

There are two types of errors that may happen in a function of the m17n library.

The first type is argument errors. When a library function is called with invalid arguments, it returns a value that indicates error and at the same time sets the external variable **merror_code** (p. 156) to a non-zero integer.

The second type is memory allocation errors. When the required amount of memory is not available on the system, m17n library functions call a function pointed to by the external variable `m17n_memory_full_handler`. The default value of the variable is a pointer to the default_error_handle() function, which just calls `exit()`.

2.25.2 Enumeration Type Documentation

2.25.2.1 enum MErrorCode

Enumeration for error code of the m17n library.

Enumeration for error code of the m17n library.

When a library function is called with an invalid argument, it sets the external variable **merror_code** (p. 156) to one of these values. All the error codes are positive integers.

When a memory allocation error happens, the function pointed to by the external variable **m17n_memory_full_handler** (p. 156) is called with one of these values as an argument.

Enumerator:

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 Variable Documentation

2.25.3.1 int merror_code

External variable to hold error code of the m17n library.

The external variable **merror_code** (p. 156) holds an error code of the m17n library. When a library function is called with an invalid argument, it sets this variable to one of `enum` **MErrorCode** (p. 155).

This variable initially has the value 0.

2.25.3.2 void(∗ m17n_memory_full_handler)(enum MErrorCode err)

Memory allocation error handler.

The external variable **m17n_memory_full_handler** (p. 156) holds a pointer to the function to call when a library function failed to allocate memory. **err** is one of `enum` **MErrorCode** (p. 155) indicating in which function the error occurred.

This variable initially points a function that simply calls the `exit` () function with **err** as an argument.

An application program that needs a different error handling can change this variable to point a proper function.

2.26 Debugging

Support for m17n library users to debug their programs.

Functions

- **MCharTable** ∗ **mdebug_dump_chartab** (**MCharTable** ∗table, int indent)

  Dump a chartable.

- **MFace** ∗ **mdebug_dump_face** (**MFace** ∗face, int indent)

  Dump a face.

- **MFont** ∗ **mdebug_dump_font** (**MFont** ∗font)

  Dump a font.

- **MFontset** ∗ **mdebug_dump_fontset** (**MFontset** ∗fontset, int indent)

  Dump a fontset.

- **MInputMethod** ∗ **mdebug_dump_im** (**MInputMethod** ∗im, int indent)

  Dump an input method.

- int **mdebug_hook** ()

  Hook function called on an error.

- **MText** ∗ **mdebug_dump_mtext** (**MText** ∗mt, int indent, int fullp)

  Dump an M-text.

- **MPlist** ∗ **mdebug_dump_plist** (**MPlist** ∗plist, int indent)

  Dump a property list.

- **MSymbol** **mdebug_dump_symbol** (**MSymbol** symbol, int indent)

  Dump a symbol.

- **MSymbol** **mdebug_dump_all_symbols** (int indent)

  Dump all symbol names.

2.26.1 Detailed Description

Support for m17n library users to debug their programs.

The m17n library provides the following facilities to support the library users to debug their programs.

- Environment variables to control printing of various information.
  - MDEBUG_INIT – If set to 1, print information about the library initialization on the call of **M17N_INIT()** (p. 7).
  - MDEBUG_FINI – If set to 1, print counts of objects that are not yet freed on the call of **M17N_FINI()** (p. 8).
  - MDEBUG_CHARSET – If set to 1, print information about charsets being loaded from the m17n database.

  - MDEBUG_CODING – If set to 1, print information about coding systems being loaded from the m17n database.
  - MDEBUG_DATABASE – If set to 1, print information about data being loaded from the m17n database.
  - MDEBUG_FONT – If set to 1, print information about fonts being selected and opened.
  - MDEBUG_FLT – If set to 1, 2, or 3, print information about which command of Font Layout Table are being executed. The bigger number prints the more detailed information.
  - MDEBUG_INPUT – If set to 1, print information about how an input method is running.
  - MDEBUG_ALL – Setting this variable to 1 is equivalent to setting all the above variables to 1.

- Functions to print various objects in a human readable way. See the documentation of mdebug_dump_XXXX() functions.

- The hook function called on an error. See the documentation of **mdebug_hook()** (p. 159).

2.26.2 Function Documentation

2.26.2.1 MCharTable∗ mdebug_dump_chartab (MCharTable ∗ *table*, int *indent*)

Dump a chartable.

The **mdebug_dump_chartab()** (p. 158) function prints a chartable **table** in a human readable way to the stderr. **indent** specifies how many columns to indent the lines but the first one.

Return value:
This function returns **table**.

2.26.2.2 MFace∗ mdebug_dump_face (MFace ∗ *face*, int *indent*)

Dump a face.

The **mdebug_dump_face()** (p. 158) function prints face **face** in a human readable way to the stderr. **indent** specifies how many columns to indent the lines but the first one.

Return value:
This function returns **face**.

2.26.2.3 MFont∗ mdebug_dump_font (MFont ∗ *font*)

Dump a font.

The **mdebug_dump_font()** (p. 158) function prints font **font** in a human readable way to the stderr.

Return value:
This function returns **font**.

2.26.2.4 MFontset∗ mdebug_dump_fontset (MFontset ∗ *fontset*, int *indent*)

Dump a fontset.

The **mdebug_dump_fontset()** (p. 159) function prints fontset **fontset** in a human readable way to the stderr. **indent** specifies how many columns to indent the lines but the first one.

Return value:
This function returns **fontset**.

2.26.2.5 MInputMethod∗ mdebug_dump_im (MInputMethod ∗ *im*, int *indent*)

Dump an input method.

The **mdebug_dump_im()** (p. 159) function prints the input method **im** in a human readable way to the stderr. **indent** specifies how many columns to indent the lines but the first one.

Return value:
This function returns **im**.

2.26.2.6 int mdebug_hook (void)

Hook function called on an error.

The **mdebug_hook()** (p. 159) function is called when an error happens. It returns -1 without doing anything. It is useful to set a break point on this function in a debugger.

2.26.2.7 MText∗ mdebug_dump_mtext (MText ∗ *mt*, int *indent*, int *fullp*)

Dump an M-text.

The **mdebug_dump_mtext()** (p. 159) function prints the M-text **mt** in a human readable way to the stderr. **indent** specifies how many columns to indent the lines but the first one. If **fullp** is zero, this function prints only a character code sequence. Otherwise, it prints the internal byte sequence and text properties as well.

Return value:
This function returns **mt**.

2.26.2.8 MPlist∗ mdebug_dump_plist (MPlist ∗ *plist*, int *indent*)

Dump a property list.

The **mdebug_dump_plist()** (p. 159) function prints a property list **plist** in a human readable way to the stderr. **indent** specifies how many columns to indent the lines but the first one.

Return value:
This function returns **plist**.

2.26.2.9 MSymbol mdebug_dump_symbol (MSymbol *symbol*, int *indent*)

Dump a symbol.

The **mdebug_dump_symbol()** (p. 160) function prints symbol **symbol** in a human readable way to the stderr. **indent** specifies how many columns to indent the lines but the first one.

Return value:
This function returns **symbol**.

Errors:
MERROR_DEBUG

2.26.2.10 MSymbol mdebug_dump_all_symbols (int *indent*)

Dump all symbol names.

The **mdebug_dump_all_symbols()** (p. 160) function prints names of all symbols to the stderr. **indent** specifies how many columns to indent the lines but the first one.

Return value:
This function returns **Mnil** (p. 17).

Errors:
MERROR_DEBUG

Chapter 3

Data Structure Documentation

3.1 M17NObjectHead Struct Reference

The first member of a managed object.

Data Fields

- void ∗ **filler** [2]

3.1.1 Detailed Description

The first member of a managed object.

When an application program defines a new structure for managed objects, its first member must be of the type `struct` **M17NObjectHead** (p. 161). Its contents are used by the m17n library, and application programs should never touch them.

3.1.2 Field Documentation

3.1.2.1 void∗ M17NObjectHead::filler[2]

Hidden from applications.

3.2 MCodingInfoISO2022 Struct Reference

Structure for a coding system of type **MCODING_TYPE_ISO_2022** (p. 76).

Data Fields

- int **initial_invocation** [2]
- char **designations** [32]
- unsigned **flags**

3.2.1 Detailed Description

Structure for a coding system of type **MCODING_TYPE_ISO_2022** (p. 76).

Structure for extra information about a coding system of type MCODING_TYPE_ISO_2022.

3.2.2 Field Documentation

3.2.2.1 int MCodingInfoISO2022::initial_invocation[2]

Table of numbers of an ISO2022 code extension element invoked to each graphic plane (Graphic Left and Graphic Right). -1 means no code extension element is invoked to that plane.

3.2.2.2 char MCodingInfoISO2022::designations[32]

Table of code extension elements. The Nth element corresponds to the Nth charset in **charset_names**, which is an argument given to the **mconv_define_coding()** (p. 77) function.

If an element value is 0..3, it specifies a graphic register number to designate the corresponds charset. In addition, the charset is initially designated to that graphic register.

If the value is -4..-1, it specifies a graphic register number 0..3 respectively to designate the corresponds charset. Initially, the charset is not designated to any graphic register.

3.2.2.3 unsigned MCodingInfoISO2022::flags

Bitwise OR of `enum MCodingFlagISO2022` .

3.3 MCodingInfoUTF Struct Reference

Structure for extra information about a coding system of type **MCODING_TYPE_UTF** (p. 76).

Data Fields

- int **code_unit_bits**
- int **bom**
- int **endian**

3.3.1 Detailed Description

Structure for extra information about a coding system of type **MCODING_TYPE_UTF** (p. 76).

3.3.2 Field Documentation

3.3.2.1 int MCodingInfoUTF::code_unit_bits

Specify bits of a code unit. The value must be 8, 16, or 32.

3.3.2.2 int MCodingInfoUTF::bom

Specify how to handle the heading BOM (byte order mark). The value must be 0, 1, or 2. The meanings are as follows:

0: On decoding, check the first two byte. If they are BOM, decide endian by them. If not, decide endian by the member `endian`. On encoding, produce byte sequence according to `endian` with heading BOM.

1: On decoding, do not handle the first two bytes as BOM, and decide endian by `endian`. On encoding, produce byte sequence according to `endian` without BOM.

2: On decoding, handle the first two bytes as BOM and decide ending by them. On encoding, produce byte sequence according to `endian` with heading BOM.

If <code_unit_bits> is 8, the value has no meaning.

3.3.2.3 int MCodingInfoUTF::endian

Specify the endian type. The value must be 0 or 1. 0 means little endian, and 1 means big endian.

If <code_unit_bits> is 8, the value has no meaning.

3.4 MConverter Struct Reference

Structure to be used in code conversion.

Data Fields

- 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 Detailed Description

Structure to be used in code conversion.

Structure to be used in code conversion. The first three members are to control the conversion.

3.4.2 Field Documentation

3.4.2.1 int MConverter::lenient

Set the value to nonzero if the conversion should be lenient. By default, the conversion is strict (i.e. not lenient).

If the conversion is strict, the converter stops at the first invalid byte (on decoding) or at the first character not supported by the coding system (on encoding). If this happens, `MConverter->result` is set to `MCONVERSION_RESULT_INVALID_BYTE` or `MCONVERSION_RESULT_INVALID_CHAR` accordingly.

If the conversion is lenient, on decoding, an invalid byte is kept per se, and on encoding, an invalid character is replaced with "<U+XXXX>" (if the character is a Unicode character) or with "<M+XXXXXX>" (otherwise).

3.4.2.2 int MConverter::last_block

Set the value to nonzero before decoding or encoding the last block of the byte sequence or the character sequence respectively. The value influences the conversion as below.

On decoding, in the case that the last few bytes are too short to form a valid byte sequence:

If the value is nonzero, the conversion terminates by error (MCONVERSION_RESULT_INVALID_BYTE) at the first byte of the sequence.

If the value is zero, the conversion terminates successfully. Those bytes are stored in the converter as carryover and are prepended to the byte sequence of the further conversion.

On encoding, in the case that the coding system is context dependent:

If the value is nonzero, the conversion may produce a byte sequence at the end to reset the context to the initial state even if the source characters are zero.

If the value is zero, the conversion never produce such a byte sequence at the end.

3.4.2.3 unsigned MConverter::at_most

If the value is nonzero, it specifies at most how many characters to convert.

3.4.2.4 int MConverter::nchars

The following three members are to report the result of the conversion.

Number of characters most recently decoded or encoded.

3.4.2.5 int MConverter::nbytes

Number of bytes recently decoded or encoded.

3.4.2.6 enum MConversionResult MConverter::result

Result code of the conversion.

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

Various information about the status of code conversion. The contents depend on the type of coding system. It is assured that `status` is aligned so that any type of casting is safe and at least 256 bytes of memory space can be used.

3.4.2.11 void* MConverter::internal_info

This member is for internally use only. An application program should never touch it.

3.5 MDrawControl Struct Reference

Type of a text drawing control.

Data Fields

- 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 Detailed Description

Type of a text drawing control.

The type **MDrawControl** (p. 166) is the structure that controls how to draw an M-text.

3.5.2 Field Documentation

3.5.2.1 unsigned MDrawControl::as_image

If nonzero, draw an M-text as image, i.e. with background filled with background colors of faces put on the M-text. Otherwise, the background is not changed.

3.5.2.2 unsigned MDrawControl::align_head

If nonzero and the first glyph of each line has negative lbearing, shift glyphs horizontally to right so that no pixel is drawn to the left of the specified position.

3.5.2.3 unsigned MDrawControl::two_dimensional

If nonzero, draw an M-text two-dimensionally, i.e., newlines in M-text breaks lines and the following characters are drawn in the next line. If <format> is non-NULL, and the function returns nonzero line width, a line longer than that width is also broken.

3.5.2.4 unsigned MDrawControl::orientation_reversed

If nonzero, draw an M-text to the right of a specified position.

3.5.2.5 unsigned MDrawControl::enable_bidi

If nonzero, reorder glyphs correctly for bidi text.

3.5.2.6 unsigned MDrawControl::ignore_formatting_char

If nonzero, don't draw characters whose general category (in Unicode) is Cf (Other, format).

3.5.2.7 unsigned MDrawControl::fixed_width

If nonzero, draw glyphs suitable for a terminal. Not yet implemented.

3.5.2.8 unsigned MDrawControl::anti_alias

If nonzero, draw glyphs with anti-aliasing if a backend font driver supports it.

3.5.2.9 unsigned MDrawControl::disable_overlapping_adjustment

If nonzero, disable the adjustment of glyph positions to avoid horizontal overlapping at font boundary.

3.5.2.10 unsigned int MDrawControl::min_line_ascent

If nonzero, the values are minimum line ascent pixels.

3.5.2.11 unsigned int MDrawControl::min_line_descent

If nonzero, the values are minimum line descent pixels.

3.5.2.12 unsigned int MDrawControl::max_line_ascent

If nonzero, the values are maximum line ascent pixels.

3.5.2.13 unsigned int MDrawControl::max_line_descent

If nonzero, the values are maximum line descent pixels.

3.5.2.14 unsigned int MDrawControl::max_line_width

If nonzero, the value specifies how many pixels each line can occupy on the display. The value zero means that there is no limit. It is ignored if <format> is non-NULL.

3.5.2.15 unsigned int MDrawControl::tab_width

If nonzero, the value specifies the distance between tab stops in columns (the width of one column is the width of a space in the default font of the frame). The value zero means 8.

3.5.2.16 void(∗ MDrawControl::format)(int line, int y, int ∗indent, int ∗width)

If non-NULL, the value is a function that calculates the indentation and width limit of each line based on the line number LINE and the coordinate Y. The function store the indentation and width limit at the place pointed by INDENT and WIDTH respectively.

The indentation specifies how many pixels the first glyph of each line is shifted to the right (if the member <orientation_reversed> is zero) or to the left (otherwise). If the value is negative, each line is shifted to the reverse direction.

The width limit specifies how many pixels each line can occupy on the display. The value 0 means that there is no limit.

LINE and Y are reset to 0 when a line is broken by a newline character, and incremented each time when a long line is broken because of the width limit.

This has an effect only when <two_dimensional> is nonzero.

3.5.2.17 int(∗ MDrawControl::line_break)(MText ∗mt, int pos, int from, int to, int line, int y)

If non-NULL, the value is a function that calculates a line breaking position when a line is too long to fit within the width limit. POS is the position of the character next to the last one that fits within the limit. FROM is the position of the first character of the line, and TO is the position of the last character displayed on the line if there were not width limit. LINE and Y are the same as the arguments to <format>.

The function must return a character position to break the line.

The function should not modify MT.

The **mdraw_default_line_break()** (p. 149) function is useful for such a script that uses SPACE as a word separator.

3.5.2.18 int MDrawControl::with_cursor

If nonzero, show the cursor according to <cursor_width>.

3.5.2.19 int MDrawControl::cursor_pos

Specifies the character position to display a cursor. If it is greater than the maximum character position, the cursor is displayed next to the last character of an M-text. If the value is negative, even if <cursor_width> is nonzero, cursor is not displayed.

3.5.2.20 int MDrawControl::cursor_width

If nonzero, display a cursor at the character position <cursor_pos>. If the value is positive, it is the pixel width of the cursor. If the value is negative, the cursor width is the same as the underlining glyph(s).

3.5.2.21 int MDrawControl::cursor_bidi

If nonzero and <cursor_width> is also nonzero, display double bar cursors; at the character position <cursor_pos> and at the logically previous character. Both cursors have one pixel width with horizontal fringes at upper or lower positions.

3.5.2.22 int MDrawControl::partial_update

If nonzero, on drawing partial text, pixels of surrounding texts that intrude into the drawing area are also drawn. For instance, some CVC sequence of Thai text (C is consonant, V is upper vowel) is drawn so that V is placed over the middle of two Cs. If this CVC sequence is already drawn and only the last C is drawn again (for instance by updating cursor position), the right half of V is erased if this member is zero. By setting this member to nonzero, even with such a drawing, we can keep this CVC sequence correctly displayed.

3.5.2.23 int MDrawControl::disable_caching

If nonzero, don't cache the result of any drawing information of an M-text.

3.5.2.24 MDrawRegion MDrawControl::clip_region

If non-NULL, limit the drawing effect to the specified region.

3.6 MDrawGlyph Struct Reference

Type of information about a glyph metric and font.

Data Fields

- 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**

    - int **from**
    - int **to**

3.6.1 Detailed Description

Type of information about a glyph metric and font.

The type **MDrawGlyph** (p. 170) is the structure that contains information about a glyph metric and font. It is used by the function **mdraw_glyph_list()** (p. 148).

3.6.2 Field Documentation

3.6.2.1 int MDrawGlyph::from

Character range corresponding to the glyph.

3.6.2.2 int MDrawGlyph::to

3.6.2.3 int MDrawGlyph::glyph_code

Font glyph code of the glyph.

3.6.2.4 int MDrawGlyph::x_advance

Logical width of the glyph. Nominal distance to the next glyph.

3.6.2.5 int MDrawGlyph::y_advance

Logical height of the glyph. Nominal distance to the next glyph.

3.6.2.6 int MDrawGlyph::x_off

X offset relative to the glyph position.

3.6.2.7 int MDrawGlyph::y_off

Y offset relative to the glyph position.

3.6.2.8 int MDrawGlyph::lbearing

Metric of the glyph (left-bearing).

3.6.2.9 int MDrawGlyph::rbearing

Metric of the glyph (right-bearing).

3.6.2.10 int MDrawGlyph::ascent

Metric of the glyph (ascent).

3.6.2.11 int MDrawGlyph::descent

Metric of the glyph (descent).

3.6.2.12 MFont∗ MDrawGlyph::font

Font used for the glyph. Set to NULL if no font is found for the glyph.

3.6.2.13 MSymbol MDrawGlyph::font_type

Type of the font. One of Mx, Mfreetype, Mxft.

3.6.2.14 void∗ MDrawGlyph::fontp

Pointer to the font structure. The actual type is (XFontStruct ∗) if <font_type> member is Mx, FT_Face if <font_type> member is Mfreetype, and (XftFont ∗) if <font_type> member is Mxft.

3.7 MDrawGlyphInfo Struct Reference

Type of information about a glyph.

Data Fields

- 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 Detailed Description

Type of information about a glyph.

The type **MDrawGlyphInfo** (p. 172) is the structure that contains information about a glyph. It is used by **mdraw_glyph_info()** (p. 148).

3.7.2 Field Documentation

3.7.2.1 int MDrawGlyphInfo::from

Start position of character range corresponding to the glyph.

3.7.2.2 int MDrawGlyphInfo::to

End position of character range corresponding to the glyph.

3.7.2.3 int MDrawGlyphInfo::line_from

Start position of character range corresponding to the line of the glyph.

3.7.2.4 int MDrawGlyphInfo::line_to

End position of character range corresponding to the line of the glyph.

3.7.2.5 int MDrawGlyphInfo::x

X coordinates of the glyph.

3.7.2.6 int MDrawGlyphInfo::y

Y coordinates of the glyph.

3.7.2.7 MDrawMetric MDrawGlyphInfo::metrics

Metric of the glyph.

3.7.2.8 MFont* MDrawGlyphInfo::font

Font used for the glyph. Set to NULL if no font is found for the glyph.

3.7.2.9 int MDrawGlyphInfo::prev_from

Character ranges corresponding to logically previous glyphs. Note that we do not need the members prev_to because it must be the same as the member <from>.

3.7.2.10 int MDrawGlyphInfo::next_to

Character ranges corresponding to logically next glyphs. Note that we do not need the members next_from because it must be the same as the member <to> respectively.

3.7.2.11 int MDrawGlyphInfo::left_from

Start position of character ranges corresponding to visually left glyphs.

3.7.2.12 int MDrawGlyphInfo::left_to

End position of character ranges corresponding to visually left glyphs.

3.7.2.13 int MDrawGlyphInfo::right_from

Start position of character ranges corresponding to visually right glyphs.

3.7.2.14 int MDrawGlyphInfo::right_to

End position of character ranges corresponding to visually left glyphs.

3.7.2.15 int MDrawGlyphInfo::logical_width

Logical width of the glyph. Nominal distance to the next glyph.

3.8 MDrawMetric Struct Reference

Type of metric for glyphs and texts.

Data Fields

- int **x**
- int **y**
- unsigned int **width**
- unsigned int **height**

3.8.1 Detailed Description

Type of metric for glyphs and texts.

The type **MDrawMetric** (p. 174) is for a metric of a glyph and a drawn text. It is also used to represent a rectangle area of a graphic device.

3.8.2 Field Documentation

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 MDrawTextItem Struct Reference

Type of textitems.

Data Fields

- **MText** ∗ **mt**
- int **delta**
- **MFace** ∗ **face**
- **MDrawControl** ∗ **control**

3.9.1 Detailed Description

Type of textitems.

The type **MDrawTextItem** (p. 175) is for *textitem* objects. Each textitem contains an M-text and some other information to control the drawing of the M-text.

3.9.2 Field Documentation

3.9.2.1 MText∗ MDrawTextItem::mt

M-text.

3.9.2.2 int MDrawTextItem::delta

Optional change in the position (in the unit of pixel) along the X-axis before the M-text is drawn.

3.9.2.3 MFace∗ MDrawTextItem::face

Pointer to a face object. Each property of the face, if not Mnil, overrides the same property of face(s) specified as a text property in <mt>.

3.9.2.4 MDrawControl∗ MDrawTextItem::control

Pointer to a draw control object. The M-text <mt> is drawn by **mdraw_text_with_control()** (p. 146) with this control object.

3.10 MFaceBoxProp Struct Reference

Type of box spec of face.

Data Fields

- 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 Detailed Description

Type of box spec of face.

The type **MFaceBoxProp** (p. 176) is to specify the detail of **Mbox** (p. 138) property of a face. The value of the property must be a pointer to an object of this type.

3.10.2 Field Documentation

3.10.2.1 unsigned MFaceBoxProp::width

Width of the box line in pixels.

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 MFaceHLineProp Struct Reference

Type of horizontal line spec of face.

Public Types

- enum **MFaceHLineType** {
  MFACE_HLINE_BOTTOM,
  MFACE_HLINE_UNDER,
  MFACE_HLINE_STRIKE_THROUGH,
  MFACE_HLINE_OVER,
  MFACE_HLINE_TOP }

Data Fields

- enum **MFaceHLineProp::MFaceHLineType type**
- unsigned **width**
- **MSymbol color**

3.11.1 Detailed Description

Type of horizontal line spec of face.

The type **MFaceHLineProp** (p. 177) is to specify the detail of **Mhline** (p. 137) property of a face. The value of the property must be a pointer to an object of this type.

3.11.2 Member Enumeration Documentation

3.11.2.1 enum MFaceHLineProp::MFaceHLineType

Type of the horizontal line.

Enumerator:

MFACE_HLINE_BOTTOM

MFACE_HLINE_UNDER

MFACE_HLINE_STRIKE_THROUGH

MFACE_HLINE_OVER

MFACE_HLINE_TOP

3.11.3 Field Documentation

3.11.3.1 enum MFaceHLineProp::MFaceHLineType MFaceHLineProp::type

Type of the horizontal line.

3.11.3.2 unsigned MFaceHLineProp::width

Width of the line in pixels.

3.11.3.3 MSymbol MFaceHLineProp::color

Color of the line. If the value is Mnil, foreground color of a merged face is used.

3.12 MFLTFont Struct Reference

Type of font to be used by the FLT driver.

Data Fields

- **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 Detailed Description

Type of font to be used by the FLT driver.

The type **MFLTFont** (p. 179) is the structure that contains information about a font used by the FLT driver.

3.12.2 Field Documentation

3.12.2.1 MSymbol MFLTFont::family

Family name of the font. It may be **Mnil** (p. 17) if the family name is not important in finding a Font Layout Table suitable for the font (for instance, in the case that the font is an OpenType font).

3.12.2.2 int MFLTFont::x_ppem

Horizontal font sizes in pixels per EM.

3.12.2.3 int MFLTFont::y_ppem

Vertical font sizes in pixels per EM.

3.12.2.4 int(∗ MFLTFont::get_glyph_id)(struct _MFLTFont ∗font, MFLTGlyphString ∗gstring, int from, int to)

Callback function to get glyph IDs for glyphs between FROM (inclusive) and TO (exclusive) of GSTRING. If the member <encoded> of a glyph is zero, the member <code> of that glyph is a character code. The function must convert it to the glyph ID of FONT.

3.12.2.5 int(∗ MFLTFont::get_metrics)(struct _MFLTFont ∗font, MFLTGlyphString ∗gstring, int from, int to)

Callback function to get metrics of glyphs between FROM (inclusive) and TO (exclusive) of GSTRING. If the member <measured> of a glyph is zero, the function must set the members <xadv>, <yadv>, <ascent>, <descent>, <lbearing>, and <rbearing> of the glyph.

3.12.2.6 int(∗ MFLTFont::check_otf)(struct _MFLTFont ∗font, MFLTOtfSpec ∗spec)

Callback function to check if the font has OpenType GSUB/GPOS features for a specific script/language. The function must return 1, if the font satisfies SPEC, or 0. It must be NULL if the font does not have OpenType tables.

3.12.2.7 int(∗ MFLTFont::drive_otf)(struct _MFLTFont ∗font, MFLTOtfSpec ∗spec, MFLTGlyphString ∗in, int from, int to, MFLTGlyphString ∗out, MFLTGlyphAdjustment ∗adjustment)

Callback function to apply OpenType features in SPEC to glyphs between FROM (inclusive) and TO (exclusive) of IN. The resulting glyphs are appended to the tail of OUT. If OUT does not have a room to store all the resulting glyphs, it must return -2. It must be NULL if the font does not have OpenType tables.

3.12.2.8 void∗ MFLTFont::internal

For m17n-lib's internal use only. It should be initialized to NULL.

3.13 MFLTGlyph Struct Reference

Type of information about a glyph.

Data Fields

- int **c**
- unsigned int **code**
- int **from**
- int **to**
- int **xadv**
- int **yadv**
- unsigned **encoded**: 1
- unsigned **measured**: 1
- unsigned **adjusted**: 1

  - int **ascent**
  - int **descent**
  - int **lbearing**
  - int **rbearing**

  - int **xoff**
  - int **yoff**

3.13.1 Detailed Description

Type of information about a glyph.

The type **MFLTGlyph** (p. 181) is the structure that contains information about a glyph.

3.13.2 Field Documentation

3.13.2.1 int MFLTGlyph::c

Character code (Unicode) of the glyph. This is the sole member to be set before calling the functions **mflt_find()** (p. 109) and **mflt_run()** (p. 110).

3.13.2.2 unsigned int MFLTGlyph::code

Glyph ID of the glyph in the font.

3.13.2.3 int MFLTGlyph::from

Starting index of the run in **MFLTGlyphString** (p. 184) that is replaced by this glyph.

3.13.2.4 int MFLTGlyph::to

Ending index of the run in **MFLTGlyphString** (p. 184) that is replaced by this glyph.

3.13.2.5 int MFLTGlyph::xadv

Advance width for horizontal layout expressed in 26.6 fractional pixel format.

3.13.2.6 int MFLTGlyph::yadv

Advance height for vertical layout expressed in 26.6 fractional pixel format.

3.13.2.7 int MFLTGlyph::ascent

Ink metrics of the glyph expressed in 26.6 fractional pixel format.

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

Horizontal and vertical adjustments for the glyph positioning expressed in 26.6 fractional pixel format.

3.13.2.12 int MFLTGlyph::yoff

3.13.2.13 unsigned MFLTGlyph::encoded

Flag to tell whether the member <code> has already been set to a glyph ID in the font.

3.13.2.14 unsigned MFLTGlyph::measured

Flag to tell if the metrics of the glyph (members <xadv> thru <rbearing>) are already calculated.

3.13.2.15 unsigned MFLTGlyph::adjusted

Flag to tell if the metrics of the glyph is adjusted, i.e. <xadv> or <yadv> is different from the normal size, or <xoff> or <yoff> is nonzero.

3.14 MFLTGlyphAdjustment Struct Reference

Type of information about a glyph position adjustment.

Data Fields

- short **back**
- unsigned **advance_is_absolute**: 1
- unsigned **set**: 1

  - int **xadv**
  - int **yadv**

  - int **xoff**
  - int **yoff**

3.14.1 Detailed Description

Type of information about a glyph position adjustment.

The type **MFLTGlyphAdjustment** (p. 183) is the structure to store information about a glyph metrics/position adjustment. It is given to the callback function **drive_otf** of **MFLTFont** (p. 179).

3.14.2 Field Documentation

3.14.2.1 int MFLTGlyphAdjustment::xadv

Adjustments for advance width for horizontal layout and advance height for vertical layout expressed in 26.6 fractional pixel format.

3.14.2.2 int MFLTGlyphAdjustment::yadv

3.14.2.3 int MFLTGlyphAdjustment::xoff

Horizontal and vertical adjustments for glyph positioning expressed in 26.6 fractional pixel format.

3.14.2.4 int MFLTGlyphAdjustment::yoff

3.14.2.5 short MFLTGlyphAdjustment::back

Number of glyphs to go back for drawing a glyph.

3.14.2.6 unsigned MFLTGlyphAdjustment::advance_is_absolute

If nonzero, the member <xadv> and <yadv> are absolute, i.e., they should not be added to a glyph's origianl advance width and height.

3.14.2.7 unsigned MFLTGlyphAdjustment::set

Should be set to 1 iff at least one of the other members has a nonzero value.

3.15 MFLTGlyphString Struct Reference

Type of information about a glyph sequence.

Data Fields

- int **glyph_size**
- **MFLTGlyph** $*$ **glyphs**
- int **allocated**
- int **used**
- unsigned int **r2l**

3.15.1 Detailed Description

Type of information about a glyph sequence.

The type **MFLTGlyphString** (p. 184) is the structure that contains information about a sequence of glyphs.

3.15.2 Field Documentation

3.15.2.1 int MFLTGlyphString::glyph_size

The actual byte size of elements of the array pointed by the member **glyphs** (p. 184). It must be equal to or greater than "sizeof (**MFLTGlyph** (p. 181))".

3.15.2.2 MFLTGlyph$*$ MFLTGlyphString::glyphs

Array of glyphs.

3.15.2.3 int MFLTGlyphString::allocated

Number of elements allocated in **glyphs** (p. 184).

3.15.2.4 int MFLTGlyphString::used

Number of elements in **glyphs** (p. 184) in use.

3.15.2.5 unsigned int MFLTGlyphString::r2l

Flag to tell if the glyphs should be drawn from right-to-left or not.

3.16 MFLTOtfSpec Struct Reference

Type of specification of GSUB and GPOS OpenType tables.

Data Fields

- **MSymbol sym**
- unsigned int ∗ **features** [2]

  - unsigned int **script**
  - unsigned int **langsys**

3.16.1 Detailed Description

Type of specification of GSUB and GPOS OpenType tables.

The type **MFLTOtfSpec** (p. 185) is the structure that contains information about the GSUB and GPOS features of a specific script and language system to be applied to a glyph sequence.

3.16.2 Field Documentation

3.16.2.1 MSymbol MFLTOtfSpec::sym

Unique symbol representing the spec. This is the same as the **OTF-SPEC** (p. 215) of the FLT.

3.16.2.2 unsigned int MFLTOtfSpec::script

Tags for script and language system.

3.16.2.3 unsigned int MFLTOtfSpec::langsys

3.16.2.4 unsigned int∗ MFLTOtfSpec::features[2]

Array of GSUB (1st element) and GPOS (2nd element) features. Each array is terminated by 0. If an element is 0xFFFFFFFF apply the previous features in that order, and apply all the other features except those that appear in the following elements. It may be NULL if there are no features.

3.17 MInputContext Struct Reference

Structure of input context.

Data Fields

- **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_show**
- int **candidates_changed**
- **MPlist** ∗ **plist**

  - int **candidate_from**
  - int **candidate_to**

3.17.1 Detailed Description

Structure of input context.

The type **`MInputContext`** `(p. 186)` is the structure of input context objects.

3.17.2 Field Documentation

3.17.2.1 MInputMethod∗ MInputContext::im

Backward pointer to the input method. It is set up be the function **minput_create_ic()** (p. 97).

3.17.2.2 MText∗ MInputContext::produced

M-text produced by the input method. It is set up by the function **minput_lookup()** (p. 98) .

3.17.2.3 void∗ MInputContext::arg

Argument given to the function minput_create_im().

3.17.2.4 int MInputContext::active

Flag telling whether the input context is currently active or inactive. The value is set to 1 (active) when the input context is created. It is toggled by the function **minput_toggle()** (p. 98).

3.17.2.5 int MInputContext::x

X and Y coordinate of the spot.

3.17.2.6 int MInputContext::y

3.17.2.7 int MInputContext::ascent

Ascent and descent pixels of the line of the spot.

3.17.2.8 int MInputContext::descent

3.17.2.9 int MInputContext::fontsize

Font size for preedit text in 1/10 point.

3.17.2.10 MText∗ MInputContext::mt

M-text at the spot, or NULL.

3.17.2.11 int MInputContext::pos

Character position in <mt> at the spot.

3.17.2.12 struct { ... } MInputContext::spot

Spot location and size of the input context.

3.17.2.13 void∗ MInputContext::info

The usage of the following members depends on the input method driver. The descriptions below are for the driver of an internal input method. They are set by the function <im>->driver.filter(). Pointer to extra information that <im>->driver.create_ic() setups. It is used to record the internal state of the input context.

3.17.2.14 MText∗ MInputContext::status

M-text describing the current status of the input context.

3.17.2.15 int MInputContext::status_changed

The function <im>->driver.filter() sets the value to 1 when it changes <status>.

3.17.2.16 MText∗ MInputContext::preedit

M-text containing the current preedit text. The function <im>->driver.filter() sets the value.

3.17.2.17 int MInputContext::preedit_changed

The function <im>->driver.filter() sets the value to 1 when it changes <preedit>.

3.17.2.18 int MInputContext::cursor_pos

Cursor position of <preedit>.

3.17.2.19 int MInputContext::cursor_pos_changed

The function <im>->driver.filter() sets the value to 1 when it changes <cursor_pos>.

3.17.2.20 MPlist∗ MInputContext::candidate_list

Plist of the current candidate groups. Each element is an M-text or a plist. If an element is an M-text (i.e. the key is Mtext), candidates in that group are characters in the M-text. If it is a plist (i.e. the key is Mplist), each element is an M-text, and candidates in that group are those M-texts.

3.17.2.21 int MInputContext::candidate_index

Index number of the currently selected candidate in all the candidates. The index of the first candidate is 0. If the number is 8, and the first candidate group contains 7 candidates, the currently selected candidate is the second element of the second candidate group.

3.17.2.22 int MInputContext::candidate_from

Start and the end positions of the preedit text where <candidate_list> corresponds to.

3.17.2.23 int MInputContext::candidate_to

3.17.2.24 int MInputContext::candidate_show

Flag telling whether the current candidate group must be shown or not. The function <im>->driver.filter() sets the value to 1 when an input method required to show candidates, and sets the value to 0 otherwise.

3.17.2.25 int MInputContext::candidates_changed

The function <im>->driver.filter() sets the value to bitwise OR of `enum MInputCandidatesChanged` when it changed any of the above members (<candidate_XXX>), and sets the value to 0 otherwise.

3.17.2.26 MPlist∗ MInputContext::plist

Plist that can be freely used by <im>->driver functions. The driver of internal input method uses it to exchange extra arguments and result for callback functions. The function <im>->driver.create_ic() sets this to an empty plist, and the function <im>->driver.destroy_ic() frees it by using **m17n_object_unref()** (p. 12).

3.18 MInputDriver Struct Reference

Structure of input method driver.

Data Fields

- int(∗ **open_im**)(**MInputMethod** ∗im)

  Open an input method.

- void(∗ **close_im**)(**MInputMethod** ∗im)

  Close an input method.

- int(∗ **create_ic**)(**MInputContext** ∗ic)

  Create an input context.

- void(∗ **destroy_ic**)(**MInputContext** ∗ic)

  Destroy an input context.

- int(∗ **filter**)(**MInputContext** ∗ic, **MSymbol** key, void ∗arg)

  Filter an input key.

- int(∗ **lookup**)(**MInputContext** ∗ic, **MSymbol** key, void ∗arg, **MText** ∗mt)

  Lookup a produced text in an input context.

- **MPlist** ∗ **callback_list**

  List of callback functions.

3.18.1 Detailed Description

Structure of input method driver.

The type **`MInputDriver`** `(p. 190)` is the structure of an input method driver that contains several functions to handle an input method.

3.18.2 Field Documentation

3.18.2.1 int(∗ MInputDriver::open_im)(MInputMethod ∗im)

Open an input method.

This function opens the input method **im**. It is called from the function **minput_open_im()** (p. 96) after all member of **im** but <info> set. If opening **im** succeeds, it returns 0. Otherwise, it returns -1. The function can setup **im->info** to keep various information that is referred by the other driver functions.

3.18.2.2 void(∗ MInputDriver::close_im)(MInputMethod ∗im)

Close an input method.

This function closes the input method **im**. It is called from the function **minput_close_im()** (p. 97). It frees all memory allocated for **im->info** (if any) after finishing all the tasks of closing the input method. But, the other members of **im** should not be touched.

3.18.2.3 int(∗ MInputDriver::create_ic)(MInputContext ∗ic)

Create an input context.

This function creates the input context **ic**. It is called from the function **minput_create_ic()** (p. 97) after all members of **ic** but <info> are set. If creating **ic** succeeds, it returns 0. Otherwise, it returns -1. The function can setup **ic->info** to keep various information that is referred by the other driver functions.

3.18.2.4 void(∗ MInputDriver::destroy_ic)(MInputContext ∗ic)

Destroy an input context.

This function is called from the function **minput_destroy_ic()** (p. 97) and destroys the input context **ic**. It frees all memory allocated for **ic->info** (if any) after finishing all the tasks of destroying the input method. But, the other members of **ic** should not be touched.

3.18.2.5 int(∗ MInputDriver::filter)(MInputContext ∗ic, MSymbol key, void ∗arg)

Filter an input key.

This function is called from the function **minput_filter()** (p. 97) and filters an input key. **key** and **arg** are the same as what given to **minput_filter()** (p. 97).

The task of the function is to handle **key**, update the internal state of **ic**. If **key** is absorbed by the input method and no text is produced, it returns 1. Otherwise, it returns 0.

It may update **ic->status**, **ic->preedit**, **ic->cursor_pos**, **ic->ncandidates**, **ic->candidates**, and **ic->produced** if that is necessary for the member <callback>.

The meaning of **arg** depends on the input method river. See the documentation of `minput_default_driver` and `minput_gui_driver` for instance.

3.18.2.6 int(∗ MInputDriver::lookup)(MInputContext ∗ic, MSymbol key, void ∗arg, MText ∗mt)

Lookup a produced text in an input context.

It is called from the function **minput_lookup()** (p. 98) and looks up a produced text in the input context **ic**. This function concatenate a text produced by the input key **key** (if any) to M-text **mt**. If **key** was correctly handled by the input method of **ic**, it returns 0. Otherwise, it returns 1.

The meaning of **arg** depends on the input method driver. See the documentation of `minput_default_driver` and `minput_gui_driver` for instance.

3.18.2.7 MPlist∗ MInputDriver::callback_list

List of callback functions.

List of callback functions. Keys are one of **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**. Values are functions of type **MInputCallbackFunc** (p. 96).

3.19 MInputGUIArgIC Struct Reference

Type of the argument to the function **minput_create_ic()** (p. 97).

Data Fields

- **MFrame ∗ frame**
- **MDrawWindow client**
- **MDrawWindow focus**

3.19.1 Detailed Description

Type of the argument to the function **minput_create_ic()** (p. 97).

The type **MInputGUIArgIC** (p. 192) is for the argument **arg** of the function **minput_create_ic()** (p. 97) to create an input context of an internal input method.

3.19.2 Field Documentation

3.19.2.1 MFrame∗ MInputGUIArgIC::frame

Frame of the client.

3.19.2.2 MDrawWindow MInputGUIArgIC::client

Window on which to display the preedit and status text.

3.19.2.3 MDrawWindow MInputGUIArgIC::focus

Window that the input context has a focus on.

3.20 MInputMethod Struct Reference

Structure of input method.

Data Fields

- **MSymbol language**
- **MSymbol name**
- **MInputDriver driver**
- void ∗ **arg**
- void ∗ **info**

3.20.1 Detailed Description

Structure of input method.

The type **`MInputMethod`** `(p. 193)` is the structure of input method objects.

3.20.2 Field Documentation

3.20.2.1 MSymbol MInputMethod::language

Which language this input method is for. The value is `Mnil` if the input method is foreign.

3.20.2.2 MSymbol MInputMethod::name

Name of the input method. If the input method is foreign, it must has a property of key `Minput_driver` and the value must be a pointer to a proper input method driver.

3.20.2.3 MInputDriver MInputMethod::driver

Input method driver of the input method.

3.20.2.4 void∗ MInputMethod::arg

The argument given to **minput_open_im()** (p. 96).

3.20.2.5 void∗ MInputMethod::info

Pointer to extra information that <driver>.open_im() setups.

3.21 MInputXIMArgIC Struct Reference

Structure pointed to by the argument **arg** of the function **minput_create_ic()** (p. 97).

Data Fields

- XIMStyle **input_style**
- Window **client_win**
- Window **focus_win**
- XVaNestedList **preedit_attrs**
- XVaNestedList **status_attrs**

3.21.1 Detailed Description

Structure pointed to by the argument **arg** of the function **minput_create_ic()** (p. 97).

The type **MInputXIMArgIC** (p. 194) is the structure pointed to by the argument **arg** of the function **minput_create_ic()** (p. 97) for the foreign input method of name **Mxim** (p. 151).

3.21.2 Field Documentation

3.21.2.1 XIMStyle MInputXIMArgIC::input_style

Used as the arguments of `XCreateIC` following `XNInputStyle`. If this is zero, (`XIMPreeditNothing` | `XIMStatusNothing`) is used, and <preedit_attrs> and <status_attrs> are set to `NULL`.

3.21.2.2 Window MInputXIMArgIC::client_win

Used as the argument of `XCreateIC` following `XNClientWindow`.

3.21.2.3 Window MInputXIMArgIC::focus_win

Used as the argument of `XCreateIC` following `XNFocusWindow`.

3.21.2.4 XVaNestedList MInputXIMArgIC::preedit_attrs

If non- `NULL`, used as the argument of `XCreateIC` following `XNPreeditAttributes`.

3.21.2.5 XVaNestedList MInputXIMArgIC::status_attrs

If non- `NULL`, used as the argument of `XCreateIC` following `XNStatusAttributes`.

3.22 MInputXIMArgIM Struct Reference

Structure pointed to by the argument **arg** of the function **minput_open_im()** (p. 96).

Data Fields

- Display ∗ **display**
- XrmDatabase **db**
- char ∗ **res_class**
- char ∗ **res_name**
- char ∗ **locale**
- char ∗ **modifier_list**

3.22.1 Detailed Description

Structure pointed to by the argument **arg** of the function **minput_open_im()** (p. 96).

The type **MInputXIMArgIM** (p. 195) is the structure pointed to by the argument **arg** of the function **minput_open_im()** (p. 96) for the foreign input method of name **Mxim** (p. 151).

3.22.2 Field Documentation

3.22.2.1 Display∗ MInputXIMArgIM::display

The meaning of the following four members are the same as arguments to XOpenIM(). Display of the client.

3.22.2.2 XrmDatabase MInputXIMArgIM::db

Pointer to the X resource database.

3.22.2.3 char∗ MInputXIMArgIM::res_class

Full class name of the application.

3.22.2.4 char∗ MInputXIMArgIM::res_name

Full resource name of the application.

3.22.2.5 char∗ MInputXIMArgIM::locale

Locale name under which an XIM is opened.

3.22.2.6 char∗ MInputXIMArgIM::modifier_list

Arguments to XSetLocaleModifiers().

Appendix A

Print compile/link options of the m17n library

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

Print information about the m17n database

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

Sample Programs

This section describes these example programs. They are to demonstrate the usage of the m17n library, not for practical use.

- **m17n-conv** (p. 202) – convert file code
- **m17n-view** (p. 203) – view file
- **m17n-date** (p. 203) – display date and time
- **m17n-dump** (p. 203) – dump text image
- **m17n-edit** (p. 205) – edit multilingual text
- **mimx-anthy** (p. 205) – external module for the input method <ja, anthy>
- **mimx-ispell** (p. 206) – external module for the input method <en, ispell>

C.1 m17n-conv – convert file code

C.1.1 SYNOPSIS

m17n-conv [OPTION ...] [INFILE [OUTFILE]]

C.1.2 DESCRIPTION

Convert encoding of given files from one to another.

If INFILE is omitted, the input is taken from standard input. If OUTFILE is omitted, the output written to standard output.

The following OPTIONs are available.

- -f FROMCODE

  FROMCODE is the encoding of INFILE (defaults to UTF-8).
- -t TOCODE

  TOCODE is the encoding of OUTFILE (defaults to UTF-8).
- -k

  Do not stop conversion on error.
- -s

  Suppress warnings.
- -v

  Print progress information.
- -l

  List available encodings.
- –version

  Print version number.
- -h, –help

  Print this message.

C.2 m17n-view – view file

C.2.1 SYNOPSIS

m17n-view [XT-OPTION ...] [OPTION ...] [FILE]

C.2.2 DESCRIPTION

Display FILE on a window.

If FILE is omitted, the input is taken from standard input.

XT-OPTIONs are standard Xt arguments (e.g. -fn, -fg).

The following OPTIONs are available.

- -e ENCODING

  ENCODING is the encoding of FILE (defaults to UTF-8).
- -s FONTSIZE

  FONTSIZE is the fontsize in point. If omitted, it defaults to the size of the default font defined in X resource.
- –version

  Print version number.
- -h, –help

  Print this message.

C.3 m17n-date – display date and time

C.3.1 SYNOPSIS

m17n-date [OPTION ...]

C.3.2 DESCRIPTION

Display the system date and time in many locales on a window.

The following OPTIONs are available.

- –version

  Print version number.
- -h, –help

  Print this message.

C.4 m17n-dump – dump text image

C.4.1 SYNOPSIS

m17n-dump [OPTION ...] [FILE]

C.4.2 DESCRIPTION

Dump a text as PNG image file.

The PNG file is written to a file created in the current directory with the name "BASE.png" where BASE is the basename of FILE. If FILE is omitted, text is read from standard input, and the image is dumped into the file "output.png".

The following OPTIONs are available.

- -s SIZE

  SIZE is the font size in point. The default font size is 12 point.

- -d DPI

  DPI is the resolution in dots per inch. The default resolution is 300 dpi.

- -p PAPER

  PAPER is the paper size: a4, a4r, a5, a5r, b5, b5r, letter, WxH, or W. In the case of WxH, W and H are the width and height in millimeter. In the case of W, W is the width in millimeter. If this option is specified, PAPER limits the image size. If FILE is too large for a single page, multiple files with the names "BASE.01.png", "BASE.02.png", etc. are created.

- -m MARGIN

  MARGIN is the horizontal and vertical margin in millimeter. The default margin is 20 mm. It is ignored when PAPER is not specified.

- -c POS

  POS is the character position of cursor to draw. By default, cursor is not drawn.

- -x

  FILE is assumed to be an XML file generated by the serialize facility of the m17n library, and FILE is deserialized before an image is created.

- -w

  Each line is broken at word boundary.

- -f FILTER

  FILTER is a string containing a shell command line. If this option is specified, the PNG image is not written info a file but is given to FILTER as standard input. If FILTER contains "%s", that part is replaced by a basename of FILE. So, the default behaviour is the same as specifying "cat > s.png" as FILTER.

  If FILTER is just "-", the PNG image is written to stdout.

- -a

  Enable anti-alias drawing.

- –family FAMILY

  Prefer a font whose family name is FAMILY.

- –language LANG

  Prefer a font specified for the language LANG. LANG must be a 2-letter code of ISO 630 (e.g. "en" for English).

- -fg FOREGROUND

  Specify the text color. The supported color names are those of HTML 4.0 and "#RRGGBB" notation.

- -bg BACKGROUND

  Specify the background color. The supported color names are the same as FOREGROUND, except that if "transparent" is specified, make the background transparent.

- -r

  Specify that the orientation of the text is right-to-left.

- -q

  Quiet mode. Don’t print any messages.

- –version

  Print the version number.

- -h, –help

  Print this message.

C.5 m17n-edit – edit multilingual text

C.5.1 SYNOPSIS

m17n-edit [XT-OPTION ...] [OPTION ...] FILE

C.5.2 DESCRIPTION

Display FILE on a window and allow users to edit it.

XT-OPTIONs are standard Xt arguments (e.g. -fn, -fg).

The following OPTIONs are available.

- –version

  Print version number.

- -h, –help

  Print this message.

This program is to demonstrate how to use the m17n GUI API. Although m17n-edit directly uses the GUI API, the API is mainly for toolkit libraries or to implement XOM (X Output Method), not for direct use from application programs.

C.6 mimx-anthy – external module for the input method <ja, anthy>

C.6.1 DESCRIPTION

The shared library mimx-anthy.so is an external module used by the input method <ja, anthy>. It exports these functions.

- init

  Initialize this module.

- fini

  Finalize this module.
- convert

  Convert the current preedit text (Hiragana sequence) into Kana-Kanji mixed text.
- change

  Record the change of candidate of the current segment.
- resize

  Enlarge or shorten the length of the current segment.
- commit

  Commit the lastly selected candidates of all the segments.

C.6.2 See also

Input Method (p. 219)

C.7 mimx-ispell – external module for the input method <en, ispell>

C.7.1 DESCRIPTION

The shared library mimx-ispell.so is an external module used by the input method <en, ispell>. It exports these functions.

- init

  Initialize this library.
- fini

  Finalize this library.
- ispell_word

  Check the spell of the current preedit text (English) and, if the spell is incorrect, return a list of candidates.

This program is just for demonstrating how to write an external module for an m17n input method, not for an actual use.

C.7.2 See also

Input Method (p. 219)

Appendix D

Data format of the m17n database

This section describes formats of these data supplied by the m17n database.

- **General** (p. 208) – General Format
- **CharsetList** (p. 210) – List of character set definitions
- **CodingList** (p. 210) – List of coding system definitions
- **Dir** (p. 211) – List of data in a database directory.
- **FLT** (p. 211) – Font Layout Table
- **FontEncoding** (p. 217) – Font Encoding
- **FontSize** (p. 218) – Font Size
- **Fontset** (p. 218) – Fontset
- **IM** (p. 219) – Input Method

D.1 General Format

D.1.1 DESCRIPTION

The **mdatabase_load()** (p. 62) function returns the data specified by tags in the form of plist if the first tag is not `Mchartable` nor `Mcharset`. The keys of the returned plist are limited to `Minteger`, `Msymbol`, `Mtext`, and `Mplist`. The type of the value is unambiguously determined by the corresponding key. If the key is `Minteger`, the value is an integer. If the key is `Msymbol`, the value is a symbol. And so on.

A number of expressions are possible to represent a plist. For instance, we can use the form `(K1:V1, K2:V2, ..., Kn:Vn)` to represent a plist whose first property key and value are K1 and V1, second key and value are K2 and V2, and so on. However, we can use a simpler expression here because the types of plists used in the m17n database are fairly restricted.

Hereafter, we use an expression, which is similar to S-expression, to represent a plist. (Actually, the default database loader of the m17n library is designed to read data files written in this expression.)

The expression consists of one or more *elements*. Each element represents a property, i.e. a single element of a plist.

Elements are separated by one or more *whitespaces*, i.e. a space (code 32), a tab (code 9), or a newline (code 10). Comments begin with a semicolon (`;`) and extend to the end of the line.

The key and the value of each property are determined based on the type of the element as explained below.

- INTEGER

  An element that matches the regular expression `-?[0-9]+` or `0[xX][0-9A-Fa-f]+` represents a property whose key is `Minteger`. An element matching the former expression is interpreted as an integer in decimal notation, and one matching the latter is interpreted as an integer in hexadecimal notation. The value of the property is the result of interpretation.

  For instance, the element `0xA0` represents a property whose value is 160 in decimal.

- SYMBOL

  An element that matches the regular expression `[^-0-9(]([^\()]|\.)+` represents a property whose key is `Msymbol`. In the element, `\t`, `\n`, `\r`, and `\e` are replaced with tab (code 9), newline (code 10), carriage return (code 13), and escape (code 27) respectively. Other characters following a backslash is interpreted as it is. The value of the property is the symbol having the resulting string as its name.

  For instance, the element `abc\ def` represents a property whose value is the symbol having the name "abc def".

- MTEXT

  An element that matches the regular expression `"([^"]|\")*"` represents a property whose key is `Mtext`. The backslash escape explained above also applies here. Moreover, each part in the element matching the regular expression `\[xX][0-9A-Fa-f][0-9A-Fa-f]` is replaced with its hexadecimal interpretation.

  After having resolved the backslash escapes, the byte sequence between the double quotes is interpreted as a UTF-8 sequence and decoded into an M-text. This M-text is the value of the property.

- PLIST

  Zero or more elements surrounded by a pair of parentheses represent a property whose key is `Mplist`. Whitespaces before and after a parenthesis can be omitted. The value of the property is a plist, which is the result of recursive interpretation of the elements between the parentheses.

D.1.2 SYNTAX NOTATION

In an explanation of a plist format of data, a BNF-like notation is used. In the notation, non-terminals are represented by a string of uppercase letters (including '-' in the middle), terminals are represented by a string surrounded by '"'. Special non-terminals INTEGER, SYMBOL, MTEXT and PLIST represents property integer, symbol, M-text, or plist respectively.

D.1.3 EXAMPLE

Here is an example of database data that is read into a plist of this simple format:

```
DATA-FORMAT ::=
    [ INTEGER | SYMBOL | MTEXT | FUNC ] *

FUNC ::=
    '(' FUNC-NAME FUNC-ARG * ')'

FUNC-NAME ::=
    SYMBOL

FUNC-ARG ::=
    INTEGER | SYMBOL | MTEXT | '(' FUNC-ARG ')'
```

For instance, a data file that contains this text matches the above syntax:

```
abc 123 (pqr 0xff) "m\"text" (_\\_ ("string" xyz) -456)
```

and is read into this plist:

```
1st element: key: Msymbol,  value: abc
2nd element: key: Minteger, value: 123
3rd element: key: Mplist,   value: a plist of these elements:
    1st element: key Msymbol,  value: pgr
    2nd element: key Minteger, value: 255
4th element: key: Mtext,    value: m"text
5th element: key: Mplist,   value: a plist of these elements:
    1st element: key: Msymbol, value: _\_
    2nd element: key: Mplist,  value: a plist of these elements:
        1st element: key: Mtext,    value: string
2nd element: key: Msymbol,  value: xyz
3rd element: key: Minteger, value: -456
```

D.2 List of character set definitions

D.2.1 DESCRIPTION

The m17n library loads a list of charset definitions from the data of tag <charset-list>. The data is loaded as a plist of this format.

```
CHARSET-LIST ::= DEFINITION *

DEFINITION ::= '(' NAME ( KEY VALUE ) * ')'

NAME ::= SYMBOL

KEY ::= SYMBOL

VALUE ::= SYMBOL | INTEGER | MTEXT | PLIST
```

`NAME` is a name of a charset to define.

`KEY` and `VALUE` pair is a property given to the function **mchar_define_charset()** (p. 66) as an element of the second argument **plist**.

D.2.2 SEE ALSO

mdbGeneral(5) (p. 208), **mchar_define_charset()** (p. 66)

D.3 List of coding system definitions

D.3.1 DESCRIPTION

The m17n library loads a list of coding system definitions from the m17n database by the tags <coding-list> at initialization time. The data is loaded as a plist of this format.

```
CODING-LIST ::= DEFINITION *

DEFINITION ::= '(' NAME ( KEY VALUE ) * ')'
NAME ::= SYMBOL

KEY ::= SYMBOL

VALUE ::= SYMBOL | INTEGER | MTEXT | PLIST
```

`NAME` is a name of a coding system to define.

`KEY` and `VALUE` pair is a property given to the function **mconv_define_coding()** (p. 77) as the second argument.

D.3.2 SEE ALSO

mdbGeneral(5) (p. 208), **mconv_define_coding()** (p. 77)

D.4 List of data in a database directory.

D.4.1 DESCRIPTION

The m17n library loads a list of definitions of data of the m17n database from files of name "mdb.dir" in each database directory at initialization time. The plist format of this file is as follows:

```
MDB-DIR ::= DEFINITION *

DEFINITION ::= '(' TAG0 [ TAG1 [ TAG2 [ TAG3 ] ] ] FILE [ VERSION ]')'

TAGn ::= SYMBOL

FILE ::= MTEXT

VERSION ::= MTEXT
```

If TAG0 is neither ‘charset’ nor ‘char-table’, and TAGn (n > 0) is a symbol ‘*’, FILE can contain a wildcard charater, and all files matching FILE accoding to the rules used by the shell are the target of database files. In that case, each file must contain SELF-DEFINITION which is a plist element providing the actual TAGn values by the form:

```
SELF-DEFINITION ::= '(' TAG0 TAG1 TAG2 TAG3 [ VERSION ] ')'
```

For instance, if a database directory contains these files:

```
zh-py.mim:
(input-method zh py)

ko-han2.mim:
(input-method ko han2)
```

these lines in "mdb.dir":

```
(input-method zh py "zh-py.mim")
(input-method ko han2 "ko-han2.mim")
```

can be shortened to this single line:

```
(input-method * "*.mim")
```

VERSION is a required version number of the m17n library. The format is "XX.YY.ZZ" where XX is a major version number, YY is a minor version number, and ZZ is a patch level.

D.5 Font Layout Table

D.5.1 DESCRIPTION

For simple scripts, the rendering engine converts character codes into glyph codes one by one by consulting the encoding of each selected font. But, to render text that requires complicated layout (e.g. Thai and Indic scripts), one to one conversion is not sufficient. A sequence of characters may have to be drawn as a single ligature. Some glyphs may have to be drawn at 2-dimensionally shifted positions.

To handle those complicated scripts, the m17n library uses Font Layout Tables (FLTs for short). The FLT driver interprets an FLT and converts a character sequence into a glyph sequence that is ready to be passed to the rendering engine.

An FLT can contain information to extract a grapheme cluster from a character sequence and to reorder the characters in the cluster, in addition to information found in OpenType Layout Tables (CMAP, GSUB, and GPOS).

An FLT is a cascade of one or more conversion stages. In each stage, a sequence is converted into another sequence to be read in the next stage. The length of sequences may differ from stage to stage. Each element in a sequence has the following integer attributes.

- code

  In the first conversion stage, this is the character code in the original character sequence. In the last stage, it is the glyph code passed to the rendering engine. In other cases, it is an intermediate glyph code.

- category

  The category code defined in the `CATEGORY-TABLE` of the current stage, or defined in the one of the former stages and not overwritten by later stages.

- combining-spec

  If nonzero, it specifies how to combine this (intermediate) glyph with the previous one.

- left-padding-flag

  If nonzero, it instructs the rendering function to insert a padding space before this (intermediate) glyph so that the glyph does not overlap with the previous one.

- right-padding-flag

  If nonzero, it instructs the rendering function to insert a padding space after this (intermediate) glyph so that the glyph does not overlap with the next one.

When the layout engine draws text, it at first determines a font and an FLT for each character in the text. For each subsequence of characters that use the same font and FLT, the layout engine generates a corresponding intermediate glyph sequence. The code attribute of each element in the intermediate glyph sequence is its character code, and all other attributes are zeros. This sequence is processed in the first stage of FLT as the current *run* (substring).

Each stage works as follows.

At first, if the stage has a `CATEGORY-TABLE`, the category of each glyph in the current run is updated. If there is a glyph that has no category, the current run ends before that glyph.

Then, the default values of code-offset, combining-spec, and left-padding-flag of this stage are initialized to zero.

Next, the initial conversion rule of the stage is applied to the current run.

Lastly, the current run is replaced with the newly produced (intermediate) glyph sequence.

D.5.2 SYNTAX and SEMANTICS

The m17n library loads an FLT from the m17n database using the tag <font, layouter, FLT-NAME>. The date format of an FLT is as follows:

```
FONT-LAYOUT-TABLE ::= FLT-DECLARATION ? STAGE0 STAGE *

FLT-DECLARATION ::= '(' 'font' 'layouter' FLT-NAME nil PROP * ')'
FLT-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
```

In the definition of `CATEGORY-SPEC`, `CODE` is a glyph code, and `CATEGORY` is ASCII code of an upper or lower letter, i.e. one of 'A', ... 'Z', 'a', .. 'z'.

The first form of `CATEGORY-SPEC` assigns `CATEGORY` to a glyph whose code is `CODE`. The second form assigns `CATEGORY` to glyphs whose code falls between the two `CODEs`.

```
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 + ')'
```

Each `RULE` specifies glyphs to be consumed and glyphs to be produced. When some glyphs are consumed, they are taken away from the current run. A rule may fail in some condition. If not described explicitly to fail, it should be regarded that the rule succeeds.

```
DIRECT-CODE ::= INTEGER
```

This rule consumes no glyph and produces a glyph which has the following attributes:

- code : `INTEGER` plus the default code-offset
- combining-spec : default value
- left-padding-flag : default value
- right-padding-flag : zero

After having produced the glyph, the default code-offset, combining-spec, and left-padding-flag are all reset to zero.

```
PREDEFINED-RULE ::= '=' | '*' | '<' | '>' | '|' | '[' | ']'
```

They perform actions as follows.

- =

  This rule consumes the first glyph in the current run and produces the same glyph. It fails if the current run is empty.

- *

  This rule repeatedly executes the previous rule. If the previous rule fails, this rule does nothing and fails.

- <

  This rule specifies the start of a grapheme cluster.

- >

  This rule specifies the end of a grapheme cluster.

- `@[`

  This rule sets the default left-padding-flag to 1. No glyph is consumed. No glyph is produced.

- `@]`

  This rule changes the right-padding-flag of the lastly generated glyph to 1. No glyph is consumed. No glyph is produced.

- `|`

  This rule consumes no glyph and produces a special glyph whose category is ' ' and other attributes are zero. This is the only rule that produces that special glyph.

```
REGEXP-BLOCK ::= '(' REGEXP RULE * ')'

REGEXP ::= MTEXT
```

`MTEXT` is a regular expression that should match the sequence of categories of the current run. If a match is found, this rule executes `RULEs` temporarily limiting the current run to the matched part. The matched part is consumed by this rule.

Parenthesized subexpressions, if any, are recorded to be used in `MATCH-BLOCK` that may appear in one of `RULEs`.

If no match is found, this rule fails.

```
MATCH-BLOCK ::= '(' MATCH-INDEX RULE * ')'

MATCH-INDEX ::= INTEGER
```

`MATCH-INDEX` is an integer specifying a parenthesized subexpression recorded by the previous `REGEXP-BLOCK`. If such a subexpression was found by the previous regular expression matching, this rule executes `RULEs` temporarily limiting the current run to the matched part of the subexpression. The matched part is consumed by this rule.

If no match was found, this rule fails.

If this is the first rule of the stage, `MATCH-INDEX` must be 0, and it matches the whole current run.

```
SUBST-BLOCK ::= '(' SOURCE-PATTERN RULE * ')'

SOURCE-PATTERN ::= '(' CODE + ')'
                   | (' 'range' CODE CODE ')'
```

If the sequence of codes of the current run matches `SOURCE-PATTERN`, this rule executes `RULEs` temporarily limiting the current run to the matched part. The matched part is consumed.

The first form of `SOURCE-PATTERN` specifies a sequence of glyph codes to be matched. In this case, this rule resets the default code-offset to zero.

The second form specifies a range of codes that should match the first glyph code of the code sequence. In this case, this rule sets the default code-offset to the first glyph code minus the first `CODE` specifying the range.

If no match is found, this rule fails.

```
FONT-FACILITY-BLOCK ::= '(' FONT-FACILITY RULE * ')'
FONT-FACILITY = '(' 'font-facility' CODE * ')'
       | '(' 'font-facility' FONT-SPEC ')'
```

If the current font has glyphs for `CODEs` or matches with `FONT-SPEC`, this rule succeeds and `RULEs` are executed. Otherwise, this rule fails.

```
COND-BLOCK ::= '(' 'cond' RULE + ')'
```

This rule sequentially executes `RULEs` until one succeeds. If no rule succeeds, this rule fails. Otherwise, it succeeds.

```
OTF-SPEC ::= SYMBOL
```

`OTF-SPEC` is a symbol whose name specifies an instruction to the OTF driver. The name has the following syntax.

```
  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 | '*' ]
```

Each `SYMBOL` specifies a tag name defined in the OpenType specification.

For `SCRIPT`, `SYMBOL` specifies a Script tag name (e.g. deva for Devanagari).

For `LANGSYS`, `SYMBOL` specifies a Language System tag name. If `LANGSYS` is omitted, the Default Language System table is used.

For `GSUB-FEATURES`, each `SYMBOL` in `FEATURE-LIST` specifies a GSUB Feature tag name to apply. '∗' is allowed as the last item to specify all remaining features. If `SYMBOL` is preceded by '∼' and the last item is '∗', `SYMBOL` is excluded from the features to apply. If no `SYMBOL` is specified, no GSUB feature is applied. If `GSUB-FEATURES` itself is omitted, all GSUB features are applied.

When `OTF-SPEC` appears in a `FONT-SPEC`, `FEATURE-LIST` specifies features that the font must have (or must not have if preceded by '∼'), and the last'∗', even if exists, has no meaning.

The specification of `GPOS-FEATURES` is analogous to that of `GSUB-FEATURES`.

Please note that all the tags above must be 4 ASCII printable characters.

See the following page for the OpenType specification.

`<http://www.microsoft.com/typography/otspec/default.htm>`

```
COMBINING ::= SYMBOL
```

`COMBINING` is a symbol whose name specifies how to combine the next glyph with the previous one. This rule sets the default combining-spec to an integer code that is unique to the symbol name. The name has the 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 and HPOS specify the vertical and horizontal positions as described below.

```
                              POINT VPOS HPOS
                              ----- ---- ----
  0----1----2 <---- top       0     t    l
  |         |                 1     t    c
  |         |                 2     t    r
  |         |                 3     B    l
  9   10   11 <---- center    4     B    c
  |         |                 5     B    r
--3----4----5-- <-- baseline  6     b    l
  |         |                 7     b    c
  6----7----8 <---- bottom    8     b    r
                              9     c    l
  |    |    |                10     c    c
left center right            11     c    r
```

The left figure shows 12 reference points of a glyph by numbers 0 to 11. The rectangle 0-6-8-2 is the bounding box of the glyph, the positions 3, 4, and 5 are on the baseline, 9-11 are on the vertical center of the box, 0-2 and 6-8 are on the top and on the bottom respectively. 1, 10, 4, and 7 are on the horizontal center of the box.

The right table shows how those reference points are specified by a pair of VPOS and HPOS.

The first VPOS and HPOS in the definition of COMBINING-NAME specify the reference point of the previous glyph, and the second VPOS and HPOS specify that of the next glyph. The next glyph is drawn so that these two reference points align.

OFFSET specifies the way of alignment in detail. If it is ’.’, the reference points are on the same position.

XOFF specifies how much the X position of the reference point of the next glyph should be shifted to the left (’<’) or right (’>’) from the previous reference point.

YOFF specifies how much the Y position of the reference point the next glyph should be shifted upward (’+’) or downward (’-’) from the previous reference point.

In both cases, INTEGER is the amount of shift expressed as a percentage of the font size, i.e., if INTEGER is 10, it means 10% (1/10) of the font size. If INTEGER is omitted, it is assumed that 5 is specified.

Once the next glyph is combined with the previous one, they are treated as a single combined glyph.

```
MACRO-NAME ::= SYMBOL
```

MACRO-NAME is a symbol that appears in one of MACRO-DEF. It is exapanded to the sequence of the correponding RULEs.

D.5.3 CONTEXT DEPENDENT BEHAVIOR

So far, it has been assumed that each sequence, which is drawn with a specific font, is context free, i.e. not affected by the glyphs preceding or following that sequence. This is true when sequence S1 is drawn with font F1 while the preceding sequence S0 unconditionally requires font F0.

```
sequence                              S0      S1
currently used font                   F0      F1
usable font(s)                        F0      F1
```

Sometimes, however, a clear separation of sequences is not possible. Suppose that the preceding sequence S0 can be drawn not only with F0 but also with F1.

```
  sequence                             S0       S1
  currently used font                  F0       F1
  usable font(s)                       F0,F1    F1
```

In this case, glyphs used to draw the preceding S0 may affect glyph generation of S1. Therefore it is necessary to access information about S0, which has already been processed, when processing S1. Generation rules in the first stage (only in the first stage) accept a special regular expression to access already processed parts.

```
  "RE0 RE1"
```

`RE0` and `RE1` are regular expressions that match the preceding sequence S0 and the following sequence S1, respectively.

Pay attention to the space between the two regular expressions. It represents the special category ' ' (see above). Note that the regular expression above belongs to glyph generation rules using font F1, therefore not only RE1 but also RE0 must be expressed with the categories for F1. This means when the preceding sequence S0 cannot be expressed with the categories for F1 (as in the first example above) generation rules having these patterns never match.

D.5.4 SEE ALSO

mdbGeneral(5) (p. 208), **FLTs provided by the m17n database** (p. 246)

D.6 Font Encoding

D.6.1 DESCRIPTION

The m17n library loads information about the encoding of each font form the m17n database by the tags <font, encoding>. The data is loaded as a plist of this format.

```
FONT-ENCODING ::= PER-FONT *

PER-FONT ::= '(' FONT-SPEC ENCODING [ REPERTORY ] ')'

FONT-SPEC ::=
    '(' [ FOUNDRY FAMILY
        [ WEIGHT [ STYLE [ STRETCH [ ADSTYLE ]]]]]
REGISTRY ')'

ENCODING ::= SYMBOL
```

`FONT-SPEC` is to specify properties of a font. `FOUNDRY` to `REGISTRY` are symbols corresponding to **Mfoundry** (p. 124) to **Mregistry** (p. 125) property of a font. See **Font** (p. 116) for the meaning of each property.

For instance, this `FONT-SPEC`:

```
    (nil alice0\ lao iso8859-1)
```

should be applied to all fonts whose family name is "alice0 lao", and registry is "iso8859-1".

`ENCODING` is a symbol representing a charset. A font matching `FONT-SPEC` supports all characters of the charset, and a character code is mapped to the corresponding glyph code of the font by this charset.

`REPERTORY` is a symbol representing a charset or "nil". Omitting it is the same as specifying `ENCODING` as `REPERTORY`. If it is not "nil", the charset specifies the repertory of the font, i.e, which character it supports. Otherwise, whether a specific character is supported by the font or not is asked to each font driver.

For so called Unicode fonts (registry is "iso10646-1"), it is recommended to specify "nil" as `REPERTORY` because such fonts usually supports only a subset of Unicode characters.

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. 124) to **Mregistry** (p. 125) property of a font. See **Font** (p. 116) 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 Fontset

D.8.1 DESCRIPTION

The m17n library loads a fontset definition from the m17n database by the tags <fontset, FONTSET-NAME>. The plist format of the data is as follows:

```
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
 [ OTF-SPEC ] [ LANG-SPEC ] ')'
```

`SCRIPT` is a symbol of script name (e.g. latin, han) or `nil`. `LANGUAGE` is a two-letter symbol of language name code defined by ISO 639 (e.g. ja, zh) or `nil`.

`FONT-SPEC` is to specify properties of a font. `FOUNDRY` to `REGISTRY` are symbols corresponding to **Mfoundry** (p. 124) to **Mregistry** (p. 125) property of a font. See **Font** (p. 116) for the meaning of each property.

`OTF-SPEC` is a symbol specifing the required OTF features. The symbol name has the following syntax.

```
OTF-SPEC-NAME ::= ':otf=' SCRIPT LANGSYS ? GSUB-FEATURES ? GPOS-FEATURES ?

SCRIPT ::= SYMBOL
LANGSYS ::= '/' SYMBOL

GSUB-FEATURES ::= '=' FEATURE-LIST ?

GPOS-FEATURES ::= '+' FEATURE-LIST ?

FEATURE-LIST ::= '~' ? FEATURE ( ',' '~' ? FEATURE ',' )
```

Here, `FEATURE` is a four-letter Open Type feature.

`LANG-SPEC` is a symbol specifying the required language support. The symbol name has the following syntax.

```
LANG-SPEC-NAME ::= ':lang=' LANG
```

Here, `LANG` is a two or three-letter ISO-639 language code.

`FLT-NAME` is a name of Font Layout Table (**Font Layout Table** (p. 211)).

D.8.2 EXAMPLE

This is an example of `PER_SCRIPT`.

```
(han
  (ja
    ((jisx0208.1983-0)))
  (zh
    ((gb2312.1980-0)))
  (nil
    ((big5-0))))
```

It instructs the font selector to use a font of registry "jisx0208.1983-0" for a "han" character (i.e. a character whose **Mscript** (p. 26) property is 'han') if the character has **Mlanguage** (p. 48) text property "ja" in an M-text and the character is in the repertories of such fonts. Otherwise, try a font of registry "gb2312.1980-0" or "big5-0". If that "han" character does not have **Mlanguage** (p. 48) text property, try all three fonts.

See the function **mdraw_text()** (p. 145) for the detail of how a font is selected.

D.9 Input Method

D.9.1 DESCRIPTION

The m17n library provides a driver for input methods that are dynamically loadable from the m17n database (see **Input Method (basic)** (p. 92) (P.92)).

This section describes the data format that defines those input methods.

D.9.2 SYNTAX and SEMANTICS

The following data format defines an input method. The driver loads a definition from a file, a stream, etc. The definition is converted into the form of plist in the driver.

```
INPUT-METHOD ::=
    IM-DECLARATION ? IM-DESCRIPTION ? TITLE ?
     VARIABLE-LIST ? COMMAND-LIST ?  MODULE-LIST ?
     MACRO-LIST ? MAP-LIST ? STATE-LIST ?

IM-DECLARATION ::= '(' 'input-method' LANGUAGE NAME EXTRA-ID ? VERSION ? ')'
LANGUAGE ::= SYMBOL
NAME ::= SYMBOL
EXTRA-ID ::= SYMBOL
VERSION ::= '(' 'version' VERSION-NUMBER ')'

IM-DESCRIPTION ::= '(' 'description' DESCRIPTION ')'
DESCRIPTION ::= MTEXT-OR-GETTEXT | 'nil'
MTEXT-OR-GETTEXT ::=  [ MTEXT | '(' '_' MTEXT ')']

TITLE ::= '(' 'title' TITLE-TEXT ')'
TITLE-TEXT ::= MTEXT

VARIABLE-LIST ::= '(' 'variable' VARIABLE-DECLARATION * ')'
VARIABLE-DECLARATION ::=  '(' VAR-NAME [ DESCRIPTION VALUE VALUE-CANDIDATE * ]')'
VAR-NAME ::= SYMBOL
VALUE ::= MTEXT | SYMBOL | INTEGER
VALUE-CANDIDATE ::= VALUE | '(' RANGE-FROM RANGE-TO ')'
RANGE-FROM ::= INTEGER
RANGE-TO ::= INTEGER

COMMAND-LIST ::= '(' 'command' COMMAND-DECLARATION * ')'
COMMAND-DECLARATION ::=  '(' CMD-NAME [ DESCRIPTION KEYSEQ * ] ')'
CMD-NAME ::= SYMBOL
```

`IM-DECLARATION` specifies the language and name of this input method.

When `LANGUAGE` is `t`, the use of the input method is not limited to one language.

When `NAME` is `nil`, the input method is not standalone, but is expected to be used in other input methods. In such cases, `EXTRA-ID` is required to identify the input method.

`VERSION` specifies the required minimum version number of the m17n library. The format is "XX.YY.ZZ" where XX is a major version number, YY is a minor version number, and ZZ is a patch level.

`DESCRIPTION`, if not nil, specifies the description text of an input method, a variable or a command. If `MTEXT-OR-GETTEXT` takes the second form, the text is translated according to the current locale by "gettext" (if the translation is provided).

`TITLE-TEXT` is a text displayed on the screen when this input method is active.

There is one special input method file "global.mim" that declares common variables and commands. The input method driver always loads this file and other input methods can inherit the variables and the commands.

`VARIABLE-DECLARATION` declares a variable used in this input method. If a variable must be initialized to the default value, or is to be customized by a user, it must be declared here. The declaration can be used in two ways. One is to introduce a new variable. In that case, `VALUE` must not be omitted. Another is to inherit the variable from what declared in "global.mim", and to give the different default value and/or to make the variable customizable specially for the current input method. In the latter case, `VALUE` can be omitted.

`COMMAND-DECLARATION` declares a command used in this input method. If a command must be bound to the default key sequence, or is to be customized by a user, it must be declared here. Like

`VARIABLE-DECLARATION`, the declaration can be used in two ways. One is to introduce a new command. In that case, `KEYSEQ` must not be omitted. Another is to inherit the command from what declared in "global.mim", and to give the different key binding and/or to make the command customizable specially for the current input method. In the latter case, `KEYSEQ` can be omitted.

```
MODULE-LIST ::= '(' 'module' MODULE * ')'

MODULE ::= '(' MODULE-NAME FUNCTION * ')'

MODULE-NAME ::= SYMBOL

FUNCTION ::= SYMBOL
```

Each `MODULE` declares the name of an external module (i.e. dynamic library) and function names exported by the module. If a `FUNCTION` has name "init", it is called with only the default arguments (see the section about `CALL`) when an input context is created for the input method. If a `FUNCTION` has name "fini", it is called with only the default arguments when an input context is destroyed.

```
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` includes macros from another input method specified by `TAGS`. When `MACRO-NAME` is not given, all macros from the input method are included.

```
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 ? ')'
```

When an input method is never standalone and always included in another method, `MAP-LIST` can be omitted.

`SYMBOL` in the definitions of `MAP-NAME` must not be `t` nor `nil`.

`MTEXT` in the definition of `KEYSEQ` consists of characters that can be generated by a keyboard. Therefore `MTEXT` usually contains only ASCII characters. However, if the input method is intended to be used, for instance, with a West European keyboard, `MTEXT` may contain Latin-1 characters.

`SYMBOL` in the definition of `KEYSEQ` must be the return value of the **minput_event_to_key**() (p. 150) function. Under the X window system, you can quickly check the value using the `xev` command. For example, the return key, the backspace key, and the 0 key on the keypad are represented as (Return) , (BackSpace) , and (KP_0) respectively. If the shift, control, meta, alt, super, and hyper modifiers are used, they are represented by the S- , C- , M- , A- , s- , and H- prefixes respectively in this order. Thus, "return with shift with meta with hyper" is (S-M-H-Return) . Note that "a with shift" .. "z with shift" are represented simply as A .. Z . Thus "a with shift with meta with hyper" is (M-H-A) .

`INTEGER` in the definition of `KEYSEQ` must be a valid character code.

`MAP-INCLUSION` includes maps from another input method specified by `TAGS`. When `MAP-NAME` is not given, all maps from the input method are included.

```
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 ? ')'
```

When an input system is never standalone and always included in another system, `STATE-LIST` can be omitted.

`STATE-INCLUSION` includes states from another input method specified by `TAGS`. When `STATE-NAME` is not given, all states from the input method are included.

The optional `STATE-TITLE-TEXT` specifies a title text displayed on the screen when the input method is in this state. If `STATE-TITLE-TEXT` is omitted, `TITLE-TEXT` is used.

In the first form of `BRANCH`, `MAP-NAME` must be an item that appears in `MAP`. In this case, if a key sequence matching one of `KEYSEQs` of `MAP-NAME` is typed, `BRANCH-ACTIONs` are executed.

In the second form of `BRANCH`, `BRANCH-ACTIONs` are executed if a key sequence that doesn't match any of `Branch's` of the current state is typed.

If there is no `BRANCH` beginning with `nil` and the typed key sequence does not match any of the current `BRANCHs`, the input method transits to the initial state.

In the third form of `BRANCH`, `BRANCH-ACTIONs` are executed when shifted to the current state. If the current state is the initial state, `BRANCH-ACTIONs` are executed also when an input context of the input method is created.

```
BRANCH-ACTION ::= ACTION
```

An input method has the following two lists of symbols.

- marker list

  A marker is a symbol indicating a character position in the preediting text. The `MARK` action assigns a position to a marker. The position of a marker is referred by the `MOVE` and the `DELETE` actions.

- variable list

A variable is a symbol associated with an integer, a symbol, or an M-text value. The integer value of a variable can be set and referred by the `SET` action. It can be referred by the `SET`, the `INSERT`, the `SELECT`, the `UNDO`, the `IF`, the `COND` actions. The M-text value of a variable can be referred by the `INSERT` action. The symbol value of a variable can not be referred directly, is used the library implicitly (e.g. candidates-charset). All variables are implicitly initialized to the integer value zero.

Each `PREDEFINED-SYMBOL` has a special meaning when used as a marker.

- `@0`, `@1`, `@2`, `@3`, `@4`, `@5`, `@6`, `@7`, `@8`, `@9`

  The 0th, 1st, 2nd, ... 9th position respectively.
- `@<`, `@=`, `@>`

  The first, the current, and the last position.
- `@-`, `@+`

  The previous and the next position.
- `@[`, `@]`

  The previous and the next position where a candidate list changes.

Some of the `PREDEFINED-SYMBOL` has a special meaning when used as a candidate index in the `SELECT` action.

- `@<`, `@=`, `@>`

  The first, the current, and the last candidate of the current candidate group.
- `@-`

  The previous candidate. If the current candidate is the first one in the current candidate group, then it means the last candidate in the previous candidate group.
- `@+`

  The next candidate. If the current candidate is the last one in the current candidate group, then it means the first candidate in the next candidate group.
- `@[`, `@]`

  The candidate in the previous and the next candidate group having the same candidate index as the current one.

And, this also has a special meaning.

- `@@`

  Number of handled keys at that moment.

These are for supporting surround text handling.

- `@-0`

  -1 if surrounding text is supported, -2 if not.
- `@-N`

  Here, `N` is a positive integer. The value is 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. When this is used as the argument of `delete` action, it specifies the number of characters to be deleted.

- `@+N`

  Here, `N` is a positive integer. The value is 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. When this is used as the argument of `delete` action, it specifies the number of characters to be deleted.

The arguments and the behavior of each action are listed below.

```
INSERT ::= '(' 'insert' MTEXT ')'
           | MTEXT
   | INTEGER
   | SYMBOL
           | '(' 'insert' SYMBOL ')'
           | '(' 'insert' '(' CANDIDATES * ')' ')'
           | '(' CANDIDATES * ')'

CANDIDATES ::= MTEXT | '(' MTEXT * ')'
```

The first and second forms insert `MTEXT` before the current position.

The third form inserts the character `INTEGER` before the current position.

The fourth and fith form treats `SYMBOL` as a variable, and inserts its value (if it is a valid character code) before the current position.

In the sixth and seventh forms, each `CANDIDATES` represents a candidate group, and each element of `CANDIDATES` represents a candidate, i.e. if `CANDIDATES` is an M-text, the candidates are the characters in the M-text; if `CANDIDATES` is a list of M-texts, the candidates are the M-texts in the list.

These forms insert the first candidate before the current position. The inserted string is associated with the list of candidates and the information indicating the currently selected candidate.

The marker positions affected by the insertion are automatically relocated.

```
DELETE ::= '(' 'delete' SYMBOL ')'
           | '(' 'delete' INTEGER ')'
```

The first form treats `SYMBOL` as a marker, and deletes characters between the current position and the marker position.

The second form treats `INTEGER` as a character position, and deletes characters between the current position and the character position.

The marker positions affected by the deletion are automatically relocated.

```
SELECT ::= '(' 'select' PREDEFINED-SYMBOL ')'
           | '(' 'select' INTEGER ')'
   | '(' 'select' SYMBOL ')'
```

This action first checks if the character just before the current position belongs to a string that is associated with a candidate list. If it is, the action replaces that string with a candidate specified by the argument.

The first form treats `PREDEFINED-SYMBOL` as a candidate index (as described above) that specifies a new candidate in the candidate list.

The second form treats `INTEGER` as a candidate index that specifies a new candidate in the candidate list.

In the third form, `SYMBOL` must have a integer value, and it is treated as a candidate index.

```
SHOW ::= '(show)'
```

This actions instructs the input method driver to display a candidate list associated with the string before the current position.

```
HIDE ::= '(hide)'
```

This action instructs the input method driver to hide the currently displayed candidate list.

```
MOVE ::= '(' 'move' SYMBOL ')'
         | '(' 'move' INTEGER ')'
```

The first form treats `SYMBOL` as a marker, and makes the marker position be the new current position.

The second form treats `INTEGER` as a character position, and makes that position be the new current position.

```
MARK ::= '(' 'mark' SYMBOL ')'
```

This action treats `SYMBOL` as a marker, and sets its position to the current position. `SYMBOL` must not be a `PREDEFINED-SYMBOL`.

```
PUSHBACK :: = '(' 'pushback' INTEGER ')'
              | '(' 'pushback' KEYSEQ ')'
```

The first form pushes back the latest `INTEGER` number of key events to the event queue if `INTEGER` is positive, and pushes back all key events if `INTEGER` is zero.

The second form pushes back keys in `KEYSEQ` to the event queue.

```
POP ::= '(' 'pop' ')'
```

This action pops the first key event that is not yet handled from the event queue.

```
UNDO :: = '(' 'undo' [ INTEGER | SYMBOL ] ')'
```

If there's no argument, this action cancels the last two key events (i.e. the one that invoked this command, and the previous one).

If there's an integer argument NUM, it must be positive or negative (not zero). If positive, from the NUMth to the last events are canceled. If negative, the last (- NUM) events are canceled.

If there's a symbol argument, it must be resolved to an integer number and the number is treated as the actual argument as above.

```
COMMIT :: = '(commit)'
```

This action commits the current preedit.

```
UNHANDLE :: = '(unhandle)'
```

This action commits the current preedit and returns the last key as unhandled.

```
SHIFT :: = '(' 'shift' STATE-NAME ')'
```

If `STATE-NAME` is `t`, this action shifts the current state to the previous one, otherwise it shifts to `STATE-NAME`. In the latter case, `STATE-NAME` must appear in `STATE-LIST`.

```
CALL ::= '(' 'call' MODULE-NAME FUNCTION ARG * ')'
```

```
ARG ::= INTEGER | SYMBOL | MTEXT | PLIST
```

This action calls the function `FUNCTION` of external module `MODULE-NAME`. `MODULE-NAME` and `FUNCTION` must appear in `MODULE-LIST`.

The function is called with an argument of the type (**MPlist** (p. 19) $*$). The key of the first element is **Mt** (p. 17) and its value is a pointer to an object of the type **MInputContext** (p. 186). The key of the second element is **Msymbol** (p. 17) and its value is the current state name. `ARGs` are used as the value of the third and later elements. Their keys are determined automatically; if an `ARG` is an integer, the corresponding key is **Minteger** (p. 23); if an `ARG` is a symbol, the corresponding key is **Msymbol** (p. 17), etc.

The function must return NULL or a value of the type (**MPlist** (p. 19) $*$) that represents a list of actions to take.

```
SET ::= ’(’ CMD SYMBOL1 EXPRESSION ’)’

CMD ::= ’set’ | ’add’ | ’sub’ | ’mul’ | ’div’

EXPRESSION ::= INTEGER | SYMBOL2 | ’(’ OPERATOR EXPRESSION * ’)’

OPERATOR ::= ’+’ | ’-’ | ’*’ | ’/’ | ’|’ | ’&’ | ’!’
            | ’=’ | ’<’ | ’>’ | ’<=’ | ’>=’
```

This action treats `SYMBOL1` and `SYMBOL2` as variables and sets the value of `SYMBOL1` as below.

If `CMD` is ’set’, it sets the value of `SYMBOL1` to the value of `EXPRESSION`.

If `CMD` is ’add’, it increments the value of `SYMBOL1` by the value of `EXPRESSION`.

If `CMD` is ’sub’, it decrements the value of `SYMBOL1` by the value of `EXPRESSION`.

If `CMD` is ’mul’, it multiplies the value of `SYMBOL1` by the value of `EXPRESSION`.

If `CMD` is ’div’, it divides the value of `SYMBOL1` by the value of `EXPRESSION`.

```
IF ::= ’(’ CONDITION ACTION-LIST1 ACTION-LIST2 ? ’)’

CONDITION ::= [ ’=’ | ’<’ | ’>’ | ’<=’ | ’>=’ ] EXPRESSION1 EXPRESSION2

ACTION-LIST1 ::= ’(’ ACTION * ’)’

ACTION-LIST2 ::= ’(’ ACTION * ’)’
```

This action performs actions in `ACTION-LIST1` if `CONDITION` is true, and performs `ACTION-LIST2` (if any) otherwise.

```
COND ::= ’(’ ’cond’ [ ’(’ EXPRESSION ACTION * ’) ] * ’)’
```

This action performs the first action `ACTION` whose corresponding `EXPRESSION` has nonzero value.

D.9.3 EXAMPLE 1

This is a very simple example for inputting Latin characters with diacritical marks (acute and cedilla). For instance, when you type:

```
    Comme’die-Franc,aise, chic,,
```

you will get this:

The definition of the input method is very simple as below, and it is quite straight forward to extend it to cover all Latin characters.

D.9.4 EXAMPLE 2

This example is for inputting Unicode characters by typing C-u (Control-u) followed by four hexadecimal digits. For instance, when you type ("^u" means Control-u):

```
^u2190^u2191^u2192^u2193
```

you will get this (Unicode arrow symbols):

The definition utilizes `SET` and `IF` commands as below:

```
(title "UNICODE")
(map
 (starter
  ((C-U) "U+"))
 (hex
  ("0" ?0) ("1" ?1) ... ("9" ?9) ("a" ?A) ("b" ?B) ... ("f" ?F)))
(state
 (init
  (starter (set code 0) (set count 0) (shift unicode)))
 (unicode
  (hex (set this @-)
       (< this ?A
  ((sub this 48))
  ((sub this 55)))
       (mul code 16) (add code this)
       (add count 1)
       (= count 4
  ((delete @<) (insert code) (shift init))))))
```

D.9.5 EXAMPLE 3

This example is for inputting Chinese characters by typing PinYin key sequence.

D.9.6 SEE ALSO

Input Methods provided by the m17n database (p. 231), **mdbGeneral(5)** (p. 208)

Appendix E

Data provided by the m17n database

- **Character Property** (p. 230)
- **Input method** (p. 231)
- **Font Layout Table** (p. 246)
- **Fontset** (p. 248)
- **The other data** (p. 251)

E.1 Character Property

- CATEGORY.tab

  Unicode general category for each character that is available as **Mcategory** (p. 27) property.

- COMBINE.tab

  Unicode combining class for each character that is available as **Mcombining_class** (p. 27) property.

- BIDI.tab

  Unicode BIDI category for each character that is available as **Mbidi_category** (p. 27) property.

- CASE-S.tab

  Unicode case-folding mapping of each character that is available as **Msimple_case_folding** (p. 27) property.

- CASE-C.tab

  Unicode complicated case-folding mapping of each character that is available as **Mcomplicated_case_folding** (p. 27) property.

- NAME.tab

  Unicode character name for each character that is available as **Mname** (p. 27) property.

- SCRIPT.tab

  Unicode script name for each character that is available as **Mscript** (p. 26) property.

- CASED.tab

  Unicode properties for case operations. Integer value 1 means cased (D47, Unicode 4.0, p.89), 2 means case-ignorable (D47a, Unicode 4.1.0), and 3 means both. Available as **Mcased** (p. 28) property.

- SOFT-DOTTED.tab

  Unicode property for case operations. Available as **Msoft_dotted** (p. 28) property.

- CASE-MAPPING.tab

  Unicode case mapping of each character that is available as **Mcase_mapping** (p. 28) property.

- BLOCKS.tab

  Unicode fallback script name for each character that is available as **Mblock** (p. 28) property. Generated manually by referring UCD Blocks.txt.

E.2 Input method

See **Input Method** (p. 219) for the format of these files.

- am-sera.mim (language:am name:sera)

```
Amharic input method with SERA.
For more information, see the page http://www.geez.org/IM/.
```

- ar-kbd.mim (language:ar name:kbd)

```
Input Method for Arabic simulating Arabic keyboard (MS Windows).
```

- as-itrans.mim (language:as name:itrans)

```
Assamese input method by ITRANS transliteration.
For the detail of ITRANS, see the page:
  <http://www.aczoom.com/itrans/>
```

- bn-itrans.mim (language:bn name:itrans)

```
Bengali input method by ITRANS transliteration.

Itrans Bengali Keymap Layout created by Avinash Chopde in
accordance with the details in the following link:

http://www.aczoom.com/itrans/beng/node4.html

Key Summary:

The consonant alphabets are represented as half-characters by
default i.e. k = <U+0995><U+09CD> . To complete the character please use ’a’
representing ’<U+0985>’ i.e. ka=<U+0995>. Consonant conjuncts can be created by
writing the consonant characters in sequential order. To complete
the conjunct either ’<U+0985>’ or any other dependent vowel [<U+0985> (a),
<U+09BE>(aa), <U+09BF>(i), <U+09C0>(ii), <U+09C1>(u), <U+09C2>(uu), <U+09C7>(e), <U+09C8>(ai), <U+09CB>
to be added at the end.

E.g. <U+0995><U+09CD><U+09B0><U+09BF><U+09DF><U+09BE> = k+r+i+Y+A

To write ’Khaanda-ta’ (<U+09CE>) use the key combination : t.h

Detailed instructions for typing are available at the above mentioned link

The following keysequences are not defined in the mentioned page,
but added for users’ sake:

Ch JN shh yh dny LLi L^i RRI R^I LLI L^I # $ ^ * ]
Shift-SPC Control-SPC
```

- bn-unijoy.mim (language:bn name:unijoy)

```
Bengali input method simulating Unijoy keyboard layout.
  <http://ekushey.org/projects/shadhinota/uni_joy.html>
```

- bo-ewts.mim (language:bo name:ewts)

```
Tibetan input method based on EWTS.
This implementation is based on THDL Extended Wylie Transliteration Scheme
Version 2.0 <http://www.thdl.org/collections/langling/ewts/ewts.php>.
```

- bo-tcrc.mim (language:bo name:tcrc)

```
Tibetan input method using the TCRC keyboard layout.
For more information, see the page:
  http://www.tibet.net/download/tcrckbd.rtf
```

- bo-wylie.mim (language:bo name:wylie)

```
Tibetan input method based on the Wylie transliteration.
It is actually the re-implementation of Emacs' tibetan-wylie input method,
and is slightly different from Extended Wylie Transliteration Scheme (EWTS).
The exact EWTS-based input method is in bo-ewts.mim.
```

- cjk-util.mim (extra-name:nil, only for inclusion)

```
Provide utilities for CJK input methods.
This is acutually not a standalone input method, but is expected
to be included in the other input method (e.g. zh-py).

The fullwidth mode is turned on by typing ">>", and turned off
by typing "<<".

The single fullwidth mode is turned on by typing "Z".  In this
mode, any key typed is converted to the fullwidth character and
is inserted, then the mode is turned off.
```

- cmc-kbd.mim (language:cmc name:kbd)

```
Cham input method simulating Cham keyboard.
Cham characters are encoded in logical order in memory and in files.
But, you can type Cham text in visual order with this input method.
Backspace and Delete also work in the manner of visual order.
```

- da-post.mim (language:da name:post)

```
Danish input method with postfix modifiers.
```

- dv-phonetic.mim (language:dv name:phonetic)

```
Dhivehi input method simulating the Dhivehi phonetic keyboard.
The layout is approved by the Molvidian Ministry of
Communication, Science and Technology.
  <http://www.mcst.gov.mv/News_and_Events/xpfonts.htm>
```

- el-kbd (language:el name:kbd)

  Input method for Greek simulating Greek keyboard.

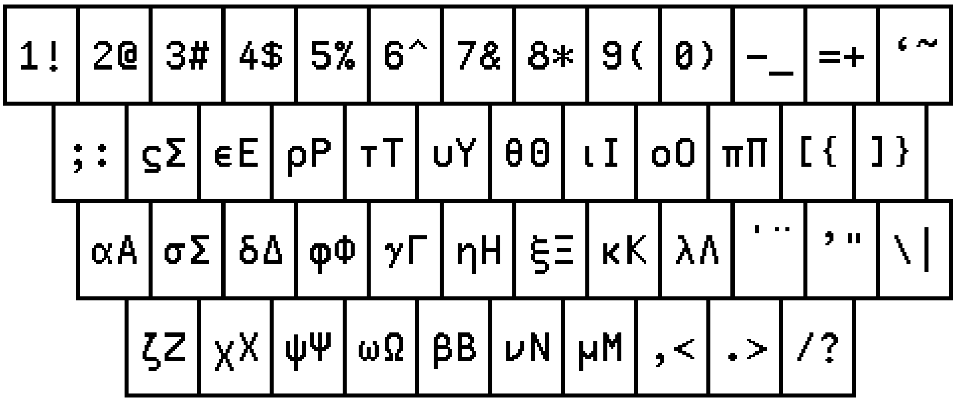

1! 2@ 3# 4$ 5% 6^ 7& 8* 9(0) -_ =+ ‘~
;: ςΣ ϵE ρP τT υY θΘ ιI oO πΠ [{]}
αA σΣ δΔ φΦ γΓ ηH ξΞ κK λΛ ´¨ ’" \|
ζZ χX ψΨ ωΩ βB νN μM ,< .> /?

Figure E.1: Keyboard Layout

- fa-isiri.mim (language:fa name:isiri)

```
Farsi input method simulating ISIRI 2901-1994 keyboard layout.
This is for typing Farsi by Arabic characters.
```

- fr-azerty.mim (language:fr name:azerty)

```
Simulating Azerty keyboard on English keyboard.

     &1  é2  "3  ’4  (5  -6  è7  _8  ç9  à0  )°  =_  ²~
      aA  zZ  eE  rR  tT  yY  uU  iI  oO  pP  ^¨  $£
       qQ  sS  dD  fF  gG  hH  jJ  kK  lL  mM  ù%  *|
        wW  xX  cC  vV  bB  nN  ,?  ;.  :/  !§

’[’ and ’{’ are used as a dead key to type a character with the
circumflex and diaeresis respectively (e.g. ’[’ ’e’ -> "ê").

’Alt-2’ and ’Alt-7’ are used as a dead key to type a character
with tilde and grave respectively (e.g. ’Alt-2’ ’n’ -> "ñ").

’Ctrl-Alt-2’ and ’Ctrl-Alt-7’ can be used as ’Alt-2’ and ’Alt-7’
respectively.

Azerty keyboard has one more key at the bottom left corner for
inputting "<" and ">".  As a normal English keyboard doesn’t
have such a key left, type ’<’ and ’>’ twice for "<" and ">"
respectively.
```

- global.mim (extra-name:nil, only for inclusion)

```
Global variable and command definitions.
This is actually not an input method, but provides documents,
default values of global variables, and default key-bindings of
global commands.
```

- grc-mizuochi.mim (language:grc name:mizuochi)

```
Mizuochi input method for classical Greek.

-------------------------------------
character     capital         small
-------------------------------------
alpha            A               a
beta             B               b
gamma            G               g
delta            D               d
epsilon          E               e
zeta             Z               z
eta              H               h
theta            Q               q
iota             I               i
kappa            K               k
lamda            L               l
mu               M               m
nu               H               n
xi               X               x
omicron          O               o
pi               P               p
rho              R               r
sigma            S               s
final sigma                      j
tau              T               t
upsilon          U               u
phi              F               f
chi              C               c
psi              Y               y
omega            W               w
```

```
-------------------------------------
sampi                           !
digamma         #
stigma                          $
koppa           &               %
-------------------------------------

------------------------
mark            key
------------------------
ypogegrammeni   J
psili           ’  or  v
dasia           ‘  or  V
oxia            /
varia           ?
perispomeni     \  or  ^
dialytika       "
ano teleia      :
erotimatiko     ;
----------------------
```

- gu-itrans.mim (language:gu name:itrans)

```
Gujarati input method by ITRANS transliteration.
For the detail of ITRANS, see the page:
  <http://www.aczoom.com/itrans/>
```

- he-kbd (language:he name:kbd)

  Input method for Hebrew simulating Hebrew keyboard.

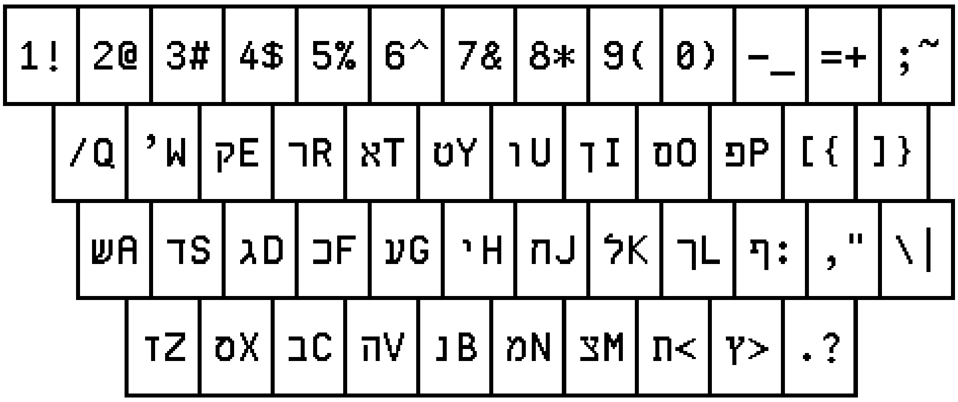

1! 2@ 3# 4$ 5% 6^ 7& 8* 9(0) -_ =+ ;~
/Q ’W קE רR אT טY וU ןI םO פP [{]}
שA דS גD כF עG יH חJ לK ךL ף: ," \|
זZ סX בC הV נB מN צM ת< ץ> .?

Figure E.2: Keyboard Layout

- hi-itrans.mim (language:hi name:itrans)

```
Hindi input method by ITRANS transliteration.
For the detail of ITRANS, see the page:
  <http://www.aczoom.com/itrans/>
```

- hi-typewriter.mim (language:hi name:typewriter)

```
Hindi input method with ‘typewriter’ method.
Still experimental.
```

- hr-kbd (language:hr name:kbd)

  Input method for Croatian.

Simulating Croatian Latin keyboard on American keyboard.

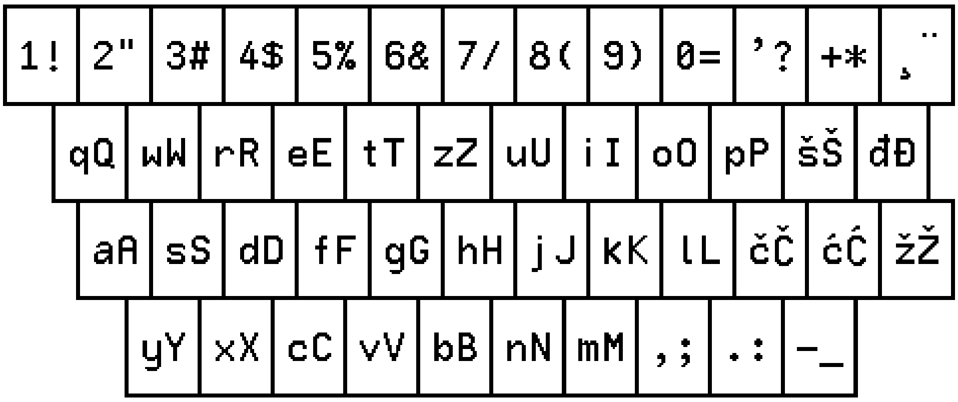

1! 2" 3# 4$ 5% 6& 7/ 8(9) 0= ’? +* ¸¨
qQ wW rR eE tT zZ uU iI oO pP šŠ đĐ
aA sS dD fF gG hH jJ kK lL čČ ćĆ žŽ
yY xX cC vV bB nN mM ,; .: -_

Figure E.3: Keyboard Layout

- hy-kbd (language:hy name:kbd)

  Input method for Armenian.

  Simulating Eastern Armenian keyboard on American keyboard.

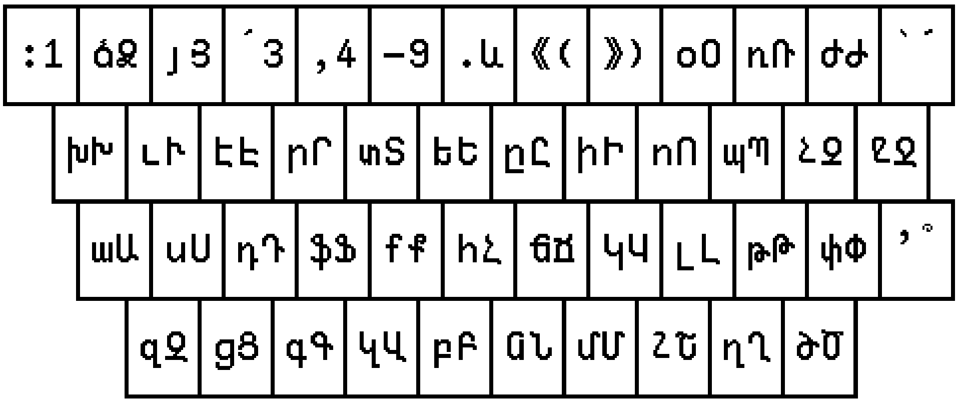

:1 ձՁ յՅ ՛3 ,4 -9 .և «(») օՕ ռՌ ժԺ ՝՜
խԽ ւՒ էԷ րՐ տՏ եԵ ըԸ իԻ ոՈ պՊ չՉ ջՋ
աԱ սՍ դԴ ֆՖ քՔ հՀ ճՃ կԿ լԼ թԹ փՓ ՚՞
զԶ ցՑ գԳ վՎ բԲ նՆ մՄ շՇ ղՂ ծԾ

Figure E.4: Keyboard Layout

- ispell.mim (language:en name:ispell)

```
Input method for English using ISPELL as a spell checker.
It uses the loadable module libmimx-ispell.so to communicate with
ISPELL program.  You can check the spelll of typed word by TAB
key.  Not for an actual use, but for demonstrating what can be
done by the m17n input method.
```

- ja-anthy.mim (language:ja name:anthy)

```
Japanese input method with Anthy as a kana-kanji converter.
Typed roma-ji is at first converted to Hiragana,
and Space key converts the Hiragana sequences
to Kanji-Hiragana mixed sequence.

This input method uses the loadable module libmimx-anthy.so to
communicate with Anthy.  For more detail about Anthy, see the page
  <http://sourceforge.jp/projects/anthy/>.
```

- ja-tcode.mim (language:ja name:tcode)

```
Input method for Japanese with TCODE.
```

- ja-trycode.mim (language:ja name:trycode)

```
Input method for Japanese with TRY-CODE.  See
<http://www.m17n.org/ntakahas/npx/aggressive/aggressive4.en.html>
for the details.
```

- ka-kbd (language:ka name:kbd)

  Input method for Georgian simulating Georgian keyboard.

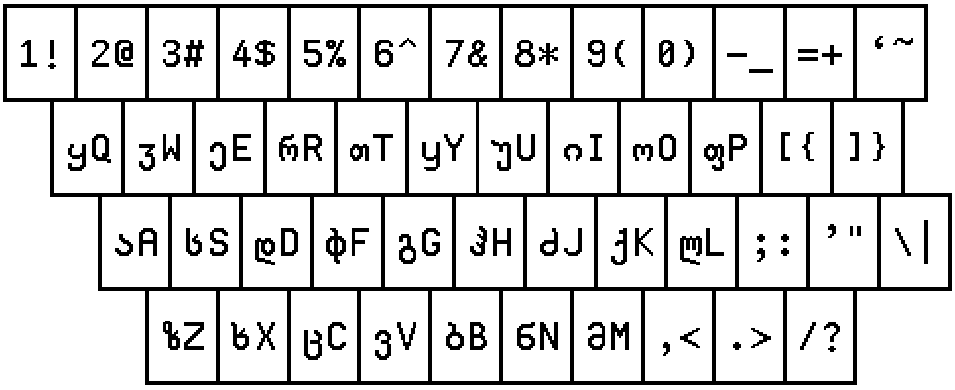
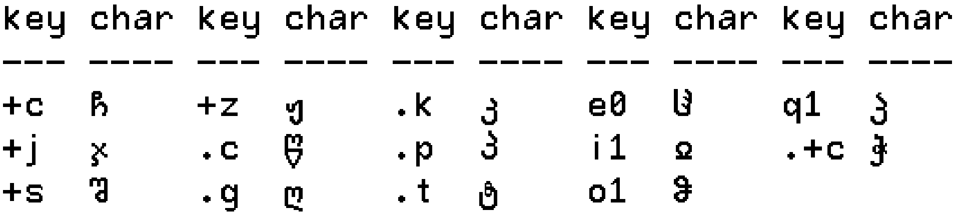


Figure E.5: Keyboard Layout

You can also input more characters by the following key sequences:

[type a key sequence to insert the corresponding character]



key	char	key	char	key	char	key	char	key	char
+c	ჩ	+z	ძ	.k	კ	e0	ჱ	q1	ჴ
+j	ჟ	.c	ჭ	.p	პ	i1	ჲ	.+c	ჳ
+s	შ	.g	ღ	.t	ტ	o1	ჵ		

Figure E.6: Extra Keys

- kk-arabic.mim (language:kk name:arabic)

```
Kazakh (with Arabic script) input method by transliteration.
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  a   A   b   v   g   R   d   e   j   z   y   k   q   l   m   n   N

 18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33
  o   O   p   r   s   t   w   u   U   f   H   h   c   S   I   i
```

- kk-kbd (language:kk name:kbd)

  Input method for Kazakh written in the Cyrillic script.

  Simulating Kazakh keyboard.

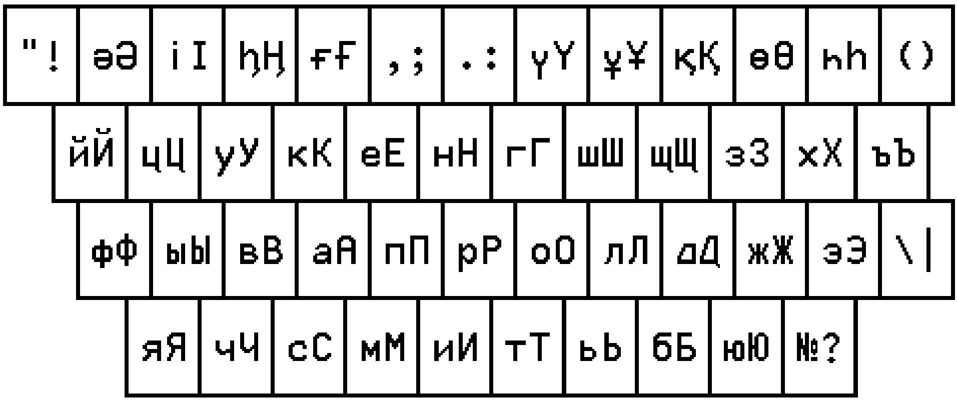

"! әӘ іІ ңҢ ғҒ ,; .: үҮ ұҰ қҚ өӨ һҺ ()
йЙ цЦ уУ кК еЕ нН гГ шШ щЩ зЗ хХ ъЪ
фФ ыЫ вВ аА пП рР оО лЛ дД жЖ эЭ \|
яЯ чЧ сС мМ иИ тТ ьЬ бБ юЮ №?

Figure E.7: Keyboard Layout

- km-yannis.mim (language:km name:yannis)

```
Khmer input method suggested by Dr. Yannis Haralambous.
```

- kn-itrans.mim (language:kn name:itrans)

```
Kannada input method by ITRANS transliteration.
For the detail of ITRANS, see the page:
  <http://www.aczoom.com/itrans/>
There are few changes from the ITRANS by Hari Prasad Nadig,
Kannada l10n Team, kannada.l10n@gmail.com
  <http://kannada.sourceforge.net>
on 18 Aug 2005.
```

- ko-han2 (language:ko name:han2)

  Hangul input method with 2-bul style.

  This input method uses this keyboard layout:

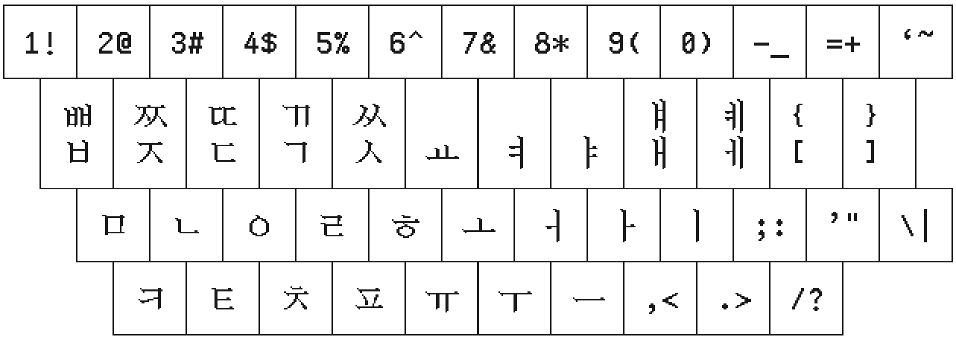


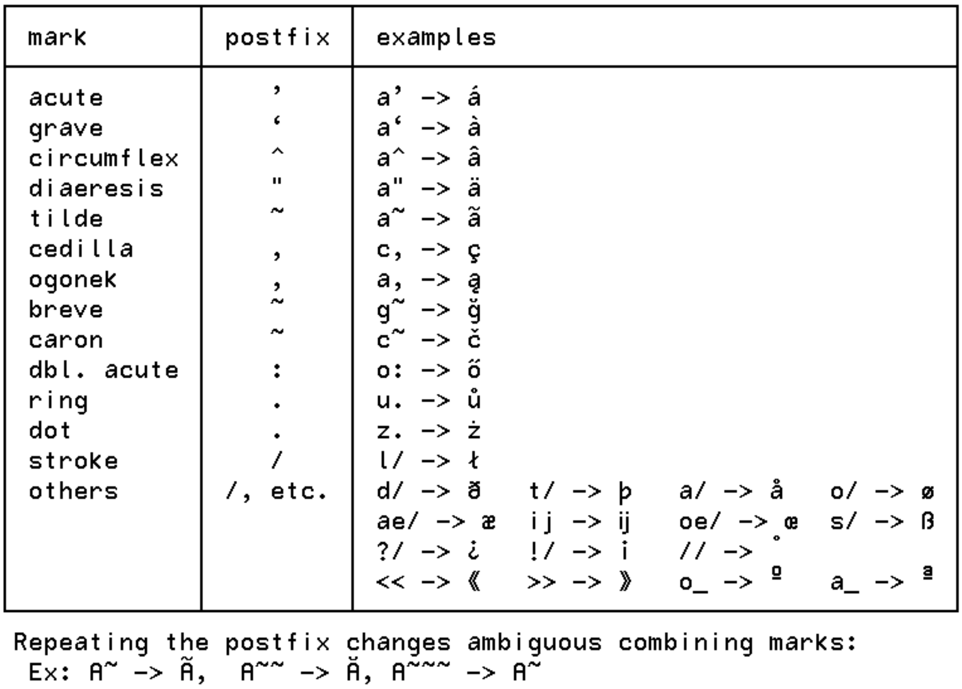
Figure E.8: Keyboard Layout

- ko-romaja.mim (language:ko name:romaja)

```
Hangul input method with romaja keys.
The roman-transliteration rules follows that of IIIMF shown in
the page <http://www3.sympatico.ca/d.moser/hangul.html>.

Common to CHOSEONG and JONGSEONG:
  <U+3131>(g) <U+3132>(gg,kk,qq,c) <U+3134>(n) <U+3137>(d) <U+3139>(l) <U+3139>(r) <U+3141>(m) <U+3142>(
  <U+3146>(ss) <U+3147>(ng) <U+3147>(x) <U+3148>(j) <U+314A>(ch) <U+314B>(k,q) <U+314C>(t) <U+314D>(p,f)

CHOSEONG:
  <U+3138>(dd,tt) <U+3143>(bb,vv) <U+3149>(jj)

JONGSEONG:
  <U+3133>(gs) <U+3135>(nj) <U+3136>(nh) <U+313A>(lg) <U+313B>(lm) <U+313C>(lb) <U+313D>(ls) <U+313E>(lt

JUNGSEONG:
  <U+314F>(a) <U+3150>(ai,ae) <U+3151> (ya,ia) <U+3152>(yai,yae,iae) <U+3153>(eo) <U+3154>(e,eoi) <U+315
  <U+3156>(ye,ie,yeoi) <U+3157>(o) <U+3158>(oa,wa,ua) <U+3159>(oai,wae,uae,oae) <U+315A>(oi,woe,uoe,oe)
  <U+315B>(yo,io) <U+315C>(u,w,oo) <U+315D>(ueo,wo,uo) <U+315E>(ue,we) <U+315F>(wi) <U+3160>(yu,iu) <U+3
  <U+3162>(eui,ui) <U+3163>(i,y,ee)

Special:
  Type uppercase letter to specify CHOSEONG explicitly.
  Type "I" to toggle the composed-syllable mode and isolated-jamo mode.
  Type ">>" to fullwidth ASCII letter mode, "<<" to shift out the mode.
  Type "Z" and a key to input fullwidth version of the key.
```

- latn-post (language:generic name:latn-post)

  Input method for Latin script with postfix modifiers.



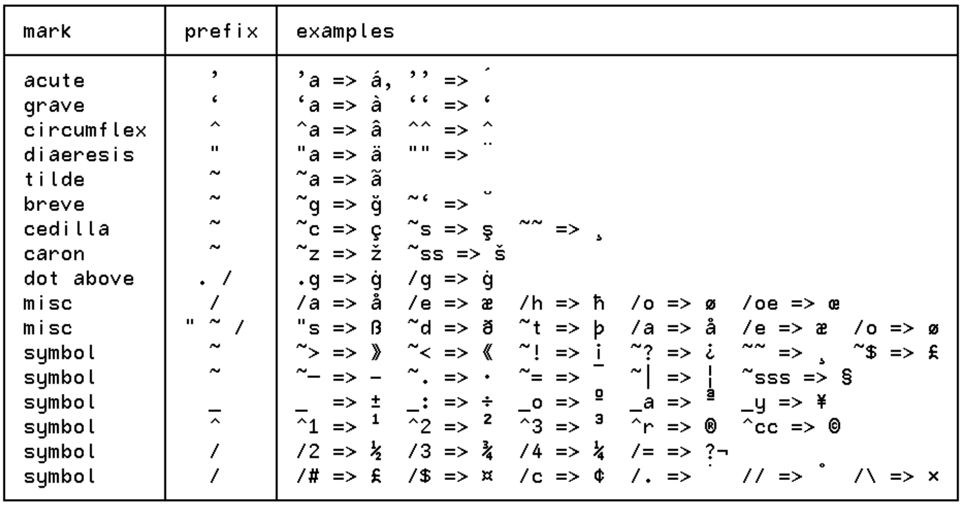
mark	postfix	examples
acute	’	a’ -> á
grave	‘	a‘ -> à
circumflex	^	a^ -> â
diaeresis	"	a" -> ä
tilde	~	a~ -> ã
cedilla	,	c, -> ç
ogonek	,	a, -> ą
breve	~	g~ -> ğ
caron	~	c~ -> č
dbl. acute	:	o: -> ő
ring	.	u. -> ů
dot	.	z. -> ż
stroke	/	l/ -> ł
others	/, etc.	d/ -> ð t/ -> þ a/ -> å o/ -> ø ae/ -> æ ij -> ĳ oe/ -> œ s/ -> ß ?/ -> ¿ !/ -> ¡ // -> ° << -> 《 >> -> 》 o_ -> º a_ -> ª

Repeating the postfix changes ambiguous combining marks:
Ex: A~ -> Ã, A~~ -> Ă, A~~~ -> A~

Figure E.9: Examples

- latn-pre (language:generic name:latn-pre)

  Input method for Latin script with prefix modifiers.



mark	prefix	examples
acute	’	’a => á, ’’ => ´
grave	‘	‘a => à ‘‘ => ‘
circumflex	^	^a => â ^^ => ^
diaeresis	"	"a => ä "" => ¨
tilde	~	~a => ã
breve	~	~g => ğ ~‘ => ˘
cedilla	~	~c => ç ~s => ş ~~ => ¸
caron	~	~z => ž ~ss => š
dot above	. /	.g => ġ /g => ġ
misc	/	/a => å /e => æ /h => ħ /o => ø /oe => œ
misc	" ~ /	"s => ß ~d => ð ~t => þ /a => å /e => æ /o => ø
symbol	~	~> => » ~< => « ~! => ¡ ~? => ¿ ~~ => ¸ ~$ => £
symbol	~	~- => - ~. => · ~= => ¯ ~| => ¦ ~sss => §
symbol	_	_ => ± _: => ÷ _o => º _a => ª _y => ¥
symbol	^	^1 => ¹ ^2 => ² ^3 => ³ ^r => ® ^cc => ©
symbol	/	/2 => ½ /3 => ¾ /4 => ¼ /= => ?¬
symbol	/	/# => £ /$ => ¤ /c => ¢ /. => ˙ // => ˚ /\ => ×

Figure E.10: Examples

- lo-kbd (language:lo name:kbd)

  Input method for Lao using Lao keyboard layout.

- lo-lrt.mim (language:lo name:lrt)

```
Lao input method using Lao-Roman transliteration.
```

- ml-itrans.mim (language:ml name:itrans)

```
Malayalam input method by ITRANS transliteration.
For the detail of ITRANS, see the page:
  <http://www.aczoom.com/itrans/>
```

- my-kbd.mim (language:my name:kbd)

```
Myanmar input method simulating the Myanmar keyboard.
```

- or-itrans.mim (language:or name:itrans)

```
Oriya input method by ITRANS transliteration.
For the detail of ITRANS, see the page:
  <http://www.aczoom.com/itrans/>
```

- pa-itrans.mim (language:pa name:itrans)

```
Panjabi input method by ITRANS transliteration.
For the detail of ITRANS, see the page:
  <http://www.aczoom.com/itrans/>
```

- rfc1345.mim (language:generic name:rfc1345)

```
Generic input method using RFC1345 mnemonics.
Input characters by typing & (ampersand) followed by two or three
keys.  It doesn’t include RFC1345 mnemonics for ASCII and
Control-1 characters (U+0000..U+009F) except for & itself which
can be input by typing & twice.
```

- ru-kbd (language:ru name:kbd)

  Input method for Russian by simulating the Russian keyboard.

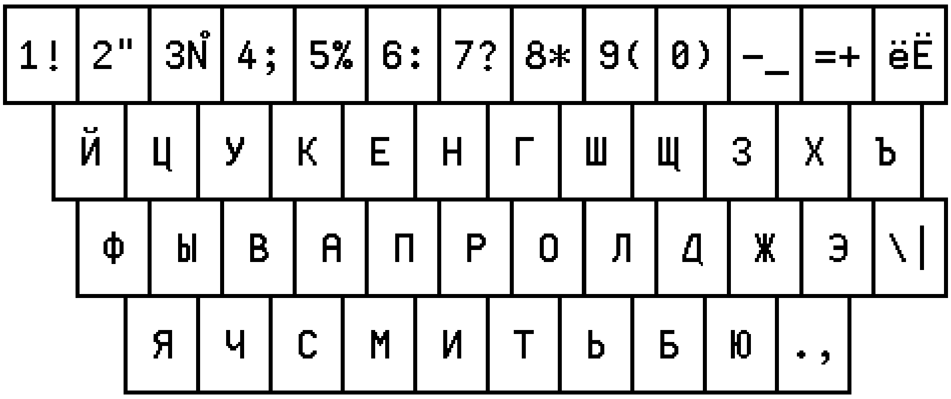


Figure E.11: Keyboard Layout

- ru-phonetic (language:ru name:phonetic)

  Input method for Russian simulating the keyboard layout based on

  Roman transcription by phonetic resemblance.

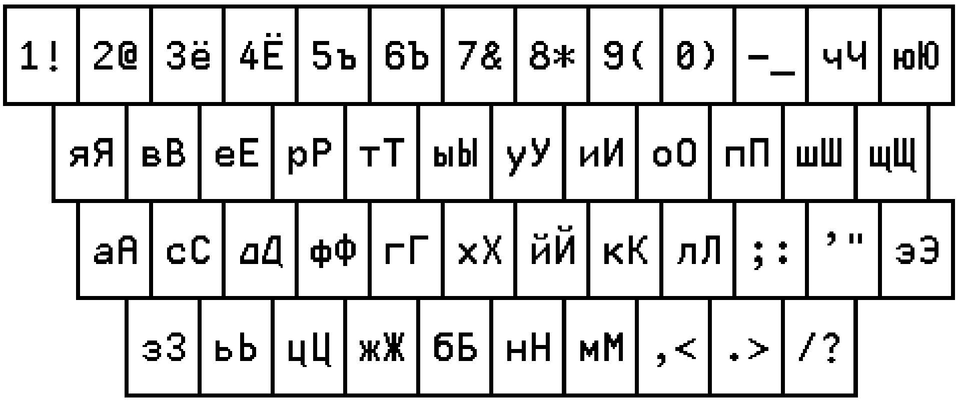

1!
2@
3ё
4Ё
5ъ
6Ъ
7&
8*
9(
0)
-_
чЧ
юЮ
яЯ вВ еЕ рР тТ ыЫ уУ иИ оО пП шШ щЩ
аА сС дД фФ гГ хХ йЙ кК лЛ ;: '"
эЭ
зЗ ьЬ цЦ жЖ бБ нН мМ ,< .> /?

Figure E.12: Keyboard Layout

- ru-yawerty (language:ru name:yawerty)

  Input method for Russian simulating the keyboard layout based on

  Roman transcription by phonetic resemblance.

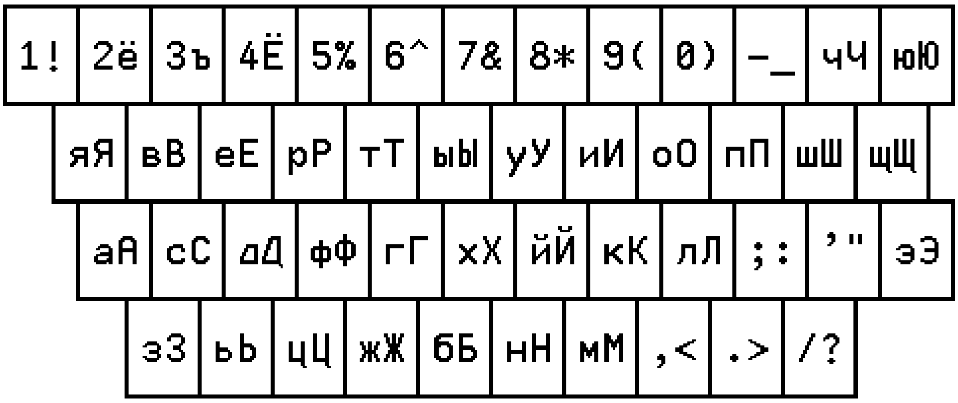
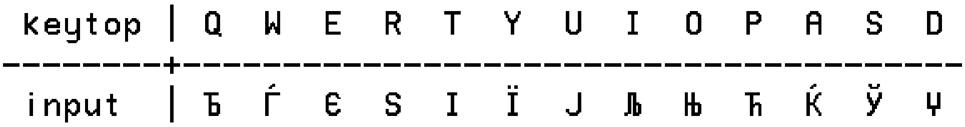

1! 2ё 3ъ 4Ё 5% 6^ 7& 8* 9(0) -_ чЧ юЮ
яЯ вВ еЕ рР тТ ыЫ уУ иИ оО пП шШ щЩ
аА сС дД фФ гГ хХ йЙ кК лЛ ;: '" эЭ
зЗ ьЬ цЦ жЖ бБ нН мМ ,< .> /?

Figure E.13: Keyboard Layout

When preceded by a '/', the second and the third rows (number key row) change as follows.



```
keytop | Q  W  E  R  T  Y  U  I  O  P  A  S  D
-------+-------------------------------------
input  | Ђ  Ѓ  Є  Ѕ  І  Ї  Ј  Љ  Њ  Ћ  Ќ  Ў  Џ
```

Figure E.14: Extra Keys

- sa-harvard-kyoto.mim (language:sa name:harvard-kyoto)

```
Sanscrit input method with Harvard-Kyoto convention.
The table is based on
  <http://en.wikipedia.org/wiki/Harvard-Kyoto>
```

- si-samanala.mim (language:si name:samanala)

```
Sinhala input method using transliteration.
The transleteration system is based on the Samanala version 2
developed by Prasad Dharmasena.
  <http://www.nongnu.org/sinhala/doc/transliteration/sinhala-transliteration_1.html>
```

- si-wijesekera-preedit.mim (language:si name:wijesekera-preedit)

```
Sinhala input method based on SLS 1134 Rev. 2:2004.
  <http://www.fonts.lk/doc/sin-kbd-layout4.pdf>
This input method uses preedit rather than surrounding text.
```

- si-wijesekera.mim (language:si name:wijesekera)

```
Sinhala input method based on SLS 1134 Rev. 2:2004.
  <http://www.fonts.lk/doc/sin-kbd-layout4.pdf>
This input method uses surrounding text if possible, or a preedit
buffer otherwise.
```

- sk-kbd (language:sk name:kbd)

  Input method for Slovak simulating the standard Slovak keyboard.

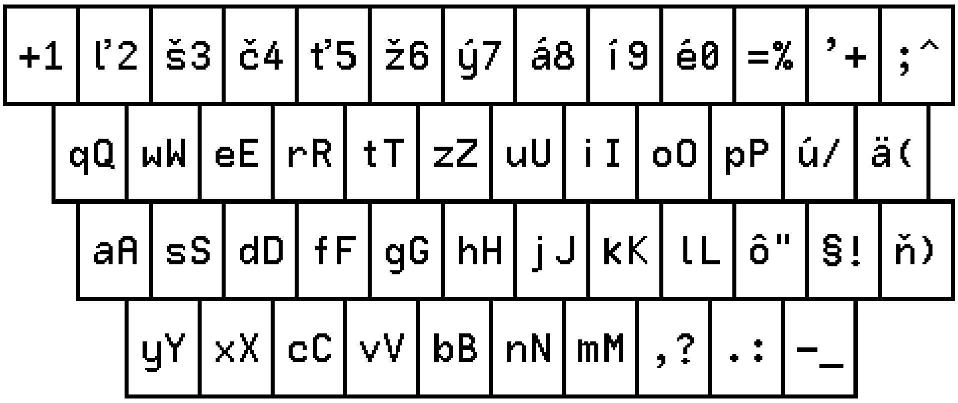

+1 ľ2 š3 č4 ť5 ž6 ý7 á8 í9 é0 =% ’+ ;^
qQ wW eE rR tT zZ uU iI oO pP ú/ ä(
aA sS dD fF gG hH jJ kK lL ô" §! ň)
yY xX cC vV bB nN mM ,? .: -_

Figure E.15: Keyboard Layout

You can also input more characters by the following key sequences:

```
key char key char key char key char key char key char
--- ---- --- ---- --- ---- --- ---- --- ---- --- ----
+C  Č    +L  Ľ    +S  Š    +Y  Ž    +r  ř    =R  Ŕ
+D  Ď    +N  Ň    +T  Ť    +d  ď    +u  ů    =l  ĺ
+E  Ě    +R  Ř    +U  Ů    +e  ě    =L  Ĺ    =r  ŕ
```

Figure E.16: Extra Keys

- sr-kbd (language:sr name:kbd)

  Input method for Serbian.

  Simulating Serbian Cyrillic keyboard on American keyboard.

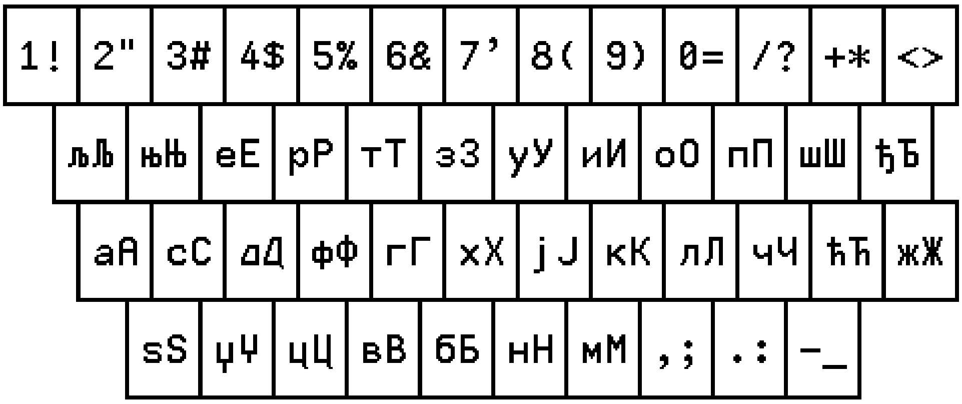

1! 2" 3# 4$ 5% 6& 7’ 8(9) 0= /? +* <>
љЉ њЊ еЕ рР тТ зЗ уУ иИ оО пП шШ ђЂ
аА сС дД фФ гГ хХ јЈ кК лЛ чЧ ћЋ жЖ
ѕЅ џЏ цЦ вВ бБ нН мМ ,; .: -_

Figure E.17: Keyboard Layout

- sv-post.mim (language:sv name:post)

  ```
  Swedish input method with postfix modifiers.
  ```

- syrc-phonetic.mim (language:generic name:syrc-phonetic)

```
Syriac input method simulating the Syriac phonetic keyboard.
The keyboard layout was published by Beth Mardutho: The Syriac Institute.
  <http://www.BethMardutho.org>
```

- ta-itrans.mim (language:ta name:itrans)

```
Tamil input method by ITRANS transliteration.
For the detail of ITRANS, see the page:
  <http://www.aczoom.com/itrans/>
```

- ta-lk-renganathan.mim (language:ta name:lk-renganathan)

```
Tamil input method with Renganathan layout.
For the detail, see the page: <http://www.locallanguages.lk/>
```

- te-itrans.mim (language:te name:itrans)

```
Telugu input method by ITRANS transliteration.
For the detail of ITRANS, see the page:
  <http://www.aczoom.com/itrans/>
```

- th-kesmanee.mim (language:th name:kesmanee)

```
Thai input method simulating the Kesmanee keyboard
with WTT 2.0 input sequence correction.
The correction algorithm follows the one shown in the following
  <http://linux.thai.net/~thep/th-xim/>
```

- th-pattachote.mim (language:th name:pattachote)

```
Thai input method simulating the Pattachote keyboard
with WTT 2.0 input sequence correction.
The correction algorithm follows the one shown in the following
  <http://linux.thai.net/~thep/th-xim/>
```

- th-tis820.mim (language:th name:tis820)

```
Thai input method simulating the TIS-820.2538 keyboard
with WTT 2.0 input sequence correction.
The correction algorithm follows the one shown in the following
  <http://linux.thai.net/~thep/th-xim/>
```

- ug-kbd.mim (language:ug name:kbd)

```
Uyghur input method simulating an Uyghur keyboard layout.
Based on <http://tarim.yulghun.com/docs/src/uyghur.xkb>
```

- unicode.mim (language:generic name:unicode)

```
Input method for Unicode BMP characters using hexadigits.
Type C-u followed by four hexadecimal numbers [0-9A-Fa-f]
of a Unicode character code.
```

- vi-base.mim (extra-name:nil, only for inclusion)

```
Provide bases for Vietnamese input methods.
This is acutually not a standalone input method, but is expected
to be included in the other Vietnamese input method (e.g. vi-telex, vi-vni).
```

- vi-tcvn.mim (language:vi name:tcvn)

```
Vietnames input method using the TCVN6064 sequence.
Typing Backslash ('\') toggles the normal mode and English mode.
The following variables are customizable:
  tone-mark-on-last: control tone mark position in equivocal cases
  backspace-is-undo: control the action of Backspace key (delete or undo)
```

- vi-telex.mim (language:vi name:telex)

```
Vietnames input method using the TELEX key sequence.
Typing Backslash ('\') toggles the normal mode and English mode.
The following variables are customizable:
  tone-mark-on-last: control tone mark position in equivocal cases
  backspace-is-undo: control the action of Backspace key (delete or undo)
```

- vi-viqr.mim (language:vi name:viqr)

```
Vietnames input method using the VIQR key sequence.
Typing Backslash ('\') toggles the normal mode and English mode.
The following variables are customizable:
  tone-mark-on-last: control tone mark position in equivocal cases
  backspace-is-undo: control the action of Backspace key (delete or undo)
```

- vi-vni.mim (language:vi name:vni)

```
Vietnames input method using the VNI key sequence.
Typing Backslash ('\') toggles the normal mode and English mode.
The following variables are customizable:
  tone-mark-on-last: control tone mark position in equivocal cases
  backspace-is-undo: control the action of Backspace key (delete or undo)
```

- zh-bopomofo (language:zh name:bopomofo)

  Input method for Bopomofo.

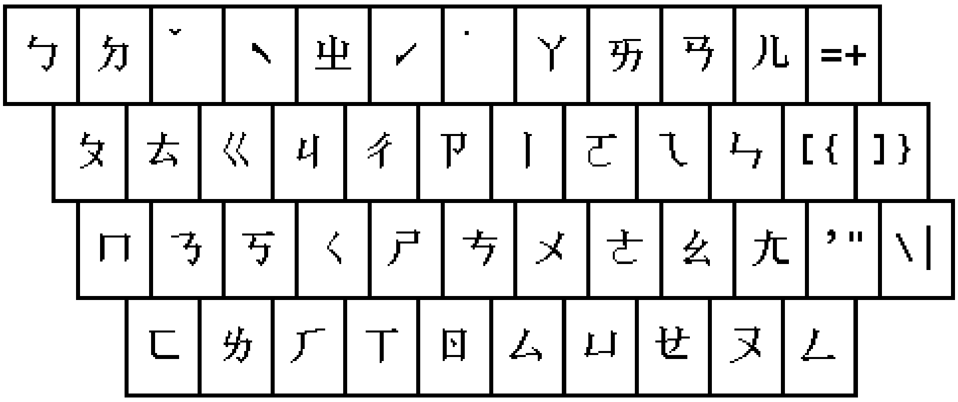


Figure E.18: Keyboard Layout

- zh-cangjie.mim (language:zh name:cangjie)

```
Chinese input method with CANGJIE method.
In addition to Chinese characters, fullwidth latin characters and
symbols are available in fullwidth mode (turns on and off by
">>" and "<<" respectively).  This mode can also be turned on
temporarily by typing "Z".
```

- zh-pinyin.mim (language:zh name:pinyin)

```
Input method for Chinese Pinyin characters.
Note that it's not for inputting Han characters.
```

- zh-py-b5.mim (language:zh name:py-b5)

```
Chinese Big5 input method with Pinyin sequence.
In addition to Chinese characters, fullwidth latin characters and
symbols are available in fullwidth mode (turns on and off by
">>" and "<<" respectively).  This mode can also be turned on
temporarily by typing "Z".
```

- zh-py-gb.mim (language:zh name:py-gb)

```
Chinese GB2312 input method with Pinyin sequence.
In addition to Chinese characters, fullwidth latin characters and
symbols are available in fullwidth mode (turns on and off by
">>" and "<<" respectively).  This mode can also be turned on
temporarily by typing "Z".
```

- zh-py.mim (language:zh name:py)

```
Chinese input method with Pinyin sequence.
In addition to Chinese characters, fullwidth latin characters and
symbols are available in fullwidth mode (turns on and off by
">>" and "<<" respectively).  This mode can also be turned on
temporarily by typing "Z".
```

- zh-quick.mim (language:zh name:quick)

```
Chinese input method with QUICK method.
In addition to Chinese characters, fullwidth latin characters and
symbols are available in fullwidth mode (turns on and off by
">>" and "<<" respectively).  This mode can also be turned on
temporarily by typing "Z".
```

- zh-tonepy-b5.mim (language:zh name:tonepy-b5)

```
Chinese Big5 input method with Pinyin+Tone sequence.
In addition to Chinese characters, fullwidth latin characters and
symbols are available in fullwidth mode (turns on and off by
">>" and "<<" respectively).  This mode can also be turned on
temporarily by typing "Z".
```

- zh-tonepy-gb.mim (language:zh name:tonepy-gb)

```
Chinese GB2312 input method with Pinyin+Tone sequence.
In addition to Chinese characters, fullwidth latin characters and
symbols are available in fullwidth mode (turns on and off by
">>" and "<<" respectively).  This mode can also be turned on
temporarily by typing "Z".
```

- zh-tonepy.mim (language:zh name:tonepy)

```
Chinese input method with Pinyin-and-tone sequence.
In addition to Chinese characters, fullwidth latin characters and
symbols are available in fullwidth mode (turns on and off by
">>" and "<<" respectively).  This mode can also be turned on
temporarily by typing "Z".
```

- zh-util.mim (extra-name:nil, only for inclusion)

```
Provide utilities for Chinese input methods.
This is acutually not a standalone input method, but is expected
to be included in the other Chinese input method (e.g. zh-py).
```

E.3 Font Layout Table

See **Font Layout Table** (p. 211) for the format of these files.

- ARAB-OTF-NO-GPOS.flt

  For Arabic OpenType fonts that don't have GPOS table to draw the Arabic script.
- ARAB-OTF.flt

  For Arabic OpenType fonts to draw the Arabic script.
- ARAB.flt

  For Arabic fonts of Unicode encoding to draw Arabic script.
- BENG-OTF.flt

  For Bengali OpenType fonts to draw the Bengali script.
- CHAM-GENERIC.flt

  For the Cham proportional fonts to draw Cham script.
- COMBINING.flt

  For combining diacritical marsk (U+0300..U+036F).
- DEVA-CDAC.flt For the font DVYG0ntt.ttf (developed by C-DAC, encoding is ISFOC) to draw Devanagari script.
- DEVA-OTF.flt

  For Devanagari OpenType fonts to draw the Devanagari script.
- GUJR-OTF.flt

  For Gujarati OpenType fonts to draw the Gujarati script.
- GURU-OTF.flt

  For Gurmukhi OpenType fonts to draw the Gurmukhi script.
- HEBR-FF.flt

  For Hebrew fonts of Unicode encoding to draw the Hebrew script. This is for such fonts that do not require an explicit combining code because accents and points have negative lbearing.
- HEBR.flt

  For Hebrew fonts of Unicode encoding to draw Hebrew script. This is for such a font that requires explicit combining code to draw accents and points.
- KHMR-ANLONG.flt

  For the font ANLONG.TTF to draw Khmer script. The font is available at:
  - infopage: `http://www.freelang.com/polices/index.html`
  - download: `http://www.freelang.com/download/fonts/ttf_khmer_anlong.zip`
- KHMR-OTF.flt

  For Khmer OpenType fonts to draw Khmer. A Font is available from `<http://www.khmeros.info/drupal/?q=en/download/fonts>`.
- KNDA-OTF.flt

  For Kannada OpenType fonts to draw the Kannada script.

- LAOO-ALICE.flt

  For the font ALICE0.TTF to draw Lao script. The font is available at:

  - infopage: `http://cg.scs.carleton.ca/~luc/laos.html`
  - download: `http://sources.asie.free.fr/aide/polices/ALICE0.TTF`

- LAOO-GENERIC.flt
- LAOO-MULE.flt

  For Lao fonts of mule encoding to draw Lao script. The font is available at:

  - infopage: `http://www.gnu.org/directory/localization/intlfonts.html`
  - download: `ftp://ftp.gnu.org/pub/gnu/intlfonts/intlfonts-1.2.1.tar.gz`

- LAOO-OTF.flt
- MLYM-CDAC.flt
- MLYM-OTF.flt

  For Malayalam OpenType fonts to draw the reformed Malayalam script.

- MLYM-RACHANA.flt

  For the Rachana Malayalam fonts to draw the traditional Malayalam script. This fonts handles virtually all ligatures with the AKHN feature without character reordering.

- MYMR-MYAZEDI.flt

  For the Myanmar Zedi family fonts to draw Myanmar script.

  - download: `http://www.myazedi.com/downloads/MyaZedi_M17N.ttf`

- NO-CTL.flt

  This is to suppress Complex Text Layout for many scripts. This FLT can be used for fonts that have Unicode encoding. Even if a glyph in a font has zero width, the glyph is displayed as if it is a spacing glyph.

- ORYA-OTF.flt

  For Oriya OpenType fonts to draw the Oriya script.

- SINH-OTF.flt

  For Sinhala OpenType fonts to draw Sinhala. A Font is available from `<http://sinhala.linux.lk/>`.

- SYRC-OTF.flt

  For Syriac OpenType fonts to draw the Syriac script.

- TAML-CDAC.flt
- TAML-OTF.flt

  For Tamil OpenType fonts to draw the Tamil script.

- TELU-OTF.flt

  For Telugu OpenType fonts to draw the Telugu script.

- THAA-OTF.flt

  For Thaana OpenType fonts to draw the Thaana script.

- THAI-GENERIC.flt

  For the Thai proportional fonts to draw Thai script.

- THAI-NORASI.flt

  For the Thai Norasi family fonts to draw Thai script. The fonts are available at:

  - debian package: ttf-thai-tlwg

- THAI-OTF.flt

- THAI-TIS620.flt

  For fixed width fonts of TIS620 encoding to draw Thai script.

- TIBT-MTIB.flt

  For the Tibetan TrueType font developped by Dr. Tomabechi to draw Tibetan script. The font is available at:

  - donwload: `http://www.m17n.org/m17n-lib-download/mtib.ttf`

- TIBT-MULE.flt

  For the muletibetan font developped by Dr. Tomabechi to draw Tibetan script. The font is available at:

  - infopage: `http://www.gnu.org/directory/Localization/intlfonts.html`
  - download: `ftp://ftp.gnu.org/pub/gnu/intlfonts/intlfonts-1.2.1.tar.gz`

- TIBT-OTF.flt

  For TibetanMachineUniAlpha.ttf to draw Tibetan script. The font is available at:

  - debian package: ttf-tmuni

E.4 Fontset

See **Fontset** (p. 218) for the format of these files.

- default.fst

  The default fontset. It contains complehensive specifications about which font to use for which script. In addition to X fonts, it contains these freely available TrueType fonts.

  - FreeSans.ttf (family: FreeSans), FreeSerif.ttf (family: FreeSerif), FreeMono.ttf (family: FreeMono)
    - infopage: `http://www.gnu.org/directory/all/freefont.html`
    - download: `http://savannah.nongnu.org/download/freefont/freefont-ttf.tar.gz`
  - Riwaj.ttf (family: Riwaj; for Arabic)
    - infopage: `http://www.geocities.com/hifazatequran/font.htm`
    - download: `http://www.geocities.com/hifazatequran/Riwaj.zip`
  - PakType Naqsh 2.2.ttf (family: PakType Naqsh; for Arabic)
  - PakType Tehreer 1.2.tty (family: PakType Tehreer; for Arabic)
    - debian package: ttf-paktype
  - Cyberbase.ttf (family: Bitstream CyberBase; for Arabic)
    - download: `http://www.ffonts.net/Bitstream-CyberBase.font.download`
  - SyrCOMEdessa.otf (family: Estrangelo Edessa; for Syriac),
    SyrCOMJerusalem.otf (family: Serto Jerusalem; for Syriac),
    SyrCOMAdiabene.otf (family: East Syriac Adiabene; for Syriac)
    - infopage: `http://www.bethmardutho.org/meltho/`

    - download: `http://www.bethmardutho.org/support/meltho/download/`
- raghu.ttf (family: Raghindi; for Devanagari)
    - infopage: `http://www.nepali.info/nepali/help.asp`
    - download: `http://www.nepali.info/nepali/fonts/raghu.ttf`
- gargi.ttf (family: gargi; for Devanagari)
    - debian package: ttf-devanagari-fonts
- AksharYoginiNormal.ttf (family: Aksharyogini; for Devanagari)
    - infopage: `http://aksharyogini.sudhanwa.com/aksharyogini.html`
    - download: `http://aksharyogini.sudhanwa.com/AksharYoginiNormal.ttf`
- MuktiNarrow.ttf (family: mukti narrow; for Bengali)
    - debian package: ttf-bengali-fonts
- LikhanNormal.otf (family: likhan; for Bengali)
    - debian package: ttf-bengali-fonts
- Sagar0.6_GPL.ttf (family: sagar; for Bengali)
    - infopage: `http://ekushey.sourceforge.net/`
    - download: `http://prdownloads.sourceforge.net/ekushey/Sagar0.6_GPL.ttf?download`
- Saab.otf (family: Saab; for Gurmukhi)
    - infopage: `http://guca.sourceforge.net/typography/fonts/saab/index.shtml`
    - download: `http://prdownloads.sourceforge.net/guca/saab.0.91.otf?download`
- padmaa-Medium-0.5.ttf (family: padmaa; for Gujarati)
    - debian package: ttf-gujarati-fonts
- utkalm.ttf (family utkal; for Oriya)
    - infopage: `http://oriya.sarovar.org/docs/getting_started/node13.html`
    - download: `http://oriya.sarovar.org/download/utkalm.ttf.gz`
- akruti1.ttf, akruti1b.ttf (family akrutitml1; for Tamil),
  akruti2.ttf, akruti2b.ttf (family akrutitml2; for Tamil),
  TSCu_Comic.ttf (family tscu_comic; for Tamil),
  TSCu_Paranar.ttf, TSCu_paranarb.ttf, TSCu_paranari.ttf (family tscu_paranar; for Tamil),
  TSCu_Times.ttf (family tscu_times; for Tamil),
  TAMu_Kadampari.ttf (family tamu_kadambri; for Tamil),
  TAMu_Kalyani.ttf (family tamu_kalyani; for Tamil),
  TAMu_Maduram.ttf (family tamu_maduram; for Tamil)
    - infopage: `http://tamillinux.sourceforge.net/`
    - download: `http://sourceforge.net/projects/tamillinux/`
- Pothana2000.ttf (family: Pothana2000; for Telugu)
    - infopage: `http://www.kavya-nandanam.com/`
    - download: `http://www.kavya-nandanam.com/dload.htm`
- Kedage-[bint].ttf (family kedage; for Kannada),
  Malige-[bint].ttf (family mallige; for Kannada)
    - download: `http://brahmi.sourceforge.net/downloads.html`
- Sampige.ttf (family: Sampige; for Kannada)
    - infopage: `http://kannada.sourceforge.net/`

    - download: `http://brahmi.sourceforge.net/dl/Sampige.ttf`
  - Rachana_04.ttf (family: rachana; for Malayalam)
    - debian package: ttf-malayalam-fonts
  - THOOLIUC.TTF (family: thoolikaunicode; for Malayalam)
    - infopage: `http://www.supersoftweb.com/ThoolikaUnicode.aspx`
    - download: `http://www.supersoftweb.com/FreeDownloades/ThooliUC.TTF`
  - lklug.otf (family: LKLUG; for Sinhala)
    - debian package:: ttf-sinhala-lklug
  - TibetanMachineUniAlpha.ttf (family: tibetan machine uni; for Tibetan)
    - debian package: ttf-tmuni
  - Norasi_Bold.ttf, Norasi_BoldItalic.ttf, Norasi_Italic.ttf, Norasi.ttf (family Norasi; for Thai)
    - debian package: ttf-thai-tlwg
  - Phetsarath_OT.ttf (family: Phetsarath OT; for Lao)
    - infopage: `http://sourceforge.net/projects/laofoss`
    - download: `http://prdownloads.sourceforge.net/laofoss/Phetsarath_OT.zip`
  - KhmerOS.ttf (family: Khmer OS; for Khmer)
    - debian package: ttf-khmeros
  - MyaZedi_M17N.ttf (family: MyaZedi; for Myanmar)
    - download: `http://www.myazedi.com/downloads/MyaZedi_M17N.ttf`
  - BPG-Classic-99Um.ttf (family: BPG Classic 99U; for Georgian)
    - infopage: `http://bpg.sytes.net/bpgfonts/bpg_classic.htm`
    - download: `http://bpg.sytes.net/bpgfonts/files/BPG-Classic-U.zip`
  - kochi-gothic.ttf, kochi-gothic-subst.ttf (family: kochi gothic;for Japanese)
    - infopage: `http://sourceforge.jp/projects/efont/files/`
    - download: `http://prdownloads.sourceforge.jp/efont/4845/kochi-substitute-20030628.tar.bz`
  - dotum.ttf (family: Baekmuk Dotum; for Hangul)
    - debian package: ttf-baekmuk
  - gbsn00lp.ttf (family: AR PL SungtiL GB; for Chinese)
    - debian package: ttf-arphic-gbsn00lp
  - bsmi00lp.ttf (family AR PL Mingti2L Big5; for Chinese)
    - debian package: ttf-arphic-bsmi00lp

- xfont.fst

  Fontset using only X fonts.

- truetype.fst

  Fontset using only freely available TrueType fonts. See the documentation of the fontset "default" for the information about each font.

- generic.fst

  Fontset mainly using generic font specifications. See the documentation of the fontset "default" for the information about each font.

E.5 The other data

- FONTENC.tbl

  Information about encodings of fonts. See the section **Font Encoding** (p. 217).

- FONTSIZE.tbl

  Information about how much to resize fonts. See the section **Font Size** (p. 218).

- CHARSET.tbl

  List of charset definitions. See the section **List of character set definitions** (p. 210) for the format of this file.

- CODING.tbl

  List of coding system definitions. See the section **List of coding system definitions** (p. 210) for the format of this file.

- SCRIPT-OTF.tbl

  Table of scripts vs the corresponding OTF script tags.

- SCRIPT-LANGUAGE.tbl

  Table of scripts vs languages using the corresponding script.

- SCRIPT-LANGUAGE.tbl

  Table of scripts vs languages using the corresponding script.

Appendix F

Tutorial for writing the m17n database

This section contains tutorials for writing various database files of the m17n database.

- **TutorialIM** (p. 254) – Tutorial of input method

F.1 Tutorial of input method

F.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 **Input Method** (p. 219) 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
```

F.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 "İ" (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" "İ")
```

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, "İ" 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 "İ".

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" "İ"))
  (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.

F.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" "İ")))
(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 ?İ))

    ;; If the character before the inputting spot is A..Z,
    ;; a..z, or İ, 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 ?İ) "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 İ by the conditional below.

```
(cond ((| (& (>= @-2 ?A) (<= @-2 ?Z))
          (& (>= @-2 ?a) (<= @-2 ?z))
          (= @-2 ?İ))
```

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 "İ" must be changed into "i", so we need another conditional. The first branch

```
((= X ?İ) "i")
```

means that "if the character remembered in X is 'İ', '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 G

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:

Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index