## NAME

mjsu − introduction to the MJSulib API

## SYNOPSIS

#include "mjsu.h"

## DESCRIPTION

MJSulib is a library of utility functions for ANSI C programs. It consists of a compiled function-library (called **libmjsu.a** on UNIX systems or mjsu.lib on most others) and the header-file **mjsu.h**.

The library and header-file are written in standard ANSI C; ancient "K&R"-flavour C compilers will not be able to compile the library or any module that uses MJSulib facilities.

The header-file defines data-types, structures, symbolic constants and enumeration constants; and provides explicit full prototype declarations for the API functions. It should be explicitly #included by all C modules that use MJSulib facilities.

### Namespace

All symbols (functions, data-types, structures, enumeration and symbolic constants, macros) defined by MJSulib have canonical names prefixed with "**mjs_**". For convenience, and to make program code more readable, shorthand aliases are provided so that eg: **mjs_remark()** can be referred to as just **remark(3)**, or **mjs_BOOL** as just **BOOL**.

If a program using the library defines it's own symbols that collide with the MJSulib shorthand alias names, the aliases can be selectively turned off in the affected program modules by defining one or more of the following preprocessor symbols before #including **mjsu.h**:

| | |
|---|---|
| MJS_TYPES | hides the type-name aliases. |
| MJS_FUNCTIONS | hides the function-name aliases. |
| MJS_CONSTANTS | hides the alias names of constants. |

Although the shorthand aliases are implemented as C preprocessor macros, all the normal C semantics are preserved: you can still take the address of a function, type-casts behave identically, and there are no extra syntactic constraints. Significantly, the addresses of the canonical function and its' alias (eg: **mjs_remark**() and **remark**()) are identical and can be used interchangeably in any pointer comparisons, pointer assignments, etc.

In the remainder of this document, and in all the other MJSulib manpages, the short names are used.

## DATA TYPES

The library uses several abstract data-types. Most are simply aliases for standard C data types, but named to indicate specific intended usage.

Note than although CHAR, TINY (or UTINY) and BYTE may be the same size in many compilation environments, their names are used to indicate a specific usage, eg: UTINY represents small positive integer values, *not* character data, and vice versa.

### Numeric Data Types

The minimum and maximum values documented here for each type are not a specification of the actual size or format of data so declared: the actual compiler-imposed ranges may be larger. The ranges documented here, however, are the maximum that portable programs can rely on.

Even if your personal favourite C compiler allows you to store truly massive values in a USHORT, other C compilers do not. And if your hardware vendor is crazy enough to provide a machine that can address 9-bit or 10-bit bytes (eg: Honeywell DPS and DEC PDP-10), remember that other manufacturers are not so generous.

Thus you should always restrict your use of these types to the ranges or uses documented here.

| | |
|---|---|
| TINY | numeric integer values from -128 to 127 inclusive. |
| UTINY | numeric integer values from 0 to 255. |

| | |
|---|---|
| SHORT | numeric integer values from -32768 to 32767 inclusive. |
| USHORT | numeric integer values from 0 to 65535 inclusive. |
| LONG | numeric integer values from -2147483648 to 2147483647 inclusive. |
| ULONG | numeric integer values from 0 to 4294967295 inclusive. |
| INT | numeric integer, usually needed only to interface with other (less-portable) APIs. |
| UINT | numeric integer that can index even the largest declarable C array. Also often needed to interface with other (less-portable) APIs. |
| SINGLE | single-precision floating-point values. |
| DOUBLE | double-precision floating-point values. |

NOTE: SINGLE and DOUBLE do not have any exact portable definition of precision, range, representation, etc. - they are included simply for completeness, as aliases of the standard C types **float** and **double** respectively.

**Enumerated Data Types**

BOOL used to represent the boolean values YES, NO, TRUE or FALSE. These enumeration constants are encoded so that C conditional statements of the form:

```
if (shall_i == YES)
        do_it();
else
        dont_do_it();
```

behave as you would intuitively expect.

**Other Data Types**

| | |
|---|---|
| VOID | is used to represent "no data" or "data of unspecified type". |
| CHAR | is used to hold (8-bit) text character-codes. |
| BYTE | an octet (8 bits) of opaque data. |
| MBITMAP | represents an in-memory bitmap, see **mbm(4)**. |
| TAGMATCH | used in pattern-matching, see **amatch(3)**. |

## SYMBOLIC CONSTANTS

**Notification-Types**

are passed to any callback function registered with **notifier()** to allow such a function to distinguish which type of notification is to be displayed:

| | |
|---|---|
| INFO_NOTIFY | an informational message. |
| WARNING_NOTIFY | a warning message. |
| ERROR_NOTIFY | a fatal error message. |
| USAGE_NOTIFY | a fatal "usage" message, typically used to report command-line errors. |

**Deprecated**

Both of the following symbols are really private symbols, not part of the API proper. You should not rely on the existance of these symbols as they are likely to be removed in future versions.

| | |
|---|---|
| MBM_VERSION | the integer file-format version-number for MBITMAPs stored in external files, see **mbm(4)**. |
| MBM_MAGIC | the string which appears at the beginning of all MBITMAPs when stored in external files, see **mbm(4)**. |

## FUNCTIONS AND FILE-FORMATS

| | |
|---|---|
| amatch(3) | look for anchored match of regular expression |
| cpystr(3) | copy multiple strings |
| error(3) | print formatted error message and exit |
| getflags(3) | collect flag arguments from command-line |
| lstol(3) | convert 32-bit quantum from LSB-first into native byte-order |
| lstos(3) | convert 16-bit quantum from LSB-first into native byte-order |
| ltols(3) | convert 32-bit quantum from native into LSB-first byte-order |
| match(3) | match a regular expression |
| mbm(5) | format of MBITMAP files and structures |
| mbm_bits(3) | bitmap manipulations |
| mbm_load(3) | load a bitmap from a file |
| mbm_mem(3) | allocate or deallocate bitmaps on the heap |
| mbm_save(3) | save a bitmap to a file |
| mbm_size(3) | return the height or width of a bitmap |
| mem_buy(3) | (re)allocate storage space on the heap, optionally failing hard |
| mem_free(3) | deallocate storage space on the heap, returning a sentinel |
| notifier(3) | override the display-method for notifications produced by **error(3)**, **warning(3)**, **remark(3)** and **usage(3)**. |
| pattern(3) | build a regular expression pattern |
| remark(3) | print formatted notification message |
| rtw_short_hash(3) | generate unsigned-short hash value for an arbitrary key |
| rtw_long_hash(3) | generate unsigned-long hash value for an arbitrary key |
| stols(3) | convert 16-bit quantum from native into LSB-first byte-order |
| strhash(3) | generate unsigned-short hash value for a string |
| str_dup(3) | duplicate a string onto the heap |
| strtos(3) | convert numeric string to short integer, with range checking |
| usage(3) | print formatted usage message and exit |
| vec_buy(3) | create or extend a vector of strings |
| vec_dup(3) | duplicate a vector of strings |
| vec_free(3) | deallocate a vector of strings |
| warning(3) | print formatted notification message |
| whatami(3) | return the name of the calling process |

## AVAILABILITY

MJSulib is written in C, conforming to ANSI X3.159-1989 (hosted program environment).

Any platform providing a standard ANSI C compilation environment and execution environment(s) should be able to compile and use MJSulib.

However, the automatic makefile-generator only works on Windows-NT, UNIX or similar systems. MJSulib has been built for some other platforms, using hand-constructed makefiles.