

MAC User's Guide – Summary (Version 3.04, September 2026)

Overview and Purpose

MAC (MACro Preprocessor) is a **lightweight, interpreted macro preprocessor** designed for **preparing and manipulating ASCII data files**. Unlike C preprocessors, MAC operates **sequentially**, processing input files **line by line** without recursive expansion. It is optimized for **flexibility, interactivity, and integration with UNIX pipelines**, making it ideal for **data preprocessing, configuration file generation, and text templating**.

Key Features

- ✓ **Sequential Processing:** No recursive macro expansion (avoids loops).
 - ✓ **UNIX-Compatible:** Works with pipes (|) and standard I/O (stdin, stdout, stderr).
 - ✓ **C-Like Syntax:** Familiar directives (e.g., `#define`, `#ifdef`) with **extensions** (e.g., arithmetic, indirection, lists).
 - ✓ **Interactive Mode:** Supports **user prompts** (`#ask`) and **debugging** (`-v[erbose]`).
 - ✓ **Mathematical Functions:** Evaluates expressions with **trigonometric, logarithmic, and rounding functions**.
 - ✓ **Text Manipulation:** Handles **quotes, concatenation, and protection** (backticks, single/double quotes).
 - ✓ **Portable:** Written in **ANSI-C**, runs on 64-bit Windows.
-

Core Concepts

1. Terminology

Term	Definition
Macro	Identifier (NAM) replaced by its definition (stored as a string).
Directive	Command starting with # (e.g., #define , #if). Modifies MAC's behavior.
Token	Printable ASCII text (e.g., <code>foo</code> , <code>123</code> , <code>\$VAR</code>). Spaces/symbols require single quotes (<code>'</code>).
Data Line	Non-directive line; processed according to active macros.
Block	Group of lines between #if/#endif .
List	Comma-separated tokens declared via #list .
Indirection	Access macro definitions dynamically using @ (e.g., @MACRO , @@MACRO).

2. Processing Flow

1. **Input:** Files or `stdin` (last file first if multiple).
2. **Directive Handling:** Lines starting with **#** are **executed immediately**.
3. **Data Processing:** Non-directive lines are **scanned for macros** and replaced.
4. **Output:** Result sent to `stdout`; errors/warnings to `stderr`.

⚠ **Critical Rule:** MAC **does not recursively expand macros**. If A is defined as B and B as 123, $A \rightarrow B$ (not 123).

Command Line Syntax

MAC [option..] [pathname..]

- **Options and filenames** can be mixed; processed **left-to-right**.
 - **Files** are stacked (last in, first processed).
 - **stdin** is processed **after** all files (if **-s** is enabled).
-

📖 Command Line Options

Option	Description	Example
-u[sage]	Print usage to stdout.	MAC -u
-b[uilt]	Print version/build info to stdout.	MAC -b
-h[elp]	Print help to stdout.	MAC -h > help.txt
-s[tdin]	Enable stdin as input (for pipes/keyboard).	cat file.txt MAC -s
-e[cho] file	Record macro directives from stdin to file.	MAC -s -e session.log
-d[efine] N V	Define macro N as V (equivalent to #define N V).	MAC -d PI 3.14159
-i[nt] N E	Define N as integer result of expression E (e.g., #int N 1+2).	MAC -i COUNT 100
-r[eal] N E	Define N as scientific notation (default: 6 decimals).	MAC -r FREQ 1.0e6
-f[ix] N E	Define N as fixed-point (default: 2 decimals).	MAC -f PRICE 19.99
-v[erbose]	Enable detailed trace to stderr.	MAC -v file.mac > out.txt
-q[uiet]	Suppress warning messages .	MAC -q file.mac
-l[ine]	Keep output and input line numbering equal .	MAC -l file.mac
-p[refix] P	Only expand macros starting with prefix P (speeds up processing).	MAC -p 'CFG_' file.mac
-c[omment]	Trace active comments (##) to stderr.	MAC -c file.mac

✅ **Tip:** Use **single quotes** (') for values with spaces/symbols (e.g., -d GREETING 'Hello World!').

🔧 Predefined Macros

Macro	Description	Example Output
\$N	Empty string.	" "
\$V	MAC version (e.g., D0S V3.04).	D0S V3.04
\$D	Current date/time at start.	Fri Oct 09 09:11:13 2026
\$F	Current filename being processed.	config.mac
\$T	Current file tag (default: \$F).	Custom Tag (via #tag)

Macro Directives

Directives **must start with #** and be the **first word** on the line.

Macro Definition & Manipulation

Directive	Description	Example
<code>#define N [V]</code>	Define N as V (no evaluation). If V is omitted, N becomes empty.	<code>#define PI 3.14159</code>
<code>#default N [V]</code>	Define N as V only if N is undefined .	<code>#default DEBUG 0</code>
<code>#undef N [N2...]</code>	Remove macros N, N2, etc.	<code>#undef TEMP A B</code>
<code>#int N [E]</code>	Define N as integer result of expression E.	<code>#int COUNT 100 + 50</code>
<code>#real [D] N [E]</code>	Define N as scientific notation (e.g., 1.23e+05) with D decimals (default: 6).	<code>#real6 FREQ 1.0e6</code>
<code>#fix [D] N [E]</code>	Define N as fixed-point (e.g., 123.45) with D decimals (default: 2).	<code>#fix2 PRICE 19.99</code>
<code>#cat N V1 [V2...]</code>	Concatenate V1, V2, etc., to N's definition.	<code>#cat PATH '/usr/' 'bin' → /usr/bin</code>
<code>#freeze N [N2...]</code>	Protect N from further processing (wraps definition in backticks `).	<code>#freeze VERSION</code>
<code>#melt N [N2...]</code>	Unprotect N (removes backticks).	<code>#melt VERSION</code>
<code>#list N</code>	Treat N as a comma-separated list ; creates N[0], N[1], etc.	<code>#list COLORS red,green,blue</code>

Conditional Processing

Directive	Description	Example
<code>#if E</code>	Execute block if E (evaluated as integer) $\neq 0$.	<code>#if (X > 10)</code>
<code>#ifdef N [V]</code>	Execute block if N is defined (and optionally equals V).	<code>#ifdef DEBUG</code>
<code>#ifndef N [V]</code>	Execute block if N is not defined (or $\neq V$).	<code>#ifndef RELEASE</code>
<code>#else [C]</code>	Alternate block for <code>#if</code> / <code>#ifdef</code> . C is ignored.	<code>#else</code>
<code>#endif [C]</code>	End of conditional block. C is ignored.	<code>#endif</code>

⚠ Rules:

- Blocks **must be nested properly** (no cross-file `#if/#endif`).
- Expressions in `#if` are **evaluated as integers** (non-zero = true).

💡 Interaction & Control

Directive	Description	Example
<code>#ask N [M]</code>	If N is undefined , prompt user for input (message M).	<code>#ask NAME 'Enter your name:'</code>
<code>#msg [M]</code>	Print M to stderr.	<code>#msg Error: Invalid value!</code>
<code>#exit [M]</code>	Exit current file ; continue with next file. M is printed to stderr.	<code>#exit File processing complete</code>
<code>#quit [M]</code>	Terminate MAC normally . M is printed to stderr.	<code>#quit Done!</code>
<code>#abort [M]</code>	Abort MAC abnormally . M is printed to stderr.	<code>#abort Critical error!</code>
<code>#stop [M]</code>	Force immediate termination (ignores all conditions).	<code>#stop Emergency stop</code>
<code>#skip [E]</code>	Toggle skip mode (ON if E ≠ 0). Skipped lines are ignored .	<code>#skip 1</code>
<code>#verbose [E]</code>	Toggle verbose mode (ON if E ≠ 0). Traces processing to stderr.	<code>#verbose 1</code>
<code>#macro [N...]</code>	List all macros (or N...) to stderr.	<code>#macro</code>
<code>#status [M]</code>	Print MAC's internal status (e.g., active prefixes) to stderr.	<code>#status 'Debug Info'</code>

📄 File Operations

Directive	Description	Example
<code>#include F</code>	Process file F (macros expanded); pathname between single quotes. If F is a macro, use its definition.	<code>#include 'header.mac'</code>
<code>#insert F</code>	Copy file F verbatim (no macro expansion); pathname between single quotes.	<code>#insert 'license.txt'</code>
<code>#tag T</code>	Set file tag T for error messages (default: filename).	<code>#tag 'Config File'</code>

Comments

Directive	Description	Example
<code>## [M]</code>	Active comment: Traced to stderr if <code>-c[omment]</code> is enabled.	<code>## This is a debug note</code>
<code>#* [M]</code>	Passive comment: Always ignored.	<code>#* Internal documentation</code>

Functions and Expressions

MAC evaluates **arithmetic expressions** using **double-precision floating-point**.

Supported Operators

Category	Operators
Arithmetic	<code>+</code> <code>-</code> <code>*</code> <code>/</code> <code>%</code> <code>^</code> (exponentiation)
Comparison	<code>></code> <code><</code> <code>>=</code> <code><=</code> <code>==</code> <code>!=</code>
Logical	<code>!</code> (NOT), <code> </code> (OR), <code>&&</code> (AND)
Grouping	<code>()</code> (use liberally to avoid ambiguity!)

⚠ **Note:** MAC's parser **does not follow C's operator precedence** for `^` (exponentiation). Use parentheses!

Mathematical Functions

Functions	Description
<code>(sin(x))</code> <code>(cos(x))</code> <code>(tan(x))</code> <code>(asin(x))</code> <code>(acos(x))</code> <code>(atan(x))</code>	Trigonometric functions
<code>(exp(x))</code> <code>(log(x))</code> <code>(log10(x))</code>	Logarithmic/exponential
<code>(sinh(x))</code> <code>(cosh(x))</code> <code>(tanh(x))</code>	Hyperbolic functions
<code>(floor(x))</code> <code>(ceil(x))</code> <code>(round(x))</code>	Rounding/floor/ceil
<code>(abs(x))</code> <code>(sign(x))</code>	Absolute value/sign

⚠ **Warning:** Monadic functions **must be enclosed in parentheses** (e.g., `(sin(0.5))`).

Text Handling

Syntax	Behavior in Directives	Behavior in Data Lines
'text'	Field delimiter (quotes removed).	No effect (quotes remain).
`text`	Unprocessed text (quotes removed).	Unprocessed text (quotes removed).
"text"	No effect (quotes remain).	No effect (quotes remain).

Error Handling

- **Errors:** Cause **immediate termination** with a message to `stderr` (exit code 1). Examples:

```
line[2]  [e1.mac]
MAC error - unmatched parenthesis in expression <(1 > 0>
```

```
line[5]  [w.mac]
MAC error - unbalanced if/endif
```

- **Warnings:** Can be suppressed with `-q[uiet]`. Examples:
 - Redefining an existing macro (MAC warning - macro <A> was already defined).
 - Undefined macro in `#undef` (MAC warning - attempt to remove an undefined macro).

Debugging Tools:

- **-v[erbose]:** Trace every step.
 - **#macro:** List all macros.
 - **#status:** Show internal state.
 - **Active comments (##):** Log custom messages.
-

Strengths and Limitations

✓ Strengths

- **Simple & Fast:** No recursion → predictable performance.
- **Flexible:** Supports **arithmetic, conditionals, indirection, and user input**.
- **UNIX-Friendly:** Works with **pipes, redirections, and scripts**.
- **Debuggable:** **Verbose mode, status checks, and active comments**.
- **Portable:** Runs on **64-bit Windows**.

✗ Limitations

- **No Recursion:** Macros are **not expanded recursively** (e.g., A→B→123 stops at B).
- **Basic Math:** No complex data structures (e.g., arrays, dictionaries).
- **No String Functions:** Limited text manipulation (e.g., no `substr`, `strlen`).
- **Parser Quirks:** **Operator precedence** differs from C (use parentheses!).
- **Memory Limits:** Max **1024-character tokens** and **1000 macros**.

Conclusion

MAC is a versatile, lightweight preprocessor for text and data manipulation, bridging the gap between **static configuration files** and **dynamic templating**. Its **C-like syntax, mathematical capabilities, and interactive features** make it a powerful tool for:

- **Generating configuration files** (e.g., for NAPA, Makefiles).
- **Preprocessing data** (e.g., CSV, INI files).
- **Automating repetitive text tasks** (e.g., code generation, reports).

By leveraging **macros, conditionals, and file inclusion**, users can create **maintainable, modular, and reusable** text-processing workflows. For advanced use, combine MAC with **UNIX pipelines** or **scripts** for seamless integration into larger systems.

 **Further Reading:** Refer to the full *MAC User's Guide* (v3.04) for detailed directive syntax, error messages, and advanced examples.