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., `foo`, `123`, `$VAR`). Spaces/symbols require **single quotes** (`'`).
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` → `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., `DOS V3.04`).	`DOS 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
`#define N [V]`	Define `N` as `V` (no evaluation). If `V` is omitted, `N` becomes empty.	`#define PI 3.14159`
`#default N [V]`	Define `N` as `V` **only if `N` is undefined**.	`#default DEBUG 0`
`#undef N [N2...]`	Remove macros `N`, `N2`, etc.	`#undef TEMP A B`
`#int N [E]`	Define `N` as **integer** result of expression `E`.	`#int COUNT 100 + 50`
`#real [D] N [E]`	Define `N` as **scientific notation** (e.g., `1.23e+05`) with `D` decimals (default: 6).	`#real6 FREQ 1.0e6`
`#fix [D] N [E]`	Define `N` as **fixed-point** (e.g., `123.45`) with `D` decimals (default: 2).	`#fix2 PRICE 19.99`
`#cat N V1 [V2...]`	**Concatenate** `V1`, `V2`, etc., to `N`'s definition.	`#cat PATH '/usr/' 'bin'` → `/usr/bin`
`#freeze N [N2...]`	**Protect** `N` from further processing (wraps definition in backticks `` ` ``).	`#freeze VERSION`
`#melt N [N2...]`	**Unprotect** `N` (removes backticks).	`#melt VERSION`
`#list N`	Treat `N` as a **comma-separated list**; creates `N[0]`, `N[1]`, etc.	`#list COLORS red,green,blue`

❓ Conditional Processing

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

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

File Operations

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

💬 Comments

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

🔢 Functions and Expressions

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

📐 Supported Operators

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

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

📊 Mathematical Functions

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

🔍 **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**: lines of at most **1022 characters** (a longer line is an error) and at most **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**.