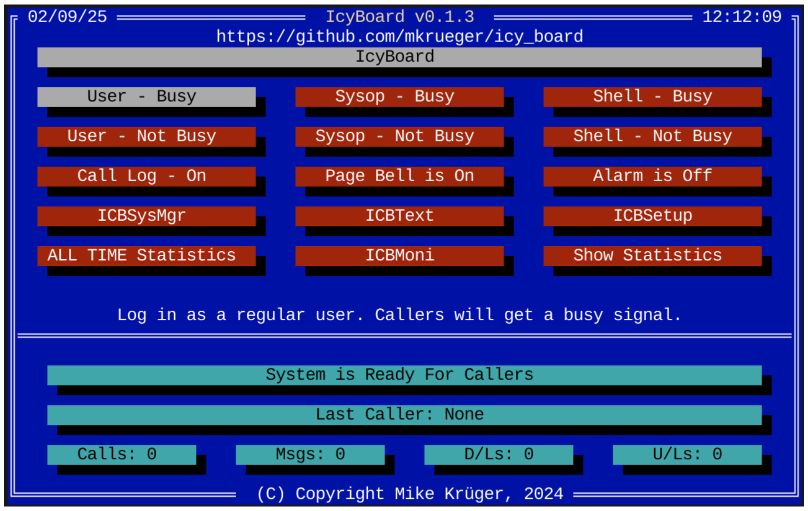
Icy Board

Mike Krüger

Oct 09, 2025

CONTENTS:



```
02/09/25 ════════════════ IcyBoard v0.1.3 ════════════════ 12:12:09
              https://github.com/mkrueger/icy_board
                            IcyBoard

    User - Busy            Sysop - Busy           Shell - Busy
    User - Not Busy        Sysop - Not Busy       Shell - Not Busy
    Call Log - On          Page Bell is On        Alarm is Off
    ICBSysMgr              ICBText                ICBSetup
    ALL TIME Statistics    ICBMoni                Show Statistics

          Log in as a regular user. Callers will get a busy signal.

                    System is Ready For Callers
                       Last Caller: None
    Calls: 0          Msgs: 0          D/Ls: 0          U/Ls: 0
════════════════ (C) Copyright Mike Krüger, 2024 ════════════════
```

Icy Board is a modern, memory-safe re-implementation of the classic PCBoard Bulletin Board System.

Written in Rust—aiming to preserve the original experience while enabling secure, scriptable expansion on today's platforms (Linux, macOS, Windows, ARM, etc.).

Unlike emulation layers that just "run the old EXE", Icy Board rebuilds core subsystems: user base, conferences, message storage (JAM), time/byte bank, accounting scaffolding, PPL execution, and TUI administration—providing a foundation that is both compatible *and* extensible.

CHAPTER

ONE

ICY BOARD

Due to its inheritance of PCBoard's architecture, Icy Board aims to provide a familiar experience for long-time users while introducing modern features and improvements.

1.1 Key Goals

- High compatibility with PCBoard 15.4 behaviors and PPE ecosystem
- Safe modernization: UTF-8, Internet protocols (Telnet, SSH, WebSockets)
- Preserve PCB / ANSI / Avatar / RIP aesthetics (nostalgia intact)
- Provide a fully featured PPL toolchain (compile, decompile, LSP)
- Make migration of legacy installations feasible
- Extend with new objects / APIs *without* breaking old PPE plugins

1.2 Non-Goals (by design)

- **"One-click shiny" out-of-box board—historical complexity retained**
  - PCBoard was never simple; neither is IcyBoard
- Heavy GUI configuration (focus is terminal / SSH TUIs)
- Running on DOS / Windows 9x / OS/2

1.3 License

Dual-licensed (Apache 2.0 / MIT) — see repository LICENSE files.

1.4 Building

Prerequisites:

- Rust toolchain (stable) — https://www.rust-lang.org/tools/install
- A UTF-8 capable terminal (most modern terminals)
- (Optional) VS Code for PPL editing

Build everything:

```
git clone https://github.com/mkrueger/icy_board.git
cd icy_board
cargo build --release
```

This will create a target/release/ directory with all executables.

1.5 Getting started

I recommend putting the bin/ directory in the path but you can just *cd bin* for now.

First create a new BBS: *./icbsetup create FOO* Then start it: *./icboard FOO*

This will fire up a new call waiting screen where you can log in as sysop. By defaulut telnet is enabled on port 1337.

NOTE: Ensure that your terminal screen is big enough - 80x25 at least.

1.6 Tools

Icy Board includes a comprehensive suite of tools for BBS management and development:

Core Executables

- `icboard` - The main BBS server daemon
- `icbsetup` - Terminal-based configuration and setup utility
- `pplc` - PPL compiler (source → PPE)
- `ppld` - PPL decompiler (PPE → source)
- `mkicbtxt` - Create/Edit ICBTXT files containing all strings used.
- `icbsysmgr` - System manager utility (user/group editor)
- `ppl-language-server` - Language server for PPL (for IDE integration)

1.7 Directory Layout

I tried to simplify the PCBoard system a bit but it has limits.

A typical Icy Board installation follows this structure:

```
FOO/                    # Your BBS root (created by icbsetup)
├── icboard.toml        # Main configuration file
├── icboard.log         # Runtime log file
├── users.toml          # User database
├── art/                # Graphics and art files
│   └── help/           # Help Files
├── main/               # Main board files
├── conferences/        # Conference data
└── tmp/                # Generated Files for backwards compatibility
```

1.8 main/ files

The `main/` directory contains core system configuration and data files:

Configuration Files

File	Description
`commands.toml` | Command definitions and keyboard shortcuts `conferences.toml` | Conference structure and access controls `languages.toml` | Language definitions (date formats, yes/no chars, locale) `protocols.toml` | File transfer protocol configurations `security_levels.toml` | Security level definitions and user limits	

User Management

File	Description
`users.toml` `groups` `vip_user.txt`	User database with all registered accounts Unix-style groups file for permission management VIP users list (sysop notified on login)

Security & Validation

File	Description
`tcan_user.txt` `tcan_passwords.txt` `tcan_email.txt` `tcan_uploads.txt`	Forbidden usernames (one per line) Forbidden passwords (weak/common passwords) Blocked email domains or addresses Prohibited upload filenames/patterns

System Files

File	Description
`icbtext.toml` `email.*`	System messages and prompts (customizable) Localized versions: `icbtext_de.toml`, etc. Email message base files (JAM format)

1.9 art/ files

It's recommended to use .pcb, .ans, .rip, .asc extensions instead of the old …G, …R sheme. This makes it easier to draw files with an ansi drawing tool as well. And file name lengths ar no longer an issue. Files can either be CP437 or UTF-8 - IcyBoard will do all conversions automatically. Note that UTF-8 requires the UTF-8 BOM. This is by design it's the only way to make a fast and correct decision about the file encoding.

Note: UTF-8 is recommended for everything.

1.10 icbsetup

icbsetup is the interactive TUI (text user interface) utility used to create, configure and maintain an Icy Board installation.

It's more than the classic PCBoard PCBSETUP untility.

- Create new BBS installations
- Import legacy PCBoard systems
- Help converting PPE plugins to modern systems

1.11 Create new BBS installations

1. Pick an identifier (letters / digits / underscore). Example: `FOO`
2. Create the instance:

   ```
   ./icbsetup create FOO
   ```

3. Start it:

   ```
   ./icboard FOO
   ```

Then the call waiting screen appears. You can access the setup or log in as user or sysop.

1.12 Import legacy PCBoard systems

Icy Board can ingest an existing PCBoard installation directly from your original `PCBOARD.DAT` (plus the related files it references). The importer converts binary/text formats into structured TOML, normalizes encodings to UTF-8 (with BOM for display files), hashes passwords, and recreates conferences, commands, security levels, protocols, colors, text resources, and user base metadata.

```
./icbsetup import /path/to/pcb /path/to/NEW_BBS_DIR
```

On success:

- Converted files populate `NEW_BBS_DIR/`
- A log file is written to `NEW_BBS_DIR/importlog.txt`
- You can start the board:

  ```
  ./icboard /path/to/NEW_BBS_DIR
  ```

Limitations are that the importer may import wrong/old paths - they may need to be manually adjusted. PPE plugins need to be manually converted as well.

1.13 Post-import tasks

1. See `importlog.txt` for warnings/errors (missing files, malformed records).
2. Manual convert PPE plugins (see below).
3. Test a migrated user: * Login * Read mail/conferences * Post a test message
4. Enable network services (telnet/ssh) only after verifying console launch works.

1.14 Converting PPE plugins to modern systems

Even if .PPE files don't need to be recompiled (they may work as-is) they may need to be adjusted to work with Icy Board. Most of them have a configuration that hint to old paths or files that don't exist anymore. So they need to be manually adjusted.

I recommend lowercasing all filenames and paths - Icy Board is case-sensitive as well as converting all text files to UTF-8 with BOM.

WARNING: Backup your original PPE files before conversion!

For that icbsetup has a PPE conversion assistant:

```
./icbsetup ppe-convert /path/to/ppe
```

This will lowercase all files and convert most fils from CP437 to UTF-8 with BOM. If a file is CP437 and is not converted. (This is likely the case because there are plenty of text files with strange extensions).

Manual convert a single file with

```
./icbsetup ppe-convert /path/to/file.nfo
```

This will convert a single CP437 file to UTF-8 with BOM. The PPE engine will automatically detect the encoding and convert to CP437 if needed.

1.14.1 Introduction to PPL

PPL (PCBoard Programming Language) is the classic scripting language used to extend and customize the PCBoard Bulletin Board System. Icy Board ships with a modern, memory-safe, fully reimplemented PPL toolchain:

- A virtual machine (runtime) executing PPE (compiled PPL) modules
- A modern, stricter compiler: `pplc`
- A robust decompiler: `ppld` (recovers structure from legacy PPEs)
- A Language Server (LSP) + VS Code extension for syntax help, hovers, navigation
- Extended language versions introducing optional new syntax and data types

Core Goals

1. High compatibility with PCBoard PPEs up to 15.4 (run, decompile, recompile)
2. Safe modernization (UTF-8 source, stricter diagnostics, secure password handling)
3. Progressive evolution (optional newer *language versions* that do not break older scripts unless you opt in)
4. Better tooling (warnings instead of silent miscompiles; IDE support; disassembly view)
5. Eliminate "anti-decompile" era tricks—make maintenance possible again

Vocabulary: Runtime vs Language Version

You will see two related version notions:

- **Runtime / PPE format version**: The bytecode / PPE container format (100-400).
- **Language version**: The surface syntax & feature set you target. By default language version = runtime version unless overridden (`--lang-version`).

You can (for example) generate a PPE in runtime format 400 but restrict yourself to language features of 340 to stay compatible with older boards (where applicable).

Toolchain Overview

Tool	Purpose
pplc *ppld* LSP Disasm	Compile *.pps* (UTF-8 or CP437) into a PPE (PCBoard Exec) Decompile an existing PPE back to readable PPL Editor services: outline, hover help, go-to, completion Optional internal view of low-level instructions

`ppld` - The Decompiler

```
ppld hello.ppe
```

Produces `hello.ppd` plus (optionally) a reconstructed control structure instead of flat GOTO spaghetti. Use `-d` to view a disassembly and `-r` for a minimal (raw) form.

Encoding & Character Set

- **Preferred input**: UTF-8 (modern editors)
- **Legacy**: Original DOS sources were CP437. Use `--cp437` if auto-detection fails.
- Compiler outputs CP437 in the PPE so legacy display semantics match PCBoard expectations.
- You may convert existing PPE data files to UTF-8 with: `icbsetup ppe-convert <PATH>` (make backups first).

Key Differences vs Legacy PPLC (Summary)

(See the detailed "PPL differences" section in `ppl.md` for the full list.)

- Reserved words: a larger set is now treated as keywords (`IF`, `FOR`, `CONTINUE`, etc.) to prevent ambiguous parses.
- Stricter: mismatched function return declarations are **errors** instead of silently ignored.
- Additional identifier support (e.g. UTF-8 cases, the Euro sign).
- Cleaner loop constructs / assignment operators in higher language versions.
- DECLARE blocks no longer required in newer versions (&gt;= 350).
- More (and safer) warnings for suspicious code; treat warnings seriously when porting.

Evolution by Language Version

- **&lt;= 340**: Classic era; close to PCBoard 15.4 semantics.
- **350** (PPL 4.0 modernization stage 1):
  - New loop forms: `REPEAT ... UNTIL` and `LOOP ... ENDLOOP`
  - Assignment operators (`+= -= *= /= %= &=`, etc.)
  - Inline `RETURN expr` (instead of assigning to function name)
  - Optional braces disambiguation improvements
  - Variable initializers: `TYPE VAR = expr` or array initializer `TYPE VAR = { a, b, c }`
- **400** (In progress – **experimental / subject to change**):
  - Distinct usage of `[]` for indexing, `{}` exclusively for array literals
  - Emerging *object-style* access to BBS domain entities (e.g. `CONFERENCE` objects, with member properties & helper functions)
  - Overloadable predefined functions (e.g. dual `CONFINFO` forms)
  - Goal: reduce need for manual file / config parsing in scripts

Use language gating to write compatible code:

```
;$IF VERSION < 350
    PRINTLN "Legacy path"
;$ELSE
    PRINTLN "Newer language features enabled"
;$ENDIF
```

Preprocessor Summary

Directives (start with `;$` on their own line):

- `;$DEFINE NAME[=VALUE]` – define a symbol (value optional)
- `;$IF expr` / `;$ELIF expr` / `;$ELSE` / `;$ENDIF` – conditional compilation
- Token substitutions: `;#Version` `;#Runtime` `;#LangVersion` expand to numeric values

Simple example:

```
PRINTLN "Compiler Version:", ;#Version
;$IF LANGVERSION >= 350
    PRINTLN "Modern language features active."
;$ENDIF
```

Types & Data Model (High Level)

- Scalars: Integer, Unsigned, Byte / Word, Boolean, Float, Double, Money, Date, Time
- Strings: Normal and “BigStr” (large string buffers)
- Arrays: 1–3 dimensional (indexed, zero-based internally)
- Password values (internally hashed if Argon2 / BCrypt storage is enabled)
- (Planned / partial in 400) Domain objects: Conference, MessageArea, FileArea, with member-like accessors or function wrappers.

Security & Safety Notes

- Passwords: Hashing (Argon2 / BCrypt) is enforced by configuration; scripts that attempt to transform (uppercase/lowercase) hashed values should expect no-ops.
- Avoid relying on internal hashes—display calls typically mask them.
- VM isolates runtime; catastrophic host crashes from buggy PPE logic are far harder now (memory safety from Rust).

Migration Workflow (Legacy PPE → Modern PPL)

1. **Decompile** legacy `FOO.PPE` → `FOO.PPS` with `ppld`.
2. **Review warnings** when recompiling with `pplc`; fix shadowed variables, questionable assignments, or deprecated idioms.
3. **Decide language version**: If you need pure compatibility, stick to 340. If modern loops / returns help clarity, switch to 350.
4. **Run under Icy Board**; validate interactive paths (menus, door launching, display).
5. **Iterate**: Use LSP tooling for rename, find references, and incremental modernization.

Disassembly for Learning

Use:

```
pplc myscript.pps --disassemble
# or
ppld legacy.ppe --disassemble
```

This produces a low-level opcode view. Helpful for verifying optimizer or diagnosing control-flow reconstruction.

Quick Reference Cheat Card

```
Compile:    pplc script.pps
Decompile:  ppld module.ppe
Disasm:     pplc script.pps -d
Encoding:   pplc --cp437 legacy.pps
Lang ver:   pplc script.pps --lang-version 350
PPE ver:    pplc script.pps --version 400
```

1.14.2 Developing PPL Applications

Create `hello.pps`:

```
PRINTLN "Hello from Icy Board PPL!"
```

Compile:

```
pplc hello.pps
```

Result: `hello.ppe`

1.14.3 PCBoard Programming Language (PPL)

This section enumerates all executable statement forms recognized by the modern Icy Board PPL compiler/VM. Internal AST variants like `Block` or `Empty` are not user-written and are omitted.

Version legend:
(100+) Available since earliest supported baseline (classic PCB era). (200+) Introduced when SELECT/CASE became available. (300+) Introduced with DECLARE / FUNCTION / PROCEDURE formalization. (350+) Modernization wave (repeat/until, loop/endloop, inline return expr, compound assignments).

Control Flow

IF single-line (100+)
Syntax: `IF ( <expr> ) <statement>` Executes exactly one following statement if expression is TRUE (non-zero / non-empty). No ELSE on same line.

IF / THEN multi-line (100+)
Syntax skeleton:

```
IF ( <expr> ) THEN
    <statements>
[ELSEIF ( <expr> ) THEN
    <statements>]...
[ELSE
```

(continues on next page)

(continued from previous page)

```
    <statements>]
ENDIF
```

Notes:

- Parentheses are required around the condition in modern forms.
- `ELSEIF` chains are evaluated in order; first TRUE branch wins.
- `ENDIF` terminator required.

SELECT / CASE (200+)

Multi-way conditional. Syntax:

```
SELECT ( <expr> )
    CASE <const_expr>[, <const_expr>...]:
        <statements>
    [CASE <const_expr_range_or_value_list>:
        <statements>]...
    [DEFAULT:
        <statements>]
ENDSELECT
```

Notes:

- Comparison is by value (string/integer/date/etc.) with standard PPL coercions.
- Multiple values per CASE separated by commas.
- Range syntax (e.g. 1..5) is supported where decompiler emits it.
- `DEFAULT` optional.

WHILE single-line (100+)

Syntax: `WHILE ( <expr> ) <statement>` Evaluates condition before each iteration; terminates when FALSE.

WHILE / ENDWHILE block (100+)

Syntax:

```
WHILE ( <expr> )
    <statements>
ENDWHILE
```

DO WHILE style (WhileDo AST) (100+ legacy form)

Some legacy PPEs decompile to a block starting with `WHILE` ending with `ENDWHILE` (same as above). The engine distinguishes single-line vs block internally; syntax to author is identical to "block" form.

REPEAT / UNTIL (350+)

Post-condition loop (always executes body at least once). Syntax:

```
REPEAT
    <statements>
UNTIL ( <expr> )
```

Loop ends when expression becomes TRUE (reverse of `WHILE` semantics).

LOOP / ENDLOOP (350+)

General loop for complex flows with manual BREAK/CONTINUE. Syntax:

```
LOOP
    <statements>
ENDLOOP
```

Equivalent to `while TRUE` with explicit termination via BREAK.

FOR / NEXT (100+)

Counter iteration.

Syntax:

```
FOR <identifier> = <start_expr> TO <end_expr> [STEP <step_expr>]
    <statements>
NEXT
```

or (legacy synonym) `ENDFOR` in place of `NEXT` (mapped internally).

Notes: * Counter variable is (re)assigned the start value first. * Step defaults to 1 or -1? (In classic PCBoard PPL: default is +1; negative requires explicit STEP -1.) * Inclusive end bound (executes while counter <= end when step > 0, or >= end when step < 0). * Modifying the loop variable inside the body is allowed but discouraged (can cause skipped termination).

GOTO (100+)

Syntax: `GOTO <label>` Transfers control unconditionally to a declared label (`:<label>` somewhere earlier or later). Use sparingly; prefer structured constructs.

GOSUB (100+)

Syntax: `GOSUB <label>` Pushes return point and jumps to label. Return occurs when an implicit `RETURN` or fall-through to end? In modern Icy Board PPL you normally use PROCEDURE/FUNCTION; GOSUB is legacy support.

Labels (100+)

Syntax: `:<label_name>` Declares a target for GOTO/GOSUB/BREAK label forms. Must be at statement start. Case-insensitive. Decompiler emits uppercase or original style.

BREAK (100+; extended label form 350+)

Syntax:

- Unlabeled: `BREAK` — exits innermost loop (WHILE / FOR / REPEAT / LOOP / SELECT inside loop).
- Labeled (350+): `BREAK :MyLabel` — jumps out to label (decompiler may emit for structured transforms).

CONTINUE (100+; labeled 350+)

Syntax:

- `CONTINUE` — skips to next iteration of current loop.
- `CONTINUE :Label` (350+) — advanced flow (rare; produced by transformations).

RETURN (100+; inline expression 350+)

Syntax (classic): `RETURN` Syntax (modern 350+): `RETURN <expr>` inside a FUNCTION (or to exit PROCEDURE early ignoring value). If legacy code assigns to the FUNCTION name and uses plain `RETURN`, both styles coexist.

Procedural & Calls

Procedure call (user-defined) (300+ for explicit PROCEDURE syntax; existed implicitly earlier)

Syntax:

- With arguments: `ProcName(arg1, arg2, ...)`
- No arguments: `ProcName`

Parentheses optional if no arguments (but recommended for clarity in modern code).

Predefined call (Built-in procedure statement) (100+)
Examples: `PRINT <expr_list>`, `PRINTLN ...`, `BYE`, file / user / message operations. Grammar: * Some accept argument lists separated by commas. * See "Predefined Procedures" section (not duplicated here).

GOSUB / RETURN pair (legacy)
See above under control flow. Consider rewriting heavy gosub usage into PROCEDURE/FUNCTION for clarity.

Assignment & Variables

LET / implicit LET (100+)
Classic form allows optional `LET` keyword:

`LET Var = <expr>` or `Var = <expr>`

Modern assignments (350+) support compound operators:

- = (assign)
- += -= *= /= %= (arithmetic)
- &= |= (bitwise / logical according to operand types)

Array / function-like indexing assignment:

`ArrayVar(i, j) = <expr>` (Parens style maintained for compatibility; some newer docs may show `ArrayVar[i, j]` when 400 object/index syntax fully stabilizes.)

Variable Declaration (typed) (300+)
Syntax prototype:

```
<TYPE> Var1[, Var2, Var3]
<TYPE> ArrayName(dim1[, dim2[, dim3]])
```

Types (summary): BOOLEAN, INTEGER, UNSIGNED, BYTE, WORD, SBYTE, SWORD, MONEY, FLOAT, DOUBLE, STRING, BIGSTR, DATE, EDATE, TIME, DDATE, TABLE, MESSAGEAREAID, PASSWORD (plus future USERDATA objects 400+).

Declarations can appear at top level (global) or at start of procedure/function bodies. (Versions <300 historically inferred variables on first assignment; modern compiler encourages explicit declarations for clarity and diagnostics.)

Comments

Single-line (100+)

- Semicolon: `;  This is a comment` (preferred)
- Apostrophe: `'  Also valid` (legacy)
- Leading *: `* Legacy style` (still recognized for imported sources)

Block comments (350+ experimental)
Parser supports multi-line block markers internally (emitted rarely by tools). Prefer single-line forms for portability.

Miscellaneous

SELECT fall-through
There is no implicit fall-through between CASE blocks; each CASE's block executes fully then control jumps to ENDSELECT (unless `BREAK` inside a nested loop is interpreted). Use multiple CASE value lists instead of stacked empty CASEs.

END (synthetic)
The decompiler may show an internal `End` comment or label transformation; you do not author a standalone `END` statement in modern PPL (`ENDIF`, `ENDWHILE`, `ENDSELECT`, `ENDLOOP`, `NEXT` serve as terminators).

Deprecated / Discouraged Patterns

- Heavy `GOTO` / `GOSUB` chains → replace with PROCEDURE / FUNCTION.
- Relying on implicit variable creation (pre-300 era) → add explicit declarations.
- Using uppercase/lowercase inconsistently for labels → labels are case-insensitive but pick one style (snake_case or ALLCAPS).
- Modifying loop variables inside FOR other than via STEP semantics → can create subtle off-by-one termination behavior.

Version Feature Matrix

Statement / Form	100–199	200–299	300–349	350+
IF/THEN	Yes	Yes	Yes	Yes
SELECT/CASE	•	Yes	Yes	Yes
WHILE	Yes	Yes	Yes	Yes
FOR/NEXT	Yes	Yes	Yes	Yes
GOTO / GOSUB	Yes	Yes	Yes	Yes
BREAK/CONTINUE	Yes	Yes	Yes	Yes (*)
REPEAT/UNTIL	•	•	•	Yes
LOOP/ENDLOOP	•	•	•	Yes
Compound assign	•	•	•	Yes
RETURN <expr>	•	•	•	Yes
DECLARE / PROC / FUNC | -		•	Yes	Yes

(*) Labeled BREAK / CONTINUE variants appear with modernization transforms (350+).

Examples

Single-line IF:

```
IF (X > 10) PRINTLN "High"
```

Block IF:

```
IF (User.TimeLeft < 5) THEN
    PRINTLN "Time is low!"
    GOTO WarnLoop
ELSEIF (User.TimeLeft < 1) THEN
    PRINTLN "Disconnecting soon..."
ELSE
    PRINTLN "Plenty of time."
ENDIF
```

FOR loop:

```
FOR I = 1 TO 10
    Total += I
NEXT
```

REPEAT / UNTIL:

```
REPEAT
    Line = INPUT()
UNTIL (Len(Line) = 0)
```

Compound assignment:

```
BytesLeft -= ChunkSize
Flags &= ~FLAG_NEW
```

Procedure call:

```
UpdateUserStats(UserId, TRUE)
```

Label / GOTO (legacy):

```
:RetryLogin
IF (Attempts > 3) GOTO Lockout
PRINT "Password: "
...
GOTO RetryLogin
:Lockout
PRINTLN "Too many attempts."
```

Return with value (350+):

```
FUNCTION Add(a, b) INTEGER
    RETURN a + b
ENDFUNC
```

PPL Functions

ABS (1.00)

`FUNCTION INTEGER ABS(INTEGER value)`

Parameters

- `value` – Integer input (may be negative)

Returns
Absolute value of `value`.

Description
Classic absolute value. Legacy edge case: the most negative 16-bit value may remain unchanged.

Example

```
DIFF = ABS(A - B)
```

ASC (1.00)

`FUNCTION INTEGER ASC(STRING ch)`

Parameters

- `ch` – String (first character used)

Returns
Code (0–255) of the first character (CP437 semantics).

Example

```
CODE = ASC("#")
```

CALLID (1.00)

`FUNCTION STRING CALLID()`

Parameters
None

Returns
Session / call identifier (may be empty on local sessions).

Description
Provides a per-connection identifier where supported.

CARRIER (1.00)

:PPL:FUNCTION INTEGER CARRIER()

Parameters None

Returns 1 if carrier (remote link) is active, 0 otherwise.

Description Legacy modem carrier detect abstraction; always 1 on purely local / emulated sessions.

CHR (1.00)

```
FUNCTION STRING CHR(INTEGER code)
```

Parameters

- code – 0–255

Returns
Single-character string.

Example

```
NL = CHR(13)
```

CONFINFO (3.20)

:PPL:FUNCTION <VARIANT> CONFINFO(INTEGER confnum, INTEGER field)

Parameters

- :PPL:confnum – Conference number
- :PPL:field – Field selector (1–54)

Returns
Value of the requested field (type varies).

Description
Reads a conference configuration attribute. (See field table earlier in this document.)

Example

```
IF (CONFINFO(100,50) = 5) PRINTLN "Conference 100 is FIDO type"
```

CONFINFO (Delete Queue Record) (3.20)

```
FUNCTION CONFINFO(INTEGER recnum)
```

Parameters

- `recnum` – Queue record number to delete

Returns
None

Description
Legacy overload used to delete Fido queue queue records (retained for compatibility).

Example

```
CONFINFO(6)
```

CURCOLOR (1.00)

```
FUNCTION INTEGER CURCOLOR()
```

Returns
Current display attribute (packed color value).

CURCONF (1.00)

`FUNCTION INTEGER CURCONF()`

Returns
Current active conference number.

CURSEC (1.00)

`FUNCTION INTEGER CURSEC()`

Returns
Effective / current security level.

CWD (3.20)

`FUNCTION STRING CWD()`

Returns
Current working directory path.

Example

```
PRINTLN "PWD = ", CWD()
```

DATE (1.00)

`FUNCTION DATE DATE()`

Returns
Current system date.

DAY (1.00)

`FUNCTION INTEGER DAY(DATE d)`

Returns
Day component (1–31).

DEFcolor (1.00)

`FUNCTION INTEGER DEFCOLOR()`

Returns
User's configured default color attribute.

DOW (1.00)

`FUNCTION INTEGER DOW(DATE d)`

Returns
Day of week (implementation-defined 0–6).

EXIST (1.00)

`FUNCTION BOOLEAN EXIST(STRING file)`

Parameters

- `file` – Path

Returns
TRUE if file exists.

Example

```
IF NOT EXIST("CONFIG.TXT") PRINTLN "Missing config."
```

FINDNEXT (3.20)

`FUNCTION STRING FINDNEXT()`

Returns
Next filename from active wildcard scan, or empty when exhausted.

FINDFIRST (3.20)

`FUNCTION STRING FINDFIRST(STRING file)`

Parameters

- `file` – Pattern (wildcards allowed)

Returns
First matching filename or empty if none.

FTELL (3.20)

`FUNCTION INTEGER FTELL(INTEGER channel)`

Parameters

- `channel` (INTEGER) - The file channel number (1-8)

Returns
Current file pointer position in bytes (0 if channel not open)

Description
FTELL returns the current file pointer offset for the specified file channel. If the channel is not open, it will return 0. Otherwise it will return the current position in the open file.

Example

```
FOPEN 1,"C:\MYFILE.TXT",O_RD,S_DN
FSEEK 1,10,SEEK_SET
PRINTLN "Current file offset for MYFILE.TXT is ",FTELL(1)
FCLOSE 1
```

GETDRIVE (3.20)

`FUNCTION INTEGER GETDRIVE()`

Returns
Current logical drive index (DOS semantics; virtual elsewhere).

GETENV (1.00)

`FUNCTION STRING GETENV(STRING var)`

Parameters

- `var` – Environment variable name

Returns
Value or empty if unset.

GETX (1.00)

```
FUNCTION INTEGER GETX()
```

Returns
Current cursor column (1-based).

GETY (1.00)

```
FUNCTION INTEGER GETY()
```

Returns
Current cursor row (1-based).

HOUR (1.00)

```
FUNCTION INTEGER HOUR(TIME t)
```

Returns
Hour component (0–23).

I2BD (3.20)

```
FUNCTION BIGSTR I2BD(INTEGER value)
```

Parameters

- `value` – Integer to serialize

Returns
8-byte BASIC double representation.

INKEY (1.00)

```
FUNCTION STRING INKEY()
```

Returns
Key (if immediately available) or empty.

INSTR (1.00)

```
FUNCTION INTEGER INSTR(BIGSTR str, STRING search)
```

Parameters

- `str` – Source text
- `search` – Substring

Returns
1-based position or 0 if not found.

KINKEY (1.00)

```
FUNCTION STRING KINKEY()
```

Returns
Last key pressed (blocking semantics differ from `INKEY` historically).

LEN (1.00)

```
FUNCTION INTEGER LEN(BIGSTR str)
```

Returns
Length of `str`.

LOGGEDON (1.00)

```
FUNCTION BOOLEAN LOGGEDON()
```

Returns
TRUE if a user is fully logged in.

LOWER (1.00)

```
FUNCTION BIGSTR LOWER(BIGSTR str)
```

Returns
Lower-case version.

LTRIM (1.00)

```
FUNCTION BIGSTR LTRIM(BIGSTR str, STRING charSet)
```

Returns
`str` with leading run of any chars in `charSet` removed.

MASK_ALNUM (1.00)

```
FUNCTION STRING MASK_ALNUM()
```

Returns
Alphanumeric mask token (used with certain masked input ops).

MASK_ALPHA (1.00)

```
FUNCTION STRING MASK_ALPHA()
```

Returns
Alphabetic mask token.

MASK_ASCII (1.00)

```
FUNCTION STRING MASK_ASCII()
```

Returns
Printable ASCII mask token.

MASK_FILE (1.00)

```
FUNCTION STRING MASK_FILE()
```

Returns
Filename mask token.

MASK_NUM (1.00)

```
FUNCTION STRING MASK_NUM()
```

Returns
Numeric input mask token.

MASK_PATH (1.00)

```
FUNCTION STRING MASK_PATH()
```

Returns
Path input mask token.

MASK_PWD (1.00)

```
FUNCTION STRING MASK_PWD()
```

Returns
Password mask token (input obscured).

MID (1.00)

```
FUNCTION BIGSTR MID(BIGSTR str, INTEGER pos, INTEGER len)
```

Returns
Substring starting at 1-based `pos` up to `len` characters.

MIN (1.00)

```
FUNCTION INTEGER MIN(TIME t)
```

Returns
Minute component (0–59).

MINLEFT (1.00)

```
FUNCTION INTEGER MINLEFT()
```

Returns
Minutes remaining in session.

MINON (1.00)

```
FUNCTION INTEGER MINON()
```

Returns
Minutes elapsed in current session.

MKDATE (1.00)

```
FUNCTION DATE MKDATE(INTEGER year, INTEGER month, INTEGER day)
```

Returns
Constructed date (invalid inputs may produce undefined / sentinel).

MONTH (1.00)

```
FUNCTION INTEGER MONTH(DATE d)
```

Returns
Month (1–12).

NOCHAR (1.00)

```
FUNCTION STRING NOCHAR()
```

Returns
System “No” confirmation character.

OS (3.20)

```
FUNCTION INTEGER OS()
```

Parameters
None

Returns
An integer indicating which operating system/PCBoard version the PPE is running under:

- 0 = Unknown
- 1 = DOS/Windows
- 2 = OS/2 (legacy - unused)
- 3 = Linux
- 4 = MacOS

Description
`OS` returns a value indicating the operating system environment. In Icy Board, this currently returns 0 (unknown) as a placeholder for compatibility. Legacy PPEs may use this to detect DOS vs OS/2 environments.

Example

```
SELECT CASE (OS())
    CASE 0
        PRINTLN "Running on Icy Board or unknown system"
    CASE 1
        PRINTLN "Running DOS version of Icy Board"
    CASE 2
        PRINTLN "Running OS/2 version of Icy Board"
END SELECT
```

I2BD (3.20)

```
FUNCTION BIGSTR I2BD(INTEGER value)
```

Parameters

- `value` – integer to serialize

Returns

- `BIGSTR` – 8 raw bytes representing a “bdreal” (double) form

Description

Converts a PPL INTEGER into an 8-byte BASIC double binary image.

Example

```
BIGSTR  raw
INTEGER v

v   = 12345
raw = I2BD(v)
FOPEN 1,"double.bin",O_WR,S_DN
FWRITE 1,raw,8
FCLOSE 1
```

TINKEY (3.20)

`FUNCTION STRING TINKEY(INTEGER ticks)`

Parameters

- `ticks` – Maximum clock ticks to wait (~18 ticks per second). Use 0 to wait indefinitely (implementation–limited upper bound ~4 hours or until carrier loss).

Returns

- `STRING` – Key pressed (special names like UP / DOWN / PGUP) or empty string if timed out

Description

Waits for user input for up to the specified number of clock ticks.

Example

```
STRING resp
PRINTLN "Press a key (10 second timeout)..."
resp = TINKEY(180)
IF (resp = "") THEN
    PRINTLN "Timeout."
ELSE
    PRINTLN "You pressed: ", resp
ENDIF
```

GETDRIVE (3.20)

`FUNCTION INTEGER GETDRIVE()`

Parameters

None

Returns

- `INTEGER` – Current “drive number” (A:=0, B:=1, C:=2, …). On non-DOS systems mapping is virtual.

Description

Returns the logical drive index. Primarily legacy; on modern platforms the value may be synthesized.

Example

```
INTEGER d
d = GETDRIVE()
IF (d = 2) PRINTLN "Drive C: is current"
```

CONFINFO (3.20)

`FUNCTION <VARIANT> CONFINFO(INTEGER confnum, INTEGER field)`

Parameters

- `confnum` – Conference number
- `field` – Field selector (see list)

Returns
Variant type depending on the field (STRING, BOOLEAN, INTEGER, BYTE, DREAL)

Description
Reads a conference configuration attribute. Field meanings:

Valid fields

1	STRING	Conference Name
2	BOOLEAN	Public Conference
3	BOOLEAN	Auto Rejoin
4	BOOLEAN	View Other Users
5	BOOLEAN	Make Uploads Private
6	BOOLEAN	Make All Messages Private
7	BOOLEAN	Echo Mail in Conf
8	INTEGER	Required Security public
9	INTEGER	Additional Conference Security
10	INTEGER	Additional Conference Time
11	INTEGER	Number of Message Blocks
12	STRING	Name/Loc MSGS File
13	STRING	User Menu
14	STRING	Sysop Menu
15	STRING	News File
16	INTEGER	Public Upload Sort
17	STRING	Public Upload DIR file
18	STRING	Public Upload Location
19	INTEGER	Private Upload Sort
20	STRING	Private Upload DIR file
21	STRING	Private Upload Location
22	STRING	Doors Menu
23	STRING	Doors File
24	STRING	Bulletin Menu
25	STRING	Bulletin File
26	STRING	Script Menu
27	STRING	Script File
28	STRING	Directories Menu
29	STRING	Directories File
30	STRING	Download Paths File
31	BOOLEAN	Force Echo on All Messages
32	BOOLEAN	Read Only
33	BOOLEAN	Disallow Private Messages

continues on next page

Table 1 – continued from previous page

34	INTEGER	Return Receipt Level
35	BOOLEAN	Record Origin
36	BOOLEAN	Prompt For Routing
37	BOOLEAN	Allow Aliases
38	BOOLEAN	Show INTRO in 'R A' scan
39	INTEGER	Level to Enter a Message
40	STRING	Join Password (private)
41	STRING	INTRO File
42	STRING	Attachment Location
43	STRING	Auto-Register Flags
44	BYTE	Attachment Save Level
45	BYTE	Carbon Copy List Limit
46	STRING	Conf-specific CMD.LST
47	BOOLEAN	Maintain old MSGS.NDX
48	BOOLEAN	Allow long (Internet) TO: names
49	BYTE	Carbon List Level
50	BYTE	NetMail Conference Type
51	INTEGER	Last Message Exported
52	DREAL	Charge Per Minute
53	DREAL	Charge Per Message Read
54	DREAL	Charge Per Message Written

Example

```
IF (CONFINFO(100,50) = 5) PRINTLN "Conference 100 is FIDO type"
```

See Also

- CONFINFO (object form – future user data variant)

CONFINFO (Delete Queue Record) (3.20)

```
FUNCTION CONFINFO(INTEGER recnum)
```

Parameters

- `recnum` – Queue record number to delete (legacy Fido queue semantics)

Returns

- None

Description

Legacy form used to delete Fido queue records. (Retained for script compatibility.)

Example

```
CONFINFO(6)  ; delete queue record #6
```

BS2I / BD2I / I2BS / I2BD See Also

- `FILEINF()` for file size/date/time
- `EXIST()` for existence checks

FINDFIRST (3.20)

`FUNCTION STRING FINDFIRST(STRING file)`

Parameters

- `file` – Path or pattern (may include wildcards like **.BAK*)

Returns

- First matching filename (no path normalization) or empty string if none

Description

Begins a wildcard (pattern) scan. Use `FINDNEXT()` repeatedly to enumerate additional matches. Only names are returned; use `FILEINF()` for metadata.

Example

```
STRING toDelete
toDelete = FINDFIRST("*.BAK")
WHILE (toDelete <> "")
    DELETE toDelete
    PRINTLN toDelete, " deleted."
    toDelete = FINDNEXT()
ENDWHILE
```

See Also

- `FINDNEXT()`, `EXIST()`, `FILEINF()`

FINDNEXT (3.20)

`FUNCTION STRING FINDNEXT()`

Parameters

- None

Returns

- Next filename in the active scan or empty string when exhausted

Description

Continues the enumeration started by `FINDFIRST()`. Stops when an empty string is returned.

Example

```
STRING n
n = FINDFIRST("*.BAK")
WHILE (n <> "")
    PRINTLN "Processing ", n
    n = FINDNEXT()
ENDWHILE
```

See Also

- `FINDFIRST()`, `FILEINF()`, `EXIST()`

RANDOM (1.00)

`FUNCTION INTEGER RANDOM(INTEGER max)`

Parameters

- `max` – Upper bound

Returns
Pseudo-random integer 0..max (legacy inclusive semantics).

READLINE (1.00)

`FUNCTION STRING READLINE(STRING file, INTEGER line)`

Returns
Contents of the specified (1-based) line or empty if out of range / not found.

REPLACE (1.00)

`FUNCTION BIGSTR REPLACE(BIGSTR str, STRING search, STRING replace)`

Returns
`str` with all `search` occurrences replaced.

REPLACESTR (2.00)

`FUNCTION BIGSTR REPLACESTR(BIGSTR str, STRING search, STRING replace)`

Returns
Same effect as `REPLACE` (alternate historical opcode).

RIGHT (1.00)

`FUNCTION BIGSTR RIGHT(BIGSTR str, INTEGER count)`

Returns
Last `count` characters (or whole string if shorter).

RTRIM (1.00)

`FUNCTION BIGSTR RTRIM(BIGSTR str, STRING charSet)`

Returns
`str` without trailing chars from `charSet`.

SCRTEXT (1.00)

```
FUNCTION STRING SCRTEXT(INTEGER col, INTEGER row, INTEGER len, BOOLEAN
rawCodes)
```

Returns
Screen slice (optionally stripping or preserving color codes).

SEC (1.00)

`FUNCTION INTEGER SEC(TIME t)`

Returns
Seconds (0–59).

SHOWSTAT (1.00)

```
FUNCTION BOOLEAN SHOWSTAT()
```

Returns
TRUE if user status line currently displayed.

SPACE (1.00)

```
FUNCTION BIGSTR SPACE(INTEGER count)
```

Returns
String of `count` spaces.

STRIP (1.00)

```
FUNCTION BIGSTR STRIP(BIGSTR str, STRING charSet)
```

Returns
`str` with every character in `charSet` removed.

STRIPATX (1.00)

```
FUNCTION BIGSTR STRIPATX(BIGSTR str)
```

Returns
`str` minus @Xnn color codes.

STRIPSTR (2.00)

```
FUNCTION BIGSTR STRIPSTR(BIGSTR str, STRING search)
```

Returns
`str` with all occurrences of `search` removed.

TIME (1.00)

```
FUNCTION TIME TIME()
```

Returns
Current system time.

TIMEAP (1.00)

```
FUNCTION STRING TIMEAP(TIME t)
```

Returns
12-hour formatted time with AM/PM.

TINKEY (3.20)

```
FUNCTION STRING TINKEY(INTEGER ticks)
```

Parameters

- `ticks` – Clock ticks to wait (0 = indefinite bound)

Returns
Pressed key or empty on timeout.

TOKCOUNT (1.00)

```
FUNCTION INTEGER TOKCOUNT()
```

Returns
Remaining token count in current parse buffer.

TOKENSTR (1.00)

```
FUNCTION STRING TOKENSTR()
```

Returns
Unconsumed token remainder as a string.

TOBIGSTR (2.00)

```
FUNCTION BIGSTR TOBIGSTR(<ANY> value)
```

Returns
`value` coerced to BIGSTR.

TOSTRING (1.00)

```
FUNCTION STRING STRING(<ANY> value)
```

Returns
String form of `value` (numbers decimal, BOOLEAN 0/1).

U_BDL (1.00)

```
FUNCTION INTEGER U_BDL()
```

Returns
Total bytes downloaded (cumulative).

U_BDLDAY (1.00)

```
FUNCTION INTEGER U_BDLDAY()
```

Returns
Bytes downloaded today.

U_BUL (1.00)

```
FUNCTION INTEGER U_BUL()
```

Returns
Total bytes uploaded.

U_FDL (1.00)

```
FUNCTION INTEGER U_FDL()
```

Returns
Files downloaded count.

U_FUL (1.00)

```
FUNCTION INTEGER U_FUL()
```

Returns
Files uploaded count.

U_INCONF (1.00)

```
FUNCTION BOOLEAN U_INCONF(INTEGER record, INTEGER conf)
```

Returns
TRUE if user record belongs to conference `conf`.

U_LDATE (1.00)

```
FUNCTION DATE U_LDATE()
```

Returns
Last logon date.

U_LDIR (1.00)

```
FUNCTION DATE U_LDIR()
```

Returns
Date user last scanned file directory (legacy metric).

U_LTIME (1.00)

```
FUNCTION TIME U_LTIME()
```

Returns
Last logon time.

U_LOGONS (1.00)

```
FUNCTION INTEGER U_LOGONS()
```

Returns
Number of prior completed logons.

U_MSGRD (1.00)

```
FUNCTION INTEGER U_MSGRD()
```

Returns
Messages read count.

U_MSGWR (1.00)

```
FUNCTION INTEGER U_MSGWR()
```

Returns
Messages written count.

U_NAME (1.00)

```
FUNCTION STRING U_NAME()
```

Returns
Current user's name.

U_PWDHIST (1.00)

```
FUNCTION STRING U_PWDHIST(INTEGER index)
```

Returns
Opaque historical password hash (don't display to callers).

U_PWDLC (1.00)

```
FUNCTION DATE U_PWDLC()
```

Returns
Date of last password change.

U_PWDTC (1.00)

```
FUNCTION INTEGER U_PWDTC()
```

Returns
Times password changed.

U_RECNUM (1.00)

```
FUNCTION INTEGER U_RECNUM(STRING username)
```

Returns
Record number for `username` (0 if not found).

U_STAT (1.00)

```
FUNCTION <VARIANT> U_STAT(INTEGER option)
```

Parameters

- `option` – Field selector (legacy; engine-defined meanings)

Returns
Stat value (type varies). Provided for compatibility; prefer explicit functions.

U_TIMEON (1.00)

```
FUNCTION INTEGER U_TIMEON()
```

Returns
Minutes used this call.

UPPER (1.00)

```
FUNCTION BIGSTR UPPER(BIGSTR str)
```

Returns
Upper-case version.

VALCC (1.00)

```
FUNCTION BOOLEAN VALCC(STRING ccNum)
```

Returns
TRUE if credit card number passes format/Luhn checks (legacy commerce support).

VALDATE (1.00)

```
FUNCTION BOOLEAN VALDATE(STRING dateStr)
```

Returns
TRUE if `dateStr` matches accepted date formats.

VALTIME (1.00)

```
FUNCTION BOOLEAN VALTIME(STRING timeStr)
```

Returns
TRUE if `timeStr` is valid.

VER (1.00)

```
FUNCTION INTEGER VER()
```

Returns
Legacy PCBoard version code (mapped / emulated).

YEAR (1.00)

```
FUNCTION INTEGER YEAR(DATE d)
```

Returns
Year component.

YESCHAR (1.00)

```
FUNCTION STRING YESCHAR()
```

Returns
System "Yes" confirmation character.

PPL Statements

CLS (1.00)

```
STATEMENT CLS
```

Parameters

- None

Returns
None

Description
Clears the caller's (and local) display screen and resets cursor to home position.

Example

```
CLS
PRINTLN "Welcome."
```

CLREOL (1.00)

`STATEMENT CLREOL`

Description
Clears from the current cursor position to the end of the line.

COLOR (1.00)

`STATEMENT COLOR(INTEGER attr)`

Parameters

- `attr` – Packed color (foreground/background + attributes)

Description
Sets current output color. Use `DEFCOLOR` / `CURCOLOR()` to query defaults.

Example

```
COLOR 14
PRINTLN "Yellow text"
```

PRINT (1.00)

`STATEMENT PRINT <expr_list>`

Description
Writes expressions to the console without appending a newline. Adjacent arguments separated by commas.

Example

```
PRINT "User: ", U_NAME()
```

PRINTLN (1.00)

`STATEMENT PRINTLN <expr_list>`

Description
Same as `PRINT` but appends a newline at end.

Example

```
PRINTLN "Bytes left:", MINLEFT()
```

SPRINT / SPRINTLN (1.00)

`STATEMENT SPRINT <expr_list>` `STATEMENT SPRINTLN <expr_list>`

Description
"Secure" print variants that typically filter control/high ASCII or respect user flags (implementation dependent).

MPRINT / MPRINTLN (1.00)

`STATEMENT MPRINT <expr_list>` `STATEMENT MPRINTLN <expr_list>`

Description
Message-area context print (legacy differentiation; acts like PRINT/PRINTLN under modern engine unless specialized).

NEWLINE (1.00)

`STATEMENT NEWLINE`

Description
Emits a single CR/LF pair (same as empty PRINTLN).

NEWLINES (1.00)

`STATEMENT NEWLINES(INTEGER count)`

Parameters

- `count` – Number of blank lines to emit (<=0 no-op)

INPUT (1.00)

`STATEMENT INPUT(<VAR> target)`

Parameters

- `target` – Variable to receive a raw line (basic editing)

Description
Reads a full line of user input (no masking/validation) into the variable.

INPUTSTR / INPUTINT / INPUTDATE / INPUTTIME / INPUTMONEY / INPUTCC (1.00)

`STATEMENT INPUTSTR(<VAR> target, INTEGER flags) STATEMENT INPUTINT(<VAR> target, INTEGER flags) STATEMENT INPUTDATE(<VAR> target, INTEGER flags) STATEMENT INPUTTIME(<VAR> target, INTEGER flags) STATEMENT INPUTMONEY(<VAR> target, INTEGER flags) STATEMENT INPUTCC(<VAR> target, INTEGER flags)`

Parameters

- `target` – Variable to fill
- `flags` – Bitwise OR of input behavior flags (e.g. FIELDLEN, UPCASE, ECHODOTS)

Description
Validating input routines specialized for type. For credit cards, format and Luhn validation can occur.

Example

```
INTEGER Age
INPUTINT Age, FIELDLEN + UPCASE
```

INPUTYN (1.00)

`STATEMENT INPUTYN(<VAR> target, INTEGER flags)`

Description
Prompts for a Yes/No style single-key answer; stores 'Y' or 'N' (or configured YESCHAR/NOCHAR) into `target`.

KILLMSG (3.20)

`STATEMENT KILLMSG(INTEGER confnum, INTEGER msgnum)`

Parameters

- `confnum` – Conference number containing the target message

- `msgnum` – Message number to delete

Returns
None

Description
Deletes the specified message from the given conference (if it exists and permissions allow).

Example

```
KILLMSG 10,10234
```

Notes
Fails silently in legacy semantics if the message cannot be removed. Modern engines may log a warning.

See Also
(future) message management functions / queries

SOUNDDELAY (3.20)

`STATEMENT SOUNDDELAY(INTEGER frequency, INTEGER duration)`

Parameters

- `frequency` – PC speaker tone frequency (legacy; ignored on some modern hosts)
- `duration` – Clock ticks to sound (~18 ticks = 1 second)

Returns
None

Description
Produces a tone for the specified duration. Introduced to replace the DOS two-step SOUND on / SOUND off sequence (not portable to OS/2 or modern systems) with a single call.

Example

```
IF (inputVal <> validVal) SOUNDDELAY 500,18
```

Notes
May be a no-op on non-emulated systems. Consider providing a visual fallback.

See Also
(None)

USELMRS (3.20)

`STATEMENT USELMRS(BOOLEAN useLmrs)`

Parameters

- `useLmrs` – TRUE to load alternate user's Last Message Read pointers on GETALTUSER; FALSE to suppress

Returns
None

Description
Controls whether subsequent `GETALTUSER` calls will also load the target user's LMRS (Last Message Read pointers). Disabling can save memory when many conferences exist and LMRS data is not needed.

Example

```
USELMRS FALSE
GETALTUSER 300
PRINTLN "Skipped loading user 300's LMRS to save memory."
USELMRS TRUE
GETALTUSER 300
PRINTLN "Now LMRS for user 300 are loaded."
```

Notes

Use the FUNCTION form `USELMRS()` (if provided) to query current state.

See Also

- `GETALTUSER`

ADDUSER (3.20)

`STATEMENT ADDUSER(STRING username, BOOLEAN keepAltVars)`

Parameters

- `username` – Name of the new user
- `keepAltVars` – TRUE leaves new user vars active (as if GETALTUSER on the new record); FALSE restores current user

Returns

None

Description

Creates a new user record with system defaults for all fields except the supplied name.

Example

```
ADDUSER "New Caller", TRUE
PRINTLN "Created & switched context to: New Caller"
```

Notes

Validate for duplicates before creation if possible.

See Also

- `GETALTUSER`
- `PUTALTUSER`

MKDIR (3.20)

`STATEMENT MKDIR(STRING path)`

Parameters

- `path` – Directory path to create

Returns

None

Description

Creates a directory (legacy DOS semantics). Intermediate path components are not automatically created.

Example

```
MKDIR "\PPE\TEST"
```

Notes
May fail silently if already exists or permissions deny.

See Also

- `RMDIR()`
- `CWD()`

RMDIR (3.20)

`STATEMENT RMDIR(STRING path)`

Parameters

- `path` – Directory path to remove (must be empty)

Returns
None

Description
Removes an empty directory.

Example

```
RMDIR "\PPE\TEST"
```

Notes
Will not remove non-empty directories.

See Also

- `MKDIR()`
- `CWD()`

CWD (3.20)

`FUNCTION STRING CWD()`

Parameters
None

Returns

- `STRING` – Current working directory path

Description
Retrieves the process (or session) current directory.

Example

```
PRINTLN "Current working directory = ", CWD()
```

Notes
Function (not a statement) but historically documented among statements.

See Also

- `MKDIR()`
- `RMDIR()`

ADJTUBYTES (3.20)

`STATEMENT ADJTUBYTES(INTEGER deltaBytes)`

Parameters

- `deltaBytes` – Positive or negative number of bytes to adjust the user's upload total

Returns
None

Description
Adjusts the tracked total upload bytes for the (current or alternate) user.

Example

```
GETALTUSER 10
ADJTUBYTES -2000
PUTALTUSER
```

Notes
Pair with `GETALTUSER` / `PUTALTUSER` to persist for alternate users.

See Also
(future accounting helpers)

GRAFMODE (3.20)

`STATEMENT GRAFMODE(INTEGER mode)`

Parameters

- `mode` – Display mode selector: * 1 = Color ANSI (if user supports) * 2 = Force color ANSI * 3 = ANSI black & white * 4 = Non-ANSI (plain) * 5 = RIP (if supported)

Returns
None

Description
Switches the caller's graphics/terminal capability mode.

Example

```
PRINTLN "Switching to color ANSI..."
GRAFMODE 1
```

Notes
Forcing modes unsupported by user terminal may cause display corruption.

See Also
Terminal / capability query functions (future)

FDOQADD (3.20)

`STATEMENT FDOQADD(STRING addr, STRING file, INTEGER flags)`

Parameters

- `addr` – FidoNet destination address
- `file` – Packet / file to queue
- `flags` – Delivery mode: 1=NORMAL, 2=CRASH, 3=HOLD

Returns
None

Description
Adds a record to the Fido queue for later processing.

Example

```
FDOQADD "1/311/40","C:\PKTS\094FC869.PKT",2
```

Notes
Paths should be validated; behavior undefined if file not present.

See Also

- `FDOQMOD()`
- `FDOQDEL()`

FDOQMOD (3.20)

`STATEMENT FDOQMOD(INTEGER recnum, STRING addr, STRING file, INTEGER flags)`

Parameters

- `recnum` – Existing queue record number to modify
- `addr` – Updated FidoNet address
- `file` – Updated file path
- `flags` – 1=NORMAL, 2=CRASH, 3=HOLD

Returns
None

Description
Modifies an existing Fido queue entry.

Example

```
FDOQMOD 6,"1/311/40","C:\PKTS\UPDATED.PKT",1
```

Notes
Duplicate legacy doc blocks collapsed into one canonical entry.

See Also

- `FDOQADD()`
- `FDOQDEL()`

FDOQDEL (3.20)

`STATEMENT FDOQDEL(INTEGER recnum)`

Parameters

- `recnum` – Queue record to delete

Returns
None

Description
Deletes a Fido queue record.

Example

```
FDOQDEL 6
```

Notes
Deleting a non-existent record has no effect (legacy behavior).

See Also

- `FDOQADD()`
- `FDOQMOD()`

CONFINFO (Modify) (3.20)

`STATEMENT CONFINFO(INTEGER confnum, INTEGER field, VAR newValue)`

Parameters

- `confnum` – Conference number
- `field` – Field selector (1–54)
- `newValue` – Value to assign (type must match field definition)

Returns
None

Description
Writes a single conference configuration field. Field meanings mirror the FUNCTION form (see earlier table for 1–54). Only appropriate types are accepted.

Security / Privacy:
Field 40 (Join Password) SHOULD be handled carefully. Avoid logging or echoing this value.

Example

```
CONFINFO 200,1,"Stan's New Conference Name"
```

Notes
Writing invalid types may produce runtime errors or be ignored depending on implementation.

See Also

- `CONFINFO()` (read / variant form)

PROMPTSTR (1.00)

`STATEMENT PROMPTSTR(<VAR> target, INTEGER flags)`

Description
Like INPUTSTR but prints a system prompt first (legacy UI consistency).

TOKENIZE (1.00)

`STATEMENT TOKENIZE(STRING line)`

Parameters

- `line` – Source to break into tokens for later `GETTOKEN()` / `TOKCOUNT()`

Description
Loads the internal token buffer with split tokens (whitespace / delimiter rules legacy-defined).

GETTOKEN (1.00)

```
STATEMENT GETTOKEN(<VAR> target)
```

Description
Pops next token (or empty if none) into `target`.

SHELL (1.00)

```
STATEMENT SHELL(STRING command)
```

Description
Executes a system shell / external program (availability/security can be restricted).

BYE / GOODBYE (1.00)

`STATEMENT BYE` `STATEMENT GOODBYE`

Description
Terminates user session gracefully (GOODBYE synonym). May trigger logoff scripts, accounting flush.

HANGUP (1.00)

```
STATEMENT HANGUP
```

Description
Immediate disconnect / carrier drop (hard termination). Prefer BYE for clean logout.

LOG (1.00)

```
STATEMENT LOG(STRING line)
```

Description
Appends `line` to the system activity / event log.

DELAY (1.00)

```
STATEMENT DELAY(INTEGER ticks)
```

Parameters

- `ticks` – ~18 per second

Description
Sleeps (non-busy) for specified ticks unless carrier loss or abort condition.

WAIT (1.00)

```
STATEMENT WAIT(INTEGER ticks)
```

Description
Similar to DELAY but may flush output first or enforce a minimum pacing (legacy pacing semantics).

BEEP (1.00)

```
STATEMENT BEEP
```

Description
Emits an audible terminal bell (Ctrl-G) if user’s terminal supports it.

KBDSTUFF (1.00)

`STATEMENT KBDSTUFF(STRING text)`

Description
Queues keystrokes into the input buffer as if typed by the caller.

KBDFLUSH / KBDCHKON / KBDCHKOFF (1.00)

`STATEMENT KBDFLUSH` `STATEMENT KBDCHKON` `STATEMENT KBDCHKOFF`

Description
Manage keyboard buffering and carrier/abort key checks.

SENDMODEM (1.00)

`STATEMENT SENDMODEM(STRING raw)`

Description
Sends raw bytes (unfiltered) to remote terminal/modem (legacy; may be sanitized in modern environments).

PAGEON / PAGEOFF (1.00)

`STATEMENT PAGEON` `STATEMENT PAGEOFF`

Description
Enable/disable user "page" requests (sysop chat paging).

CHAT (1.00)

`STATEMENT CHAT`

Description
Enters sysop chat mode if available (toggles live keyboard sharing).

FLAG (1.00)

`STATEMENT FLAG(INTEGER flagId)`

Description
Sets a transient per-session flag bit (implementation-defined). Often used with prompt display logic.

ALIAS (1.00)

`STATEMENT ALIAS(STRING newName)`

Description
Temporarily changes display name (legacy; may not persist).

GETUSER / PUTUSER (1.00)

`STATEMENT GETUSER(INTEGER record)` `STATEMENT PUTUSER`

Parameters (GETUSER)

- `record` – User record number

Description
Loads user record into current context / writes modified current user back to storage.

GETALTUSER / FREALTUSER / PUTALTUSER (1.00 / 3.20+ semantics)

`STATEMENT GETALTUSER(INTEGER record)` `STATEMENT FREALTUSER` (Persist changes with `PUTALTUSER` (if provided) or `PUTUSER` after adjusting context.)

Description
Loads an alternate user profile (for inspection/modification) while preserving original active user data.

ADJTIME (1.00)

`STATEMENT ADJTIME(INTEGER deltaMinutes)`

Description
Adjusts remaining time this call by `deltaMinutes` (negative to subtract).

ADJBYTES / ADJTBYTES / ADJDBYTES / ADJTFILES (1.00+)

`STATEMENT ADJBYTES(INTEGER delta)` `STATEMENT ADJTBYTES(INTEGER delta)` (uploads) `STATEMENT ADJDBYTES(INTEGER delta)` (downloads) `STATEMENT ADJTFILES(INTEGER delta)` (upload file count)

Description
Adjust quota/accounting counters. Prefer the more explicit *T*/*D* forms when available. (You already documented `ADJTUBYTES`—the upload bytes variant in expanded semantics.)

DELETE / RENAME (1.00)

`STATEMENT DELETE(STRING file)` `STATEMENT RENAME(STRING old, STRING new)`

Description
Remove or rename a filesystem entry (basic DOS semantics; silent failure if missing or permission denied).

FCREATE / FOPEN / FAPPEND (1.00)

```
STATEMENT FCREATE(INTEGER ch, STRING file, INTEGER access, INTEGER share)
STATEMENT FOPEN(INTEGER ch, STRING file, INTEGER access, INTEGER share)
STATEMENT FAPPEND(INTEGER ch, STRING file, INTEGER access, INTEGER share)
```

Parameters

- `ch` – Channel number (1–8)
- `file` – Path
- `access` – One of `O_RD`, `O_WR`, `O_RW`
- `share` – One of `S_DN`, `S_DR`, `S_DW`, `S_DB`

Description
Opens a file for subsequent buffered I/O. Create always truncates/creates; Append opens write and seeks end.

Example

```
FCREATE 1,"log.txt",O_WR,S_DN
FPUTLN 1,"Session start"
FCLOSE 1
```

FPUT / FPUTLN / FPUTPAD (1.00)

`STATEMENT FPUT(INTEGER ch, STRING data) STATEMENT FPUTLN(INTEGER ch, STRING data) STATEMENT FPUTPAD(INTEGER ch, STRING data, INTEGER width)`

Description
Write text (optionally newline or right-pad to width).

FGET (1.00)

`STATEMENT FGET(INTEGER ch, <VAR> target, INTEGER length)`

Description
Reads up to `length` bytes (or line depending on legacy mode) into `target`.

FSEEK (1.00)

`STATEMENT FSEEK(INTEGER ch, INTEGER offset, INTEGER whence)`

Parameters

- `whence` – SEEK_SET, SEEK_CUR, SEEK_END

FFLUSH (1.00)

`STATEMENT FFLUSH(INTEGER ch)`

Description
Forces buffered channel output to disk.

FCLOSE / FCLOSEALL (1.00)

`STATEMENT FCLOSE(INTEGER ch) STATEMENT FCLOSEALL`

Description
Close one or all open channels (releases locks).

FREAD / FWRITE (1.00)

`STATEMENT FREAD(INTEGER ch, <VAR> bigstrTarget, INTEGER bytes) STATEMENT FWRITE(INTEGER ch, BIGSTR buffer, INTEGER bytes)`

Description
Raw byte read/write (binary).

FREWIND (1.00)

`STATEMENT FREWIND(INTEGER ch)`

Description
Equivalent to `FSEEK ch,0,SEEK_SET`.

DISPFILE / DISPTEXT / DISPSTR (1.00)

`STATEMENT DISPFILE(STRING file, INTEGER flags) STATEMENT DISPTEXT(STRING text, INTEGER flags) STATEMENT DISPSTR(STRING text)`

Description
Display PCBoard @-code aware content (file or inline). Flags may control paging, security, or language substitution.

RESETDISP / STARTDISP (1.00)

STATEMENT RESETDISP STATEMENT STARTDISP(INTEGER flags)

Description
Manage internal buffered display/paging state.

JOIN (1.00)

STATEMENT JOIN(INTEGER confnum)

Description
Switches current conference (permission verified).

CONFFLAG / CONFUNFLAG (1.00)

STATEMENT CONFFLAG(INTEGER confnum, INTEGER flagMask) STATEMENT CONFUNFLAG(INTEGER confnum, INTEGER flagMask)

Description
Set / clear specific conference attribute bits (F_MW, F_SYS, etc.).

BITSET / BITCLEAR (1.00)

STATEMENT BITSET(<VAR> var, INTEGER bit) STATEMENT BITCLEAR(<VAR> var, INTEGER bit)

Description
Sets or clears (0-based) bit in integer variable.

INC / DEC (1.00)

STATEMENT INC(<VAR> var) STATEMENT DEC(<VAR> var)

Description
var = var ± 1 (legacy bytecode convenience).

ALIAS (already documented above, retained for clarity)

SAVESCRN / RESTSCRN (1.00)

STATEMENT SAVESCRN STATEMENT RESTSCRN

Description
Save/restore current screen buffer (local + remote if supported).

ANSIPOS (1.00)

STATEMENT ANSIPOS(INTEGER col, INTEGER row)

Description
Directly positions cursor (1-based coordinates).

KBDSTRING (1.00)

STATEMENT KBDSTRING(STRING str)

Description
Inject entire string into keyboard buffer (contrast KBDSTUFF which may differ historically).

SETENV (1.00)

```
STATEMENT SETENV(STRING name, STRING value)
```

Description
Sets (or overrides) an environment variable for subsequent processes / shell calls.

CHDIR (3.20)

```
STATEMENT CHDIR(STRING path)
```

Description
Changes the current working directory.

RENAME (already included above)

SHORTDESC (3.20)

```
STATEMENT SHORTDESC(STRING text)
```

Description
Sets a short descriptive string for the PPE (shown in sysop listings / logs).

MOVEmsg (3.20)

```
STATEMENT MOVEMSG(INTEGER fromConf, INTEGER msgNum, INTEGER toConf)
```

Description
Moves a message between conferences (permissions & existence required).

SETBANKBAL (3.20)

```
STATEMENT SETBANKBAL(INTEGER userRec, MONEY amount)
```

Description
Adjusts stored "bank" balance (economy/game feature – semantics engine-defined).

WEBREQUEST (400 tentative)

```
STATEMENT WEBREQUEST(STRING url, <VAR> responseBigStr)
```

Description
Experimental HTTP GET/HEAD style fetch populating response data (subject to change; may require runtime 400).

D* Database / Table Primitives (Overview)

(Full per-statement docs can be added—summary here)

- `DCREATE name, layout...` – Create structured data file
- `DOPEN name` / `DCLOSE` / `DCLOSEALL`
- Record navigation: `DTOP`, `DBOTTOM`, `DGO n`, `DSKIP delta`
- CRUD: `DADD`, `DAPPEND`, `DBLANK` (new empty), `DDELETE`, `DRECALL`
- Locking: `DLOCK`, `DLOCKR`, `DLOCKG`, `DUNLOCK`
- Field IO: `DGET`, `DPUT`
- Index / seek: `DSEEK`, `DFCOPY`
- Alias / pack: `DSETALIAS`, `DPACK`

- NewName variants (DN*) manage named index or alt dataset.

Add a request if you want these expanded in the same detailed template.