

Future-work analysis — 2026-10-25

This note records the design discussions from 2026-10-25 so the questions do not have to be re-derived later. It covers TBITSIZE, address arithmetic, set ordering, complex types, pragmas, UTF-8 string libraries, collections, the TopSpeed B-tree Toolkit, communications, pyserial, ModBase, network-library design, terminal escape libraries, C FFI and GTK4/SQLite/SDL bindings, an audit of the existing gm2 bindings, the GTK4 OOP facade, PIM versus ISO coroutines, external assembly, and the dynamic-set design. It is not a commitment to implement everything.

State at the time of writing: suite 310/310, one-shot fixpoint OK (3,845,439 bytes), per-object fixpoint OK (11 objects), corpus 521/612 compile-OK, 0 crashes, 64 LANG and 27 LIB.

1. TBITSIZE

TBITSIZE is a GNU Modula-2 SYSTEM procedure function:

PROCEDURE TBITSIZE (<type>) : **CARDINAL**;

It returns the **minimum number of bits needed to represent a type**, not its normal storage size in bytes. It is mainly useful for packed records/bitfields. A BOOLEAN may be one bit logically but occupy a word in an unpacked record.

- GNU Modula-2 supports it (and uses it in M2WIDEST).
- V3 does not support it.
- V3's existing SIZE/TSIZE report storage size in bytes.
- Adding it would need:
 - a type/variable argument form in the grammar;
 - a descriptor query for minimum bit width (integer/real/char/enum/ subrange/record/packed-record cases);
 - constant folding so TBITSIZE(BYTE) = 8 works in CONST;
 - interaction with packed record layout.

It is a small-to-medium language gap, not a runtime feature.

2. ADDRESS arithmetic

ISO SYSTEM defines:

PROCEDURE ADDADR (addr: **ADDRESS**; offset: **CARDINAL**): **ADDRESS**;

PROCEDURE SUBADR (addr: **ADDRESS**; offset: **CARDINAL**): **ADDRESS**;

PROCEDURE DIFADR (addr1, addr2: **ADDRESS**): **INTEGER**;

PROCEDURE MAKEADR (high: <type>; ...): **ADDRESS**;

V3 currently supports:

- ADR
- ADDRESS/pointer assignment
- ADDRESS/pointer equality
- VAL/CAST for compatible scalar representations

V3 does **not** support:

- ADDADR/SUBADR/DIFADR/MAKEADR
- ADDRESS + offset, ADDRESS - offset, ADDRESS - ADDRESS
- C-style arithmetic on typed POINTER TO values

Why it is missing

ADDRESS is an opaque pointer-like value in V3's type system. Raw pointer arithmetic is not part of the PIM/ISO base language and would weaken V3's separation between typed pointers and addresses.

Recommended implementation

Add the ISO SYSTEM functions first, not general pointer arithmetic:

- byte-addressed ADDADR/SUBADR;
- DIFADR returning the byte difference (range-checked for INTEGER);
- optional MAKEADR;
- tests for offsets, null addresses, overflow and pointer/address equality.

For external C/assembly interaction, a C helper remains the pragmatic escape hatch.

3. Set ordering

PIM/ISO set operations are:

- equality =, inequality #, membership IN;
- union +, difference -, intersection *, symmetric difference /.

There is **no standard <, <=, >, >= ordering for sets**. GNU Modula-2 adds inclusion ordering:

```
a <= b    (* a is a subset of b *)
a < b     (* a is a proper subset of b *)
a >= b    (* a is a superset of b *)
a > b     (* a is a proper superset of b *)
```

V3 currently rejects these; corpus case largeset5 remains a LANG failure.

If implemented as a gm2-compatibility extension

Fixed/inline sets:

```
a <= b    iff    (a AND NOT b) == 0 for every word
a < b     iff    a <= b AND a # b
```

Dynamic sorted-range sets:

```
a <= b    iff every range of a is covered by a range of b
a < b     iff a <= b AND a # b
```

Compiler work:

- SymTab.RelCheck/OrdCheck: accept set ordering;
- Lower: branch fixed/dynamic set ordering;
- QbeGen: subset/superset comparisons;
- tests: empty sets, equal sets, subset, proper subset, superset, disjoint sets, mixed widths;
- largeset5 would likely move from LANG to compile-OK.

This is a safe gm2 compatibility extension, not standard PIM/ISO.

4. PIM versus ISO coroutines

PIM

Low-level SYSTEM primitives:

```

TYPE PROCESS;
PROCEDURE NEWPROCESS (body: PROC; workspace: ADDRESS;
                      size: CARDINAL; VAR p: PROCESS);
PROCEDURE TRANSFER (VAR from: PROCESS; to: PROCESS);
PROCEDURE IOTRANSFER (VAR from, to: PROCESS;
                      interruptNo: CARDINAL);

```

- caller supplies workspace;
- caller explicitly transfers;
- no scheduler;
- no coroutine identity query;
- no attach/detach/protection model;
- IOTRANSFER uses an explicit interrupt number.

V3 implements this with ucontext; IOTRANSFER currently degrades to a normal transfer because there is no interrupt subsystem.

ISO 10514-1 COROUTINES

```

DEFINITION MODULE COROUTINES;

```

TYPE

```

COROUTINE;
INTERRUPTSOURCE = CARDINAL;
PROTECTION = [UnassignedPriority .. 7];

```

```

PROCEDURE NEWCOROUTINE (procBody: PROC; workspace: ADDRESS;
                      size: CARDINAL; VAR cr: COROUTINE;
                      [initProtection: PROTECTION]);
PROCEDURE TRANSFER (VAR from: COROUTINE; to: COROUTINE);
PROCEDURE IOTRANSFER (VAR from: COROUTINE; to: COROUTINE);
PROCEDURE ATTACH (source: INTERRUPTSOURCE);
PROCEDURE DETACH (source: INTERRUPTSOURCE);
PROCEDURE IsATTACHED (source: INTERRUPTSOURCE): BOOLEAN;
PROCEDURE HANDLER (source: INTERRUPTSOURCE): COROUTINE;
PROCEDURE CURRENT (): COROUTINE;
PROCEDURE LISTEN (p: PROTECTION);
PROCEDURE PROT (): PROTECTION;
PROCEDURE TurnInterrupts (to: PROTECTION): PROTECTION;
PROCEDURE ListenLoop;
END COROUTINES.

```

Main differences:

Area	PIM	ISO
Context type	PROCESS	COROUTINE
Creation	NEWPROCESS	NEWCOROUTINE + optional protection
Switch	TRANSFER	TRANSFER
Interrupt transfer	IOTRANSFER(..., interruptNo)	IOTRANSFER(from, to) with attached source
Current context	none	CURRENT
Interrupt registration	none	ATTACH/DETACH/IsATTACHED/HANDLER
Protection/priority	none	PROTECTION , LISTEN, PROT, TurnInterrupts
Event loop	user-written	ListenLoop

V3 has PIM coroutines but no ISO COROUTINES module. A pure coroutine subset (NEWCOROUTINE,

TRANSFER, CURRENT, workspace management) could map onto the existing ucontext layer. The real interrupt/protection parts need an OS signal/event layer.

5. External assembly instead of inline ASM

External assembly works today through EXTERNAL and the normal linker; no compiler support for inline ASM is required.

```
DEFINITION MODULE AsmOps;
PROCEDURE Add (a, b : CARDINAL) : CARDINAL;
END AsmOps.
```

```
IMPLEMENTATION MODULE AsmOps;
PROCEDURE Add (a, b : CARDINAL) : CARDINAL;
EXTERNAL;
END AsmOps.
```

```
.text
.globl Add
.type Add,@function
```

Add:

```
movl %edi, %eax
addl %esi, %eax
ret
```

```
cc -c asm_add.s -o asm_add.o
```

```
M2 -c AsmOps.def AsmOps.mod
```

```
qbe -o AsmOps.s AsmOps.ssa
```

```
cc -c AsmOps.s -o AsmOps.o
```

```
cc Prog.o AsmOps.o asm_add.o runtime/syslib/shim.c -o Prog -lm
```

For M2make --separate, add the assembly object with --extra asm_add.o or link manually.

ABI notes

- QBE target is amd64_sysv.
- INTEGER/CARDINAL are 32-bit; LONGINT is 64-bit.
- pointers are 64-bit l values.
- VAR parameters receive addresses.
- external procedures get no V3 static-link argument.
- open arrays, especially ARRAY OF CHAR, are passed as descriptor addresses, not raw C arrays.
- record/array value parameters are passed by address.
- composite results use a hidden result pointer.

For anything beyond scalar arguments, use a small C wrapper around the assembly routine so the V3-facing signature is C-compatible.

This is the recommended route for low-level code; inline ASM is unnecessary.

6. Dynamic SET OF CARDINAL / INTEGER — chosen design

V3 now uses a sorted linked list of disjoint inclusive ranges:

set = pointer to

```
{ lo : CARDINAL; hi : CARDINAL; next : pointer }
```

This was chosen over a full 2³²-bit bit-vector (512 MiB per set value) and over a linked list of individual bits (too much per-value overhead). The range-list representation is:

- O(number of contiguous runs), not O(cardinality);

- compact for dense ranges;
- supports the full 32-bit base range;
- INTEGER values map with `VAL(CARDINAL, x) XOR 0x80000000`.

Implemented operations include empty/range literals, INCL/EXCL/IN, assignment, equality, union/intersection/difference and symmetric difference. Bounded sets such as `SET OF [0..100000]` keep the inline bit-vector form.

Current limitations

- dynamic `SET CONST` is rejected (loud 230), because no static range-list layout exists yet;
- record/array assignment containing dynamic sets currently copies the pointer rather than deeply copying the range list;
- nodes use `malloc`; assignment frees old lists but there is no GC or program-exit cleanup;
- `PACKEDSET` remains capped at 32 bits.

Future work: runtime-initialised dynamic constants, deep composite copy, ownership/lifetime rules, and optionally a hash/paged bitset if range lists become too fragmented.

7. Complex types: built-in types versus a library

V3 currently has no complex arithmetic. `COMPLEX32`, `COMPLEX64` and `COMPLEX128` are declaration-compatible aliases of `REAL` (currently 8 bytes each), so they are placeholders, not real complex types. Their names imply two 4-, 8- and 16-byte components (8, 16 and 32 bytes respectively), so the current aliases are also the wrong size. The comparison table correctly marks `COMPLEX/LONGCOMPLEX` as unsupported.

What ISO expects

ISO Modula-2 treats `COMPLEX/LONGCOMPLEX` as **predefined types**. The language is expected to provide:

- complex arithmetic operators (+, -, *, /);
- the `CMPLX` constructor;
- `RE/IM` accessors;
- complex-aware assignment, parameters and function results.

Transcendental functions live in the separate `ComplexMath` and `LongComplexMath` library modules. The ISO architecture is therefore:

```

compiler-known COMPLEX/LONGCOMPLEX types
+ operator lowering / builtins
+ ComplexMath / LongComplexMath libraries

```

A separate library can provide complex-number *functions*, but it cannot by itself provide natural Modula-2 syntax such as `a + b` or `CMPLX`. Operator overloading does not exist in Modula-2, so a library-only design is necessarily a nonstandard utility API.

Why not classes

Classes are a poor fit for complex numbers:

- complex numbers are small value types, not objects with identity;
- class methods give `a.Add(b)` instead of `a + b`;
- class allocation/dispatch is unnecessary overhead for a pair of reals;
- classes are not ISO Modula-2 OO (V3's `CLASS` is Clarion-form);
- a class still cannot add operators, literals or `RE/IM` syntax.

Use a value record or a compiler-known type, not a class.

What generics can do

V3 has ISO 10514-2 generics, so a generic implementation could share code between real and long-real variants:

```
GENERIC MODULE Complex(T);  
TYPE Value = RECORD re, im : T END;  
...  
END Complex.
```

That is useful for implementation sharing, but generics still do not provide operators, literals or ISO source compatibility. Since V3's REAL and LONGREAL are both 8 bytes today, two small concrete modules are also acceptable and simpler for separate compilation.

Option A: nonstandard value-record library

For practical complex math without an ISO-conformance claim:

```
DEFINITION MODULE Complex;
```

```
TYPE
```

```
  T = RECORD  
    re, im : REAL  
  END;
```

```
PROCEDURE Add (a, b : T) : T;
```

```
PROCEDURE Sub (a, b : T) : T;
```

```
PROCEDURE Mul (a, b : T) : T;
```

```
PROCEDURE Div (a, b : T) : T;
```

```
PROCEDURE Conj (z : T) : T;
```

```
PROCEDURE Abs (z : T) : REAL;
```

```
PROCEDURE Arg (z : T) : REAL;
```

```
END Complex.
```

Advantages: value semantics, no allocation, simple ABI, easy to test. Limitations: call syntax only; no operator or literal support; does not satisfy ISO COMPLEX.

Do **not** call this type COMPLEX or reuse the existing COMPLEX* aliases, because that would falsely suggest ISO support.

Option B: ISO-compatible built-in types

Implement COMPLEX/LONGCOMPLEX as compiler-known value types, with arithmetic lowered to runtime helpers:

```
COMPLEX = two REAL values
```

```
a + b    -> __m2_complex_add(a, b)
```

```
a * b    -> __m2_complex_mul(a, b)
```

```
CMPLX    -> compiler builtin
```

```
RE/IM     -> compiler builtins
```

Then implement ComplexMath and LongComplexMath as libraries on top. This avoids requiring QBE to understand a native complex type while still providing ISO syntax and semantics.

Work items:

- representation/layout (COMPLEX is 16 bytes with the current 8-byte REAL; COMPLEX32 cannot be 8 bytes until a real 4-byte SHORTREAL exists);
- type checking for mixed real/complex expressions and conversions;
- lowering of arithmetic to helper calls;
- CMPLX, RE, IM and constant expressions;
- ABI for complex parameters/results, including separate compilation;

- library implementations of `abs`, `arg`, `conj`, `sqrt`, `exp`, `ln`, trigonometric functions and `polarToComplex`;
- tests for arithmetic, constants, mixed precision, copying, parameters, function results and separate objects.

For C interop, do not assume the C99 `_Complex` ABI without checking; prefer small C wrappers where necessary.

Recommendation

- If ISO conformance is a project goal, complex types must eventually be built-in; the library is the math layer, not a replacement.
 - If only practical complex math is wanted, add the value-record `Complex` library first, with no classes and no fake `COMPLEX` names.
 - Generics are optional implementation sharing, not a substitute for language support.
 - Regardless of option, remove or clearly mark the current `COMPLEX32/64/128 REAL` aliases so programs cannot silently get wrong-size, non-complex values.
-

8. Pragmas: TopSpeed compatibility versus portable scoped pragmas

V3 has no general pragma facility. It currently parses and ignores two gm2 declaration pragmas (`__BUILTIN__` and `__ATTRIBUTE__ __BUILTIN__ (( name ))`) after `PROCEDURE`, and its TopSpeed sidecar lexes (`*# ... *`) as an ordinary comment. The local `EXCEPTIONS.def` therefore omits `<* noreturn *>`.

There are two separate design questions:

1. give V3 a native pragma facility;
2. consume legacy TopSpeed pragmas.

The recommendation is to separate them: use the portable, scoped model as the native design, and add a narrow TopSpeed compatibility mode.

8.1 TopSpeed V3 pragmas

TopSpeed Modula-2 pragmas are a special form of comment:

```
(*# save *)
(*# check(range => on, index => off, overflow => off) *)
PROCEDURE P (...);
(*# restore *)
```

They are **stateful compiler directives**:

- a setting applies from that point forward;
- `save` / `restore` maintain a stack of compiler settings;
- a procedure's calling convention is the setting in force at its definition;
- a variable's layout is the setting in force at its declaration.

They are grouped into pragma classes:

- `check(...)` — range, index, overflow, NIL, stack checks;
- `call(...)` — calling convention, register parameters, interrupt handlers, open-array passing;
- `data(...)` — segments, packing, volatile, enum size, stack/heap size;
- `optimize(...)` — CSE, register allocation, peephole, CPU choice;
- `module(...)` — initialization code, implementation-object rules;
- `name(...)` — external symbol naming;
- `debug(...)` — debug information and procedure tracing;
- `option(...)` — language extensions such as bitwise operations on cardinals;
- `save` / `restore` — compiler-option stack;
- `link`, `project`, `define` — project-system integration.

Older \$ compiler directives are also accepted by TopSpeed for compatibility.

Strengths

- Pragmatic and expressive.
- Local check overrides without recompiling the build system.
- Historical TopSpeed code already uses the syntax.
- Easy to lex as a comment; V3's corpus already parses (**# ... **).

Problems for V3

- Much of it is tied to 16-bit x86: segments, near/far calls, interrupts, `reg_param`, `iopl`, memory models, overlays, CPU and coprocessor selection. These do not map to V3's amd64/QBE target.
- Semantics are position/state-dependent rather than attached to declarations.
- Source annotations are mixed with global build settings such as optimisation and debug options.
- It is a frozen, vendor-specific dialect, not a portable standard.
- Because (**# ... **) is a comment to other compilers, ignoring it silently changes checks, layout or ABI without any diagnostic.

TopSpeed pragmas are therefore useful as a **compatibility adapter**, not as V3's native pragma model.

8.2 Benjamin / R10 portable pragmas

The portable proposal (used by Modula-2 R10/BSK) uses *<* ... *>*:

PROCEDURE Reboot *<*NORETURN*>*;

TYPE Header = **RECORD** *<*PACKED*>*

...

END;

VAR Port *<*VOLATILE*>* : BYTE;

It defines:

- standard pragmas in uppercase;
- implementation-defined pragmas with a lowercase implementation prefix, e.g. *<*v3.SomeOption=...*>*;
- explicit scopes for module, type, record, field, procedure and parameter placement, plus point-forward and no-scope placement;
- combinable clauses in one block;
- PRESETS / UNSET for external pragma settings instead of a TopSpeed-style global save / restore;
- a mandatory BSK subset: DEPRECATED, FFI, FFIDENT, INLINE and PRIVATETO, plus ENCODING where Unicode is supported.

The full proposal is much larger: ABI, ADDR, ALIGN, PACKED, PADBITS, LEADBITS, VOLATILE, NORETURN, PURE, WEAK, IN, OUT, SINGLEASSIGN, ARC, GC, OWNER, BORROWER, RETAIN, RELEASE, MSG, TELL, TICKET and others. R10 itself omits some of these (RANGE, RELEASE, RETAIN).

Strengths

- Attached to syntax, not a global mutable state.
- Deterministic and easier to validate.
- Fits definition/implementation consistency and separate compilation.
- *<* ... *>* aligns with ISO-style/gm2 pragma syntax.
- Implementation-defined pragmas have a collision-free namespace.
- Provides a path to standard annotations for ABI, layout, volatility and diagnostics.
- Much better fit for a strict, self-hosting compiler.

Problems

- The proposal is large; implementing all of it is a project of its own.

- Many pragmas are promises that require real analysis: PURE/WEAK/DETM need effect checking, IN/OUT need parameter-write analysis, NORETURN needs control-flow checks and SINGLEASSIGN needs data-flow checks.
- Some are irrelevant to V3: ARC, GC, OWNER, BORROWER, RETAIN, RELEASE; RESIDENT, CONTIGUOUS and LOWLATENCY are mostly OS/runtime hints; ENCODING needs scanner/Unicode work.
- ADDR, ALIGN, PACKED, PADBITS and LEADBITS require real layout support.
- The proposal is still evolving; V3 should implement a versioned subset, not chase every edit.

8.3 Comparison

Aspect	TopSpeed V3	Benjamin / portable
Status	Vendor dialect	Proposed portable specification, used by R10/BSK
Delimiters	(<code>*# ... *</code>)	<code><# ... *></code>
Form	<code>class(name => value, ...), save, restore</code>	PRAGMA, PRAGMA=expr, clause sequences
Scope	Stateful from point forward	Attached to explicit syntactic scopes
ABI/layout	Many 16-bit-specific directives	Portable ABI/layout annotations
Unknown pragmas	Often ignored or comment-like	Standard versus implementation-defined, diagnosable
Separate compilation	Needs external project-state discipline	Naturally attachable to symbols and object stamps
V3 fit	Compatibility adapter	Native pragma design

TopSpeed is not a standard; the `<# ... *>` syntax is the better long-term direction because it is already used by ISO-style/gm2 compilers and by the portable proposal.

8.4 Recommended V3 design

Use `<# ... *>` with explicit scope attachment as the native pragma syntax. Do not make `(# ... *)` the default; handle it only in the TopSpeed sidecar or behind an explicit compatibility flag.

Phase 1 — infrastructure and low-risk semantics

- scanner/parser support for `<# ... *>` and adjacent blocks;
- AST/symbol-table pragma nodes with source locations;
- scope validation;
- unknown-pragma policy;
- DEPRECATED;
- NORETURN;
- NOINLINE;
- INLINE as advisory, with a warning if not honoured;
- MSG / TICKET diagnostics if desired;
- PRIVATETO.

NORETURN is especially valuable immediately: it lets V3 consume standard ISO/gm2 definition files without stripping pragmas.

Phase 2 — layout and ABI

- PACKED;
- ALIGN / MEMALIGN;
- FFI / FFIDENT;
- ADDR only once absolute variables and address arithmetic are designed;
- PADBITS / LEADBITS only after packed-record layout exists.

Every ABI/layout pragma must participate in:

- definition/implementation consistency checks;
- the separate-compilation object ABI stamp;
- the content hash used by M2make --separate.

Otherwise two objects can disagree about record layout or symbol names and still link.

Phase 3 — contracts

- PURE, WEAK, DETM;
- IN / OUT;
- SINGLEASSIGN.

These should not be accepted as no-ops. If V3 cannot verify them, it should reject them or emit a clear “accepted but not enforced” warning. Silently pretending to enforce a promise is worse than not supporting it.

Do not implement initially

- ARC, GC, OWNER, BORROWER, RETAIN, RELEASE;
- RESIDENT, CONTIGUOUS, LOWLATENCY;
- full PRESETS / pragma-file machinery;
- Unicode ENCODING.

These need runtime support V3 does not have, or are platform hints with little value on the current target.

8.5 TopSpeed compatibility

Keep TopSpeed pragmas out of the default V3 grammar. Handle them in the existing TopSpeed-V3-M2.atg sidecar, or add a --topspeed-pragmas mode to the main compiler.

Map only the meaningful subset:

- check(range/index/overflow/nil/stack);
- data(packed/volatile);
- module(init_code);
- call(c_conv);
- save / restore.

Reject or warn about:

- near_call, far, segments, seg_name, ds_*;
- reg_param, reg_saved, reg_return;
- interrupt, iopl;
- overlays, memory models, cpu, copro;
- Windows/OS2/DOS-specific options.

TopSpeed stateful pragmas can be translated by keeping an option stack during parsing and snapshotting the current state onto each declaration. That is an adapter, not the native model.

8.6 Never silently ignore

- A TopSpeed pragma that V3 treats as a comment must produce at least a warning in TopSpeed-compatibility mode.
- An unknown standard pragma should be an error.
- An unknown v3.* pragma should be a warning or informational message.

Otherwise V3 will compile code with different checks or ABI and call it success.

8.7 Sources

- TopSpeed Developer’s Guide, chapter on pragmas.

- trijezdci/M2-Pragmas wiki, especially *Pragma Specification*, *Pragma Placement* and *Pragma Scopes*.
- Modula-2 R10/BSK pragma facilities chapter.

9. UTF-8 string and character libraries

V3 has good language-level Unicode support but only a thin library layer. The architecture is “UTF-8 at the edges, codepoints internally”, not “UTF-8 strings everywhere”.

9.1 What exists

- UCHAR — 32-bit code point.
- UString — codepoint-based literals and ARRAY OF UCHAR values.
- U'...' / U"... " literals with strict RFC 3629 decoding at compile time (error 234 on malformed input).
- UCHR, CHR8, UORD.
- UString whole-string assignment to a fixed ARRAY OF UCHAR (count-clamped, zero-codepoint terminated).
- + concatenation for UString literals and single-codepoint operands, plus LEN/LENGTH/HIGH for literals.
- runtime/syslib/Utf8 — scalar codec: Decode (one sequence from a byte array to UCHAR and next) and Encode (one UCHAR to bytes). MaxUTF8Length = 4, MaxCodePoint = 1114111.
- lib/iso/TextIO — UTF-8 output: UWriteChar, UWriteString, UWriteLn.
- Tests: utf8_prog, utext_prog, t_ustrcat, t_ustrassign, t_unicode, plus malformed-literal rejection tests.

Utf8 lives in runtime/syslib and is always shared. TextIO is in lib/iso, so its U* output helpers are ISO-dialect library API; other dialects can import Utf8 manually.

9.2 What is missing

Area	Current state
UTF-8 codec	One codepoint at a time only
Whole-string validation	Manual Decode loop only
UTF-8 input	None
UTF-8 output	TextIO.U* only
UTF-8 length/index/slice	None
UString library	None
UString comparison/search	None
Dynamic UString	None
Unicode case conversion	None
Unicode character classes	CharClass is ASCII-only
Normalization / collation / grapheme clusters	None
Locale/bidi support	None

lib/common/Strings is byte-oriented:

- Length counts bytes, not codepoints;
- Slice, Pos, Insert, Replace and Delete use byte positions;
- Capitalize is explicitly ASCII;
- slicing UTF-8 bytes can split a multibyte sequence.

lib/common/CharClass is ASCII-only; there is no UCHAR version and no Unicode category lookup.

Input is byte/line oriented. SysIO, STextIO, IOChan, Files and StreamFile have no UTF-8 decoding or validation policy.

9.3 UString is not a complete string type

A U"..." literal is a descriptor { count, codepoints... }. A declared ARRAY OF UCHAR is a fixed-length array. As a result:

- literals have a logical count;
- fixed arrays have a capacity plus a zero terminator after assignment;
- LEN/HIGH on a fixed array refer to declared bounds, not the number of codepoints actually assigned;
- TextIO.UWriteString writes 0 .. HIGH(u), not just through a zero terminator;
- there is no general UString equality, comparison, slicing or search.

UString concatenation is bounded and transient. The shim uses a static 4096-codepoint buffer, silently truncates beyond that, and the result is valid only until the next `m2ustrcat` call. Single-codepoint conversion uses a small rotating pool (16 descriptors). This is adequate for literal expressions, not for a general string runtime.

9.4 Correctness item

`Utf8.Encode` documents that it returns 0 for a non-scalar value, but its implementation does not reject the surrogate range D800..DFFF. Those values are currently encoded as three-byte UTF-8 sequences, which is invalid UTF-8. Add the surrogate check and a regression test.

9.5 Recommended library design

Keep byte strings and codepoint strings as separate layers. Do not make Strings Unicode-aware in place; existing code expects bytes.

1. UTF-8 byte-string module

For interoperability with files, sockets, C APIs and existing V3 ARRAY OF CHAR strings:

- Validate
- Length (codepoint count)
- ByteLength
- Index (codepoint index to byte offset)
- Next / Prev
- Slice
- Compare
- Find
- ToUString
- FromUString

This is the most practical "UTF-8 string" library for V3 because CHAR arrays are already the byte-oriented representation.

2. UString codepoint module

For operations over ARRAY OF UCHAR:

- logical length
- copy/concat with explicit capacity
- compare
- slice
- find
- conversion to/from UTF-8
- codepoint iteration

First resolve the length model: either consistently use a zero terminator, or introduce an explicit descriptor with separate length and capacity. Do not mix both.

3. Input/output

Add UTF-8 input alongside output:

- whole-line UTF-8 reading;
- a documented invalid-input policy;
- UReadChar / UReadString (or equivalent);
- conversion into UCHAR/UString values.

4. Unicode properties and case

This is a later layer:

- case mapping;
- character categories;
- whitespace;
- normalization;
- collation.

These need generated Unicode tables, not hand-written ranges.

9.6 Suggested order within the Unicode work

1. Fix Utf8.Encode surrogate rejection and add a test.
2. Add the UTF-8 byte-string module (validation, iteration, length/index, conversion).
3. Add the UString codepoint module and settle its length model.
4. Add UTF-8 input.
5. Add Unicode case/property tables if needed.
6. Keep Strings/CharClass byte/ASCII semantics and add separate Unicode modules rather than changing existing behaviour.

10. Collections library and the TopSpeed B-tree Toolkit

10.1 Current state

V3 has no collections library. What exists is only infrastructure:

- lib/common/Storage — ALLOCATE, DEALLOCATE, REALLOCATE
- lib/gm2/SysStorage — raw allocation
- lib/gm2/DynamicStrings — heap strings
- ISO generics: type/value/procedure parameters and kind constraints
- generic container demonstrations in tests:
 - compiler/tests/showcase23lib.def — DList(T)
 - compiler/tests/showcase24stack.def — GStack(T)

Those test modules are proof of concept, not production libraries: they have no Done/Clear, no ownership policy, and use one NEW per node. The MyWork/Strings/DataStructures FIFO/LIFO/dynamic-array files are incomplete and are not part of V3.

10.2 What the TopSpeed B-tree Toolkit is

The real toolkit is installed with the TS-V3 tree. Useful references:

```
$HOME/bin/Dos/M2/TS-V3/INCLUDE/BTREE.DEF
$HOME/bin/Dos/M2/TS-V3/SRC/BTREE.MOD
$HOME/bin/Dos/M2/TS-V3/DOC/BTREE.DOC
$HOME/bin/Dos/M2/TS-V3/Manuals/B-Tree toolkit.odt
```

It is a **disk-based B-tree/ISAM toolkit**, not an in-memory collections library. Its architecture is:

- one physical file (FHandle)
- multiple logical data and index files (IHandle) in that physical file
- fixed or variable-length data records, up to 65,535 bytes
- index keys up to 128 bytes
- multiple indexes per data file, updated automatically by Add/Delete

- application-supplied CompareFunction and KeyFunction
- duplicate keys configurable per index
- Find/Search (exact and greater-or-equal), Next/Prev/Reset cursors
- Add, Delete, Change, Open, OpenData, OpenIndex, Close, Flush, LastError, LastFError
- 1024-byte pages with a 32-page LRU cache
- optional data compression through Pack/APACK2
- shared-file locking and multi-tasking support
- 4 GiB physical-file limit
- free-list allocation; the file never shrinks automatically
- no automatic compaction
- default error handler prints and terminates the program

The current definition also includes AccessMode values such as FixSize, Size16, Size32, NoDealloc and Compress.

The source is 16-bit DOS/OS/2 code: inline x86 assembly, segment arithmetic, TopSpeed pragmas, FI0x, CoreMem, and raw ADDRESS/ARRAY OF BYTE interfaces. It is a design reference, not a portable V3 implementation.

10.3 Good ideas to borrow

- Separate logical data and index files, with multiple indexes per data file.
- Application-supplied key derivation and comparison.
- Cursor-oriented API (Find, Search, Next, Prev, Reset) rather than exposing pages.
- Per-handle error state, with an optional handler.
- Bounded page cache with LRU replacement.
- Configurable duplicate-key handling.
- Explicit open/close/flush lifecycle.
- A versioned file header and free-space management.
- Optional compression as a separate layer.

These map naturally to V3 generics:

```
Compare : PROCEDURE(a, b : K) : INTEGER;
Hash     : PROCEDURE(k : K) : CARDINAL;
Equal    : PROCEDURE(a, b : K) : BOOLEAN;
```

10.4 What not to copy

- Fatal-error-by-default: a library must return status.
- Raw ADDRESS/ARRAY OF BYTE public interfaces: provide typed wrappers.
- Fixed 128-byte keys, 1 KiB pages and 4 GiB files: make limits explicit/configurable where practical.
- 16-bit segment/assembly code.
- DOS SHARE/FI0x locking: use V3 file APIs and POSIX locking later.
- No compaction: provide a copy/rebuild tool or document the limitation.
- Raw binary records without schema/version metadata.
- Automatic indexing only for future additions; rebuilding an index must be an explicit operation.
- Copying the source code: it is old, platform-specific and carries its own licensing terms.

10.5 Recommended library layout

Keep in-memory collections and the persistent B-tree separate.

In-memory containers

Vector(T)	dynamic array
Deque(T)	FIFO / double-ended queue
Stack(T)	LIFO
HashMap(K,V)	hash table with Hash/Equal callbacks
BTreeMap(K,V)	optional ordered map/set with Compare

Vector is the foundation. Stack and Deque should reuse it or a ring-buffer implementation. Avoid one allocation per element unless stable references are required.

HashMap should use open addressing with tombstones or separate chaining, a load factor around 0.7, and explicit resize/rehash. Add a `StringMap(V)` convenience wrapper with owned string keys.

BTreeMap is the in-memory ordered structure: comparator callback, first/last/next/prev, range scan, optional duplicate keys, no pager.

Persistent B-tree

BTreeFile – disk-based index, toolkit-inspired

This is a separate library, not part of Vector/HashMap. It needs:

- file format and versioning
- page size and cache policy
- split/merge and free-space management
- cursor state
- locking policy
- crash/durability policy
- compaction/rebuild
- optional compression

The TopSpeed toolkit is the checklist for this stage.

10.6 V3 constraints

- No GC: containers need explicit Init/Done/Clear.
- No automatic deep copy: pointer-element ownership must be documented.
- Storage.REALLOCATE exists, but arbitrary pointer arithmetic does not. TopSpeed uses AddAddr/IncAddr/DecAddr assembly helpers; V3 cannot rely on that.
- A truly growable Vector(T) needs one of:
 - a fixed maximum capacity parameter,
 - chunked storage,
 - or a small compiler/runtime extension for variable-sized typed arrays.
- Generic instances are supported, but multiple actual-type instances should be regression-tested.
- Keep array length and capacity separate.
- Use records and modules, not classes, for containers.
- Do not use a fatal error handler as the default.

Suggested error model:

```
TYPE Status = (Ok, Empty, NotFound, InvalidIndex,  
                OutOfMemory, DuplicateKey);
```

If enum handling is awkward in a given context, use integer status codes or BOOLEAN plus LastError.

10.7 Suggested order within the collections work

1. Vector(T)
2. Deque(T) / Stack(T)
3. HashMap(K,V)
4. StringMap(V)
5. In-memory BTreeMap(K,V)
6. Persistent BTreeFile based on the TopSpeed architecture

The TopSpeed B-tree Toolkit should not delay steps 1–3.

11. Communications and ModBase toolkits

These are three different things and should not be merged into one design:

- the TopSpeed Communication Toolkit: DOS/OS-2 serial driver, file transfer protocols, terminal emulation and scripting;
- pyserial: a modern portable serial transport library (transport only);
- ModBase: a dBase III-compatible database engine.

All TopSpeed/ModBase source is commercial historical code; use it as a design reference, not copyable source. pyserial is BSD-3-Clause.

11.1 Current V3 state

V3 has no serial, socket, terminal-emulation, scripting or database library. What exists:

- lib/iso/IOChan, StreamFile, Files for byte/line channels;
- lib/common/SysIO and runtime/syslib/SysShim for stdio/file I/O;
- EXTERNAL and the C shim for libc access;
- PIM coroutines (no general scheduler);
- Storage/SysStorage.

11.2 TopSpeed Communication Toolkit

References:

```
$HOME/bin/Dos/M2/TS-V3/SRC/RS232.MOD
$HOME/bin/Dos/M2/TS-V3/SRC/XMODEM.MOD
$HOME/bin/Dos/M2/TS-V3/SRC/YMODEM.MOD
$HOME/bin/Dos/M2/TS-V3/SRC/WXMODEM.MOD
$HOME/bin/Dos/M2/TS-V3/SRC/KERMIT.MOD
$HOME/bin/Dos/M2/TS-V3/SRC/TERM.MOD
$HOME/bin/Dos/M2/TS-V3/SRC/SCRIPTC0.MOD
$HOME/bin/Dos/M2/TS-V3/SRC/SCRIPTPR.MOD
$HOME/bin/Dos/M2/TS-V3/INCLUDE/RS232.DEF
$HOME/bin/Dos/M2/TS-V3/INCLUDE/FTSupport.DEF
$HOME/bin/Dos/M2/TS-V3/Manuals/Modula2-Communication-toolkit.odt
```

The low-level RS232 driver:

- COM1-COM4 with fixed I/O addresses;
- direct 8250/16550 UART register access;
- interrupt vectors and interrupt-controller programming;
- 1024-byte TX and RX ring buffers;
- RTS/CTS or XON/XOFF flow control;
- high/low water marks at 3/4 and 1/4 of the RX buffer;
- SerialRead timeout in tenths of a second;
- SerialWrite copies into the TX ring;
- DTR/RTS/DCD control and break generation;
- TX/RX hooks for monitoring;
- separate DOS and OS/2 implementations.

Limitations:

- singleton driver, not a multi-port object model;
- SelectPort disables the previous handler;
- InitPort/Install have no error result;
- busy-waiting on full TX buffer and read timeout;
- no read/write cancellation;
- no buffer-level query;
- no clean open/close lifecycle (only Install/UnInstall);
- fixed port table and hardware assumptions.

The protocol layer:

- XMODEM (checksum/CRC negotiation);
- YMODEM, YMODEM Batch, YMODEM-g;
- Windowed XMODEM;
- Kermit;
- shared FTSupport global state and FBuf;
- CRC table and status/progress variables.

The protocol implementations are useful as protocol state-machine references, but they call RS232 and FIO directly and drive status windows through CommUtil/Window. They are not transport-agnostic. Only one transfer can be active at a time.

The toolkit also includes:

- VT100, VT52 and ANSI terminal emulation;
- keyboard driver;
- menu library;
- ProComm-like compiled script language;
- scanner, compiler, pseudo-code interpreter;
- cooperative SingleStep execution;
- WAITFOR, WHEN, macros, logging, file-transfer commands.

The script VM is a useful design for an application-level automation engine, not a general V3 library requirement.

11.3 pyserial as the transport model

pyserial 3.5 is installed at:

/usr/lib/python3/dist-packages/serial

Its architecture:

- common SerialBase plus platform backends;
- POSIX backend uses termios, fcntl and select/poll;
- Windows backend uses the Win32 serial API;
- URL handlers for socket://, rfc2217://, loop://, spy://;
- port enumeration via list_ports and sysfs metadata;
- threaded reader/protocol adapters and separate asyncio integration.

Useful API semantics:

- separate read, write and inter-byte timeouts;
- blocking, non-blocking and infinite timeout modes;
- short reads on timeout and partial-write handling;
- in_waiting / out_waiting;
- cancel_read / cancel_write;
- input/output buffer reset;
- flush waits for transmission to drain;
- DTR/RTS/CTS/DSR/RI/CD;
- XON/XOFF and RTS/CTS flow control;
- send_break;
- exclusive access via flock;
- custom baud via BOTHER, low-latency mode, RS485 on Linux;
- loopback/spy transports for tests.

Comparison:

Area	TopSpeed	pyserial
Target	DOS/OS-2 PC UART	POSIX/Windows OS tty
Serial layer	direct UART + IRQ	termios + OS driver + poll
Buffering	application ring buffers	kernel buffers + waiting counts

Area	TopSpeed	pyserial
Timeouts	tenths, busy wait	read/write/inter-byte, monotonic
Cancellation	none	read/write cancellation
Multiple ports	one active singleton	multiple instances
Discovery	COM1-COM4	device paths, URLs, sysfs
Advanced	hooks	RS485, custom baud, low latency, exclusive
Protocols	X/Y/WX/Kermit	none in core

Recommendation: model V3's serial layer on pyserial's feature set, not on TopSpeed's hardware driver. Keep TopSpeed's protocol descriptions and state machines as references for optional file-transfer modules.

11.4 Recommended communications architecture

Core:

```

Serial      POSIX tty transport
Transport   read/write/flush/poll/cancel/close interface
Socket      optional TCP transport
Loopback    deterministic test transport

```

Protocols and application layers:

```

XModem / YModem / WModem  optional
Kermit                    only if required
Terminal                  optional VT100/ANSI emulation
Script                    optional application-level VM

```

The protocol modules must depend only on Transport, never directly on Serial.

A C shim should implement the POSIX details:

```

m2serial_open
m2serial_configure
m2serial_read
m2serial_write
m2serial_poll
m2serial_flush
m2serial_set_modem
m2serial_send_break
m2serial_close

```

Use status codes, not fatal errors. Support:

- blocking and non-blocking modes;
- separate read/write/inter-byte timeouts;
- cancellation;
- in_waiting/out_waiting;
- flow control;
- modem lines;
- exclusive open;
- optional custom baud/RS485/low-latency;
- loopback tests.

A blocking read with timeout is sufficient initially. A reader coroutine or event loop can be added later; avoid binding the API to threads.

11.5 ModBase

References:

```
$HOME/bin/Dos/M2/TS-V3/MODBASE/MODBASE3.DEF
$HOME/bin/Dos/M2/TS-V3/MODBASE/DBINDEXES.DEF
$HOME/bin/Dos/M2/TS-V3/MODBASE/DBFIELDS.DEF
$HOME/bin/Dos/M2/TS-V3/MODBASE/LOCKS.DEF
$HOME/bin/Dos/M2/TS-V3/MODBASE/HASH.DEF
$HOME/bin/Dos/M2/TS-V3/Manuals/Modbase 3.odt
```

ModBase is a dBase III-compatible database engine:

- .DBF data files, .NDX indexes, .DBT memo files;
- fixed/variable fields;
- multiple indexes per data file;
- automatic index updating via an update list;
- buffered record and index I/O;
- safety/flush modes;
- record/range locking with retries;
- delete/undelete flags;
- index rebuild/check tools;
- full-screen field editing via Repertoire.

Core modules:

ModBase3	data file, record buffer, fields, memo, locks
DBIndexes	B-tree index, cursors, build/rebuild, auto update
DBFields	typed field access (date, logical, numeric, string)
DBRelate	relational lookup through an index
Locks	file/range locking and retry

DBIndexes uses 512-byte B-tree nodes, a configurable in-memory node buffer, a 128-byte key limit, and a maximum depth of 24. Hash is an opaque byte-key table from Repertoire, not part of the core dBase engine.

Good ideas to borrow:

- opaque DBFile/DBIndex handles;
- explicit schema/field descriptors;
- typed accessors over a raw record buffer;
- multiple secondary indexes and an update list;
- key-derivation callbacks for composite indexes;
- explicit safety/flush policy;
- range/record locking with retry policy;
- rebuild/check utilities;
- cursor operations (GoTop, GoBottom, Next, Prev);
- separate memo storage;
- Hash API shape for a generic hash table.

Why not port it:

- it is a dBase III compatibility project, not a general database layer;
- it depends on Repertoire modules (HandleIO, StringIO, LowLevel, VStorage, ErrorManager, SmartScreen, FAPI) and DOS/OS-2 services;
- compiler-independent initialization and per-compiler number handling add indirection V3 does not need;
- error handling is mostly integer OS codes plus interactive confirmation;
- no transaction/WAL story;
- dBase III storage/index formats are legacy;
- the license restricts source/unlinked-object redistribution.

Recommendation:

- use SQLite through a C binding for general database needs;
- or build a small typed table/index layer on BTreeFile;
- implement dBase III compatibility only if it is an explicit requirement, as a separate project with compatibility tests;
- use ModBase's schema/index/locking ideas as design input only.

11.6 Suggested order within communications/database work

1. Serial POSIX transport with pyserial-inspired semantics.
 2. Transport abstraction plus Loopback tests.
 3. Optional Socket transport.
 4. Optional XMODEM/YMODEM.
 5. Optional terminal emulation.
 6. Optional script VM.
 7. Database: SQLite binding or typed B-tree table layer.
 8. dBase III/ModBase compatibility only if required.
-

12. Network library design

V3 currently has no socket/network support. lib/iso/IOChan is a file/stdio-style channel API with no addresses, timeouts, non-blocking mode or datagrams, so it can be a convenience bridge but not the primary network API.

The network library should be layered:

Application

HTTP / WebSocket / custom protocols

|

TLS (optional)

|

Transport (connected byte stream or datagram)

|

Net (addresses, DNS, TCP/UDP sockets, listeners)

|

C shim / OS sockets

Transport is shared with the communications work:

- serial transport
- TCP transport
- loopback transport
- TLS transport wrapping another transport

Protocol modules should depend only on Transport, never directly on Socket.

12.1 Core API shape

Use opaque handles; do not expose file descriptors or sockaddr.

DEFINITION MODULE Net;

TYPE

Family = (IPv4, IPv6);

Kind = (Stream, Datagram);

ShutdownMode = (StopReading, StopWriting, StopBoth);

Status = (Ok, WouldBlock, Timeout, Closed, Refused,
Unreachable, BadAddress, NotFound,
PermissionDenied, OutOfResources,

```

        NetworkError, UnknownError);

Socket;
Address;

PROCEDURE Resolve(host : ARRAY OF CHAR; port : CARDINAL;
    family : Family; kind : Kind;
    VAR addr : Address) : Status;

PROCEDURE Listen(addr : Address; backlog : CARDINAL;
    VAR listener : Socket) : Status;

PROCEDURE Accept(listener : Socket;
    VAR conn : Socket;
    VAR peer : Address) : Status;

PROCEDURE Connect(addr : Address; timeoutMS : CARDINAL;
    VAR conn : Socket) : Status;

PROCEDURE Read(conn : Socket; VAR buf : ARRAY OF BYTE;
    max, timeoutMS : CARDINAL;
    VAR n : CARDINAL) : Status;

PROCEDURE Write(conn : Socket; VAR buf : ARRAY OF BYTE;
    len, timeoutMS : CARDINAL;
    VAR n : CARDINAL) : Status;

PROCEDURE WriteAll(conn : Socket; VAR buf : ARRAY OF BYTE;
    len, timeoutMS : CARDINAL;
    VAR written : CARDINAL) : Status;

PROCEDURE Shutdown(conn : Socket; mode : ShutdownMode);
PROCEDURE Close(VAR s : Socket);

PROCEDURE LocalAddress(s : Socket; VAR a : Address) : Status;
PROCEDURE PeerAddress(s : Socket; VAR a : Address) : Status;
PROCEDURE AddressToString(a : Address; VAR s : ARRAY OF CHAR);
PROCEDURE LastError(s : Socket) : CARDINAL;

```

Rules:

- Socket and Address are opaque.
- **ARRAY OF BYTE** is the binary buffer type; text helpers wrap it.
- Read and Write may return short counts.
- WriteAll loops over partial writes with a timeout.
- Timeouts are milliseconds, not floating-point seconds.
- WouldBlock is a normal result in non-blocking mode.
- Closed means orderly shutdown or peer close.
- LastError returns the OS error code for diagnostics.

12.2 Addresses and DNS

Do not model addresses as a single **CARDINAL** IPv4 value.

Use an opaque Address supporting:

- IPv4 and IPv6;
- textual host names;
- numeric addresses;
- port numbers;

- address-to-string and string-to-address conversion.

Use `getaddrinfo`, not `gethostbyname`. A resolver may return several addresses; Connect should be able to try them in order.

For listening sockets:

- use `AI_PASSIVE` for wildcard binds;
- handle IPv4/IPv6 dual-stack and optional `IPV6_V6ONLY`;
- use `SO_REUSEADDR` for restart;
- require an explicit wildcard bind for public interfaces.

A useful security default: servers bind to loopback unless the caller explicitly asks for a public interface.

Initial limitation: `getaddrinfo` is blocking and has no timeout. An asynchronous resolver can be added later.

12.3 TCP and UDP

TCP:

- Listen, Accept, Connect;
- stream reads/writes with timeouts;
- partial writes;
- graceful Shutdown;
- optional `SO_KEEPALIVE` and `TCP_NODELAY`;
- no automatic reconnection.

UDP:

- SendTo / RecvFrom;
- datagram boundaries preserved;
- maximum datagram size reported clearly;
- optional connected UDP;
- no stream semantics.

Keep TCP and UDP wrappers separate publicly even if they share a core socket module.

12.4 Concurrency and event loop

Do not require threads.

Phase 1:

- blocking sockets with explicit timeouts;
- poll in the C shim;
- one connection or request at a time is allowed.

Phase 2:

- non-blocking sockets;
- a PollSet abstraction:

TYPE

Event = SET OF (Readable, Writable, Error);

PROCEDURE Init(VAR p : PollSet);

PROCEDURE Add(VAR p : PollSet; s : Socket; events : Event);

PROCEDURE Remove(VAR p : PollSet; s : Socket);

PROCEDURE Wait(VAR p : PollSet; timeoutMS : CARDINAL;
VAR ready : CARDINAL) : Status;

PROCEDURE Events(p : PollSet; s : Socket) : Event;

Phase 3:

- integrate PollSet with PIM coroutines or an event loop;
- a connection handler can yield while waiting for I/O;
- no hidden thread pool.

Use poll first; epoll is an optimization, not a public API requirement.

12.5 Errors and security

Use status codes as the default; fatal errors and uncaught exceptions are not the normal path.

The shim must:

- retry EINTR;
- handle EINPROGRESS for non-blocking connect;
- translate EAGAIN/EWOULDBLOCK to WouldBlock;
- handle SIGPIPE with MSG_NOSIGNAL or an ignored signal;
- never leak descriptors on failed connect/accept/listen;
- preserve an OS error code for diagnostics.

12.6 TLS and HTTP

Do not implement TLS. Use OpenSSL or mbedTLS through a C shim.

TLS module:

Tls.Connect(transport, hostname, ...)

Tls.Read

Tls.Write

Tls.Close

Defaults:

- certificate verification enabled;
- SNI;
- hostname verification;
- no fallback to plaintext;
- clear errors for certificate failures.

HTTP is later:

- HTTP/1.1 client first;
- headers and Content-Length;
- chunked transfer encoding;
- keep-alive;
- redirects and size limits;
- no HTTP/2 initially.

WebSocket is later still.

12.7 Platform strategy

Start with POSIX/Linux:

- C shim over socket, bind, listen, accept, connect, send, recv, poll, getaddrinfo, setsockopt, shutdown, close;
- hide sockaddr_storage, addrinfo, fd_set and pollfd behind the shim.

Later add a Windows Winsock2 backend behind the same API.

Do not expose sockaddr, fd_set, pollfd or raw file descriptors in Modula-2.

12.8 Suggested module layout

runtime/syslib/net.c	POSIX C shim
lib/common/Net.def/.mod	addresses, DNS, sockets, status
lib/common/Transport.def/.mod	connected byte transport
lib/common/Poll.def/.mod	readiness/event API
lib/common/Loopback.def/.mod	deterministic tests
lib/common/Tls.def/.mod	optional, OpenSSL/mbedtls
lib/common/Http.def/.mod	optional, later

Possible names:

- Net — socket/address layer
- Tcp / Udp — protocol-specific wrappers
- Poll — event readiness
- Transport — shared with serial
- Tls — optional security layer
- Http — optional application protocol

12.9 Tests

Use loopback and ephemeral ports only:

- TCP client/server on 127.0.0.1 and ::1;
- UDP send/receive;
- partial read/write;
- connect timeout and refused connection;
- DNS failure and bad address;
- peer close and local shutdown;
- SIGPIPE / write-after-close;
- large transfers;
- multiple simultaneous sockets;
- poll timeout;
- listener restart with SO_REUSEADDR;
- TLS certificate failure and success later.

No external network is required for the suite.

12.10 Suggested order within network work

1. POSIX socket shim and Net status/address API.
2. TCP client/server.
3. UDP.
4. Poll and non-blocking sockets.
5. Transport integration with serial/loopback.
6. TLS via OpenSSL or mbedtls.
7. HTTP/1.1 client, then server.
8. WebSocket or other protocols if needed.

Do not start with HTTP or TLS. Do not expose raw sockets or sockaddr. Do not require threads. Reuse the Transport abstraction so serial, TCP, loopback and TLS share one protocol-facing interface.

13. Terminal escape libraries: VT100, VT52, ANSI, ReGIS

The existing MyWork/VT100 repository contains four useful terminal output libraries:

VT100	strict DEC VT100 escape sequences
VT52	strict DEC VT52 escape sequences

Ansi modern ANSI/xterm superset with colours and modes
Regis ReGIS vector graphics commands

There are 27 tests covering the output sequences, plus a SplitV1 helper used by the CPR (cursor-position report) test.

13.1 Important distinction: emitter, not emulator

These modules are **escape-sequence emitters**. They generate output such as ESC[2J, ESC[3;7H, and ReGIS commands. They do not parse incoming terminal data or maintain a screen buffer.

That is the right scope for an output library. A terminal **emulator** — needed to render incoming VT100/ANSI data in the communications stack — is a separate parser/screen model and should not be confused with these modules.

13.2 Porting issues

The modules were written for gm2 -fiso and currently depend on:

STextIO
NumberIO
ASCII
IO
termios
FIO

V3 has STextIO and NumberIO, but the other dependencies are either missing or not desirable in a portable library:

- IO and termios are used only by Unbuffered/Buffered (raw input for CPR replies);
- FIO.StdIn is used only for the same raw-input support;
- ASCII.esc can be replaced locally;
- NumberIO is PIM-oriented and should not be pulled into a common library just to emit decimal CSI parameters.

The current emitters also write directly to stdout through STextIO, which prevents using them with a serial or TCP transport. That is the main architectural change needed.

13.3 Recommended structure

Add a small output sink:

DEFINITION MODULE TermOut;

TYPE Output = **PROCEDURE**(s : **ARRAY OF CHAR**);

PROCEDURE SetOutput(out : Output);

PROCEDURE WriteString(s : **ARRAY OF CHAR**);

PROCEDURE WriteChar(c : **CHAR**);

PROCEDURE WriteCard(n : **CARDINAL**); (** decimal, no padding **)

END TermOut.

Default output can remain STextIO, but it can be redirected to:

- a serial Transport;
- a TCP Transport;
- a file or log;
- a test buffer;
- another terminal sink.

Then the terminal modules call TermOut.WriteString, TermOut.WriteChar and TermOut.WriteCard, not STextIO/NumberIO directly.

Move raw-mode input into a separate optional module:

Tty – termios save/restore, raw mode, non-blocking reads

VT100.Unbuffered/Buffered and the equivalents in VT52, Ansi and Regis should be removed from the emitters and replaced by Tty operations. This keeps the emitters pure and portable.

13.4 Suggested module layout

lib/common/TermOut.def/.mod	output sink + decimal/ESC helpers
lib/common/VT100.def/.mod	strict VT100 emitter
lib/common/VT52.def/.mod	strict VT52 emitter
lib/common/Ansi.def/.mod	ANSI/xterm emitter
lib/common/Regis.def/.mod	ReGIS emitter (optional)
lib/common/Tty.def/.mod	optional POSIX raw-mode input

lib/common is the simplest location because it is included by all dialects. A dedicated lib/term folder is possible but would require build-selection changes.

Regis is specialist vector graphics; include it but mark it optional. SplitV1 is test-only and should not become a core library module.

13.5 Tests

Convert the existing tests to headless byte tests:

```
VAR buf : ARRAY [0..1023] OF CHAR;
```

```
TermOut.SetOutput(Capture);  
VT100.CursorPosition(3, 7);  
Check(buf, "ESC[3;7H");
```

Benefits:

- no terminal is required;
- output is deterministic;
- the old od byte checks become assertions;
- tests run in the normal V3 suite.

Cover at least:

- VT100 SGR, cursor, erase, tabs, modes, charsets, reports;
- VT52 cursor, erase, mode, identification;
- ANSI SGR, colours, cursor, erase, modes, printing;
- ReGIS enter/exit, lines, circles, text, colours, macros.

The CPR/live-terminal test needs a pty or a real terminal. Keep it as an optional integration test, not part of the default headless suite.

13.6 Integration with communications and networking

TermOut makes the emitters usable with the communications design:

- serial transport writes through Transport;
- TCP transport writes through Transport;
- TermOut points at the transport writer;
- a future terminal emulator consumes incoming transport bytes and drives a screen model.

This gives a clean split:

```
terminal emitters -> TermOut -> Transport  
terminal emulator <- Transport <- incoming bytes
```

13.7 Licensing and provenance

The MyWork/VT100 repository currently has no explicit license file. Before vendoring the files into V3:

- confirm it is original/owned code, or
- keep it as a separate repository/submodule with a clear license, or
- add a V3-compatible license/provenance header.

The reference documents in docs/ are independent of the code.

13.8 First increment

1. Add TermOut.
2. Port VT100, VT52 and Ansi to TermOut.
3. Remove Unbuffered/Buffered from the emitters.
4. Add Tty separately.
5. Port Regis.
6. Add headless byte tests.
7. Keep CPR/live-terminal input tests optional.

No compiler changes are required.

14. C FFI foundation and GTK4/SQLite/SDL bindings

The planned GTK4, SQLite and SDL bindings are not just three wrapper libraries. They need a C ABI/FFI layer, a binding generator and pkg-config-aware build/link integration.

14.1 Current FFI state

V3 already has:

- DEFINITION/IMPLEMENTATION MODULE FOR "C";
- EXPORT UNQUALIFIED;
- EXTERNAL procedures;
- a special FROM libc IMPORT ... path with known prototypes and permissive varargs for unknown names;
- raw ADDRESS, POINTER T0 and pointer-sized procedure values;
- C varargs support for the special libc intrinsic path.

What is missing for large C APIs:

- explicit C link-name mapping;
- C-compatible record/union layout;
- a true 4-byte float;
- correct size_t/ssize_t widths;
- C bool vs V3 BOOLEAN;
- C arrays versus V3 descriptors;
- general foreign varargs;
- callback/trampoline support;
- a binding generator;
- multi-library build/link integration.

14.2 Underscores are already allowed

The scanner already accepts underscores:

```
ident = ( letter | "_" ) { letter | digit | "_" } .
```

This was added for __FILE__ / __LINE__ compatibility (commit a33fc7c). Names such as these already parse:

gtk_application_new
SDL_CreateWindow
sqlite3_open_v2

The docs in docs/features.md and docs/language-report.md still say “no underscores” and should be updated.

This reduces binding friction:

- direct C function names;
- C constants such as GTK_ALIGN_FILL;
- C struct fields such as SDL_Event.type;
- fewer renamed identifiers in generated bindings.

It does **not** solve:

- C struct layout and alignment;
- C unions and bitfields;
- C arrays vs V3 descriptors;
- char * and string ownership;
- true 4-byte float;
- size_t/ssize_t width;
- C bool;
- callbacks and closures;
- varargs;
- C macros/inline functions;
- build/link integration.

Policy recommendation:

- keep _ as a documented V3 extension;
- gate it behind strict PIM/ISO mode if strict compatibility matters;
- update the feature matrix and language report;
- still provide explicit EXTERNAL "c_name" for long/decorated names and C wrappers.

SymTab.Name is 64 bytes, so names longer than 63 characters can truncate; link names need an independent, larger mechanism.

14.3 C ABI/type gaps

The binding layer needs a C ABI profile:

TYPE

```
CInt    = INTEGER;      (* C int *)
CUInt   = CARDINAL;     (* C unsigned int *)
CLong   = LONGINT;      (* C long on LP64 *)
CULong  = LONGCARD;
CSize   = LONGCARD;     (* size_t, 64-bit on amd64 *)
CSSize  = LONGINT;      (* ssize_t *)
CFloat  = SHORTREAL;    (* needs a true 4-byte float *)
CDouble = REAL;
CBool   = INTEGER;      (* C int / gboolean / SDL_bool *)
CEnum   = INTEGER;      (* C enum, 32-bit *)
```

Required changes:

- true SHORTREAL/REAL32 4-byte float;
- correct CSIZE_T/CSSIZE_T widths on amd64;
- C bool (1 byte) separate from BOOLEAN (4 bytes);
- C enums represented as integer aliases, not Modula-2 enums;
- C long/unsigned long target-dependent aliases;
- C string and buffer ownership helpers;
- C void * as ADDRESS;

- C function pointers as compatible procedure values.

14.4 Structs, unions and arrays

V3 records are tightly packed, with no alignment/padding. One- dimensional array fields use V3 descriptors; C arrays are inline. Variant records are not C unions.

Two strategies:

1. **C accessor/wrapper strategy (recommended first)**
 - C shims allocate and manipulate C structs.
 - V3 sees opaque ADDRESS handles and gets/sets fields through C functions.
 - Avoids compiler layout work.
 - Works for SDL SDL_Event, SDL_AudioSpec, SQLite structs, GTK GdkRGBA/GValue.
2. **C-compatible record layout (long term)**
 - Add ALIGN, PACKED, UNION, inline arrays and bitfields.
 - Allows direct struct declarations.
 - Much larger and requires ABI layout tests.

For the first bindings, use wrappers. Do not start by making every V3 record C-compatible.

14.5 C FFI support library

Add a small C support module, for example lib/c/C:

- C scalar aliases;
- char * <-> ARRAY OF CHAR conversion;
- raw buffer with explicit length;
- allocation/free matching the C allocator;
- errno/GetLastError mapping;
- optional callback/trampoline helpers;
- opaque C-struct accessor conventions.

The exact API can evolve, but ownership must be explicit:

- borrowed string;
- owned string;
- C-allocated string that must be freed with free/g_free;
- pointer valid only for the duration of a call.

14.6 Binding generator

Recommended source by target:

Target	Best source	Strategy
GTK4	GObject Introspection .gir files	Generate types, enums, functions, signals, ownership
GLib/GObject/GIO	.gir files	Base bindings
SQLite	sqlite3.h	Small hand-written thin binding
SDL3	SDL3/SDL.h	Header parsing plus hand-written struct/callback wrappers

The generator should emit:

- .def declarations with safe Modula-2 names;
- EXTERNAL "c_name" link names;
- C shims for structs/unions, callbacks, varargs, GError**, GValue and string ownership;
- tests for each generated subsystem.

For GTK4, use the installed .gir files as the source of truth rather than raw headers. They include type names, transfer/ownership rules, nullable annotations, signals and properties.

Do not attempt full GTK4 generation first. Start with application, window, widget basics, buttons, labels, boxes, lists, files and signals.

14.7 Build and link integration

M2make currently supports one extra C file and links -lm only. It needs:

```
--cflag ...
--ldflag ...
--include ...
--c-source ...
--library ...
--pkg-config gtk4
--pkg-config sqlite3
--pkg-config sdl3
```

Requirements:

- multiple C shim files;
- multiple include directories;
- library order;
- -pthread where needed;
- pkg-config --cflags --libs;
- rpath handling;
- C shim/library flags in the ABI stamp.

GTK4 needs glib-2.0, gobject-2.0, gio-2.0, gtk4 and usually -pthread. SQLite and SDL3 are much simpler.

14.8 Callbacks and event loops

C callbacks need special handling:

- callbacks must be top-level, not nested procedures;
- no static-link closure can cross into C;
- Modula-2 exceptions must not escape a C callback;
- user data should be passed as ADDRESS and decoded by a dispatcher;
- GTK signals, SDL audio/timer callbacks and SQLite callbacks need trampolines or a global callback registry.

GTK4 also brings in GLib/GObject:

- reference counting (g_object_ref/g_object_unref);
- floating references;
- signals (g_signal_connect_data);
- GError** out-parameters;
- GValue and properties;
- GMainLoop/GtkApplication integration.

A V3 GTK binding must define its main-loop and ownership policy explicitly.

14.9 Target strategy

SQLite first

SQLite is the best first binding:

- opaque sqlite3 and sqlite3_stmt;
- C strings;
- sqlite3_int64;

- bind/column operations;
- callbacks;
- in-memory databases for tests.

A safe wrapper should hide statement finalization, `sqlite3_free`, `SQLITE_TRANSIENT` ownership, error strings and column lifetimes. SQLite is an excellent FFI conformance test for ints, int64, doubles, blobs, strings, pointers, callbacks and error codes.

SDL3 second

SDL3 needs:

- opaque window/renderer/texture pointers;
- `SDL_Event`, `SDL_Keysym`, `SDL_Rect`, `SDL_Color`, `SDL_AudioSpec`;
- `SDL_bool`, `Uint32`, `Sint32`, `Uint64`;
- true float or conversion wrappers;
- audio/timer callbacks;
- `SDL_Log` varargs;
- event loop.

Use C helpers to translate `SDL_Event` and `SDL_AudioSpec` to small V3 records. Prefer SDL3 unless SDL2 compatibility is required. Test with `SDL_VIDEODRIVER=dummy` and `SDL_AUDIODRIVER=dummy`.

GTK4 last

GTK4 is the largest:

- GLib base types and strings;
- GObject refcount and signals;
- GError;
- GValue and properties;
- GLib/GSList;
- GMainLoop/GtkApplication;
- many opaque types;
- ownership/transfer annotations.

GTK4 should come after the C FFI foundation, callbacks, build/link integration and GIR generator. A first binding should be a curated subset, not the full API.

14.10 Recommended order

1. C FFI foundation:
 - underscore policy/docs;
 - `EXTERNAL "c_name" / FFIDENT`;
 - true 4-byte float;
 - correct `size_t`/`ssize_t`;
 - C string/buffer/error helpers;
 - C struct accessor strategy.
2. pkg-config-aware build integration and ABI stamping.
3. FFI conformance tests.
4. SQLite binding plus safe wrapper.
5. SDL3 binding plus event/audio helpers.
6. GIR-based generator for GLib/GObject.
7. GTK4 binding subset and headless tests.

Bindings should live outside the core library:

<code>lib/c/</code>	C FFI helpers
<code>bindings/sqlite3/</code>	SQLite binding and wrapper
<code>bindings/sdl3/</code>	SDL3 binding and wrapper
<code>bindings/gtk4/</code>	GLib/GObject/GTK4 bindings

15. Audit of existing gm2 bindings: SQLite, SDL, GTK4

The existing GNU Modula-2 binding projects are a strong foundation, but they are gm2-targeted and are not drop-in for V3.

15.1 m2SQLITE

Contents:

- src/SQLite.def — thin FOR "C" binding of the core SQLite API
- lib/SQLiteUtils — pure-M2 helpers
- 7 tests
- examples: version, hello_db
- showcase_all
- open/close, exec, prepare/step/finalize, bind/column access, callbacks, BLOB I/O, backup, extended result codes and error strings

Quality:

- good API coverage;
- sensible opaque handles (ADDRESS);
- sqlite3_int64 as LONGINT;
- double as REAL;
- top-level callback procedure types;
- in-memory SQLite makes testing easy.

Main V3 blocker: the C string boundary.

The binding declares:

```
PROCEDURE sqlite3_open (filename: ARRAY OF CHAR;  
                        VAR db: DbHandle) : INTEGER;
```

gm2's FOR "C" passes ARRAY OF CHAR as char *. V3 currently passes the **descriptor address** in a normal external call. A generated V3 call was checked:

```
call $check_str(1 $CStr_str0)
```

\$CStr_str0 is { length, bytes... }, not the first character. A C function expecting char * sees the length word instead.

This must be fixed before SQLite, SDL or GTK can bind to C string parameters.

SQLite is otherwise the best first binding.

15.2 m2SDL/SDL3

There are two independent bindings:

- m2SDL/SDL2 — older subset
- m2SDL/SDL3 — current, more complete

Use SDL3 for new work; keep SDL2 only for legacy compatibility.

SDL3 contents:

- 12 .def modules
- SDL3Utils
- 7 tests
- 4 examples
- 7 showcases
- pure-M2 SDL3Rect and SDL3Key
- opaque window/renderer handles as ADDRESS
- event handling through a 128-byte buffer with decoded offsets
- video, renderer, events, clipboard, filesystem, power, locale

Good design points:

- avoids modelling `SDL_Event` as a V3 variant record;
- decodes known event fields from a raw buffer;
- uses `SDLXxx` type aliases;
- has dummy-driver tests (`SDL_VIDEODRIVER=dummy`, `SDL_AUDIODRIVER=dummy`).

V3 blockers:

1. `SHORTREAL` is 4 bytes in gm2 but currently aliases 8-byte `REAL` in V3. This breaks `SDL_FRect`, `SDL_FPoint`, float draw coordinates and `SetRenderDrawColorFloat`.
2. `C bool` is 1 byte. V3 `BOOLEAN` is 4 bytes, so `[ BOOLEAN ]` results and bool parameters need a `CBool/BYTE` mapping or C wrappers.
3. The `ARRAY OF CHAR` descriptor/string ABI issue above.
4. The event decoder uses fixed offsets 0, 28 and 36 and little-endian LE32; correct on tested Linux/amd64 but fragile. A C helper using `offsetof` or a C event decoder is safer.
5. Integer `SDL_Point/SDL_Rect` records match C layout; float records do not until true float32 exists.

15.3 m2-GTK4

This is the largest and most complete binding project.

Contents:

- 432 `.def` modules in `src/`
- `GtkUtils` and `GtkClosures` helpers in `lib/`
- C shim `lib/m2gtkshim.c`
- 45 tests
- 30 examples
- 19 demo modules plus the `m2 gtk-demo` browser
- `tools/gir2def.py` generator
- `tools/gir_verify.py` verifier

Generator status:

- 301 type modules and 3,719 procedures for Gtk-4.0;
- 10,256 identifiers verified against installed libraries;
- generated files are intentionally review/cherry-pick material, not a replacement for the hand-written `src/` modules.

The architecture is good:

- one FOR "C" module per GTK/GDK/Gsk/Gio/Adw class;
- opaque objects as `ADDRESS`;
- `GtkClosures` provides per-connection callback contexts and destroy handlers;
- `m2gtkshim.c` provides C-side `GObject` subclasses for custom widgets, paintables, list models and layout managers.

`GtkClosures` + `m2gtkshim.c` is exactly the kind of trampoline/shim layer V3 will need.

V3 blockers:

1. Ignorable-result syntax. GTK modules use:

```
PROCEDURE gtk_widget_get_visible (...) : [ INTEGER ];
```

V3 currently rejects `[ T ]` (invalid `ProcHeading`). Either add this extension or strip it while porting.

2. Varargs. GTK uses declarations such as:

```
PROCEDURE g_object_new (...; ...) : ADDRESS;
```

```
PROCEDURE g_object_set (...; ...);
```

```
PROCEDURE gtk_alert_dialog_new (format: ARRAY OF CHAR; ...) : ...;
```

V3 has no general . . . declarations/calls. Use typed C wrappers or avoid those APIs in the first subset.

3. C string boundary, as for SQLite/SDL.
4. True 4-byte float is needed for GdkRGBA, Graphene, Gsk path APIs, GtkFrame alignment and many Adw widgets.
5. Value structs:
 - GdkRectangle (four int) likely matches V3 tight records;
 - GdkRGBA (four gfloat) does not until float32 is fixed;
 - GValue is 24 bytes and may match;
 - GtkTreeIter, PangoRectangle, Graphene records need layout audits.
6. GError**, GValue and property APIs need C helpers or explicit wrapper conventions.
7. gir2def.py is intentionally coarse: value structs become opaque ADDRESS, GError** and varargs are dropped, and signals/properties are not generated. It is a bootstrap tool, not a complete GTK API.
8. GTK needs pkg-config gtk4, -pthread, optional libadwaita / GtkSourceView / libspelling / libpanel, and multiple C shims. V3's M2make currently supports only one extra C file and links -lm only.

15.4 Reuse plan

Directly reusable:

- SQLite .def and SQLiteUtils as the first port;
- SDL3 video/render/events/rect/key definitions and test/style;
- GTK GtkClosures, GtkUtils design and m2gtkshim.c;
- GIR generator/verifier after adapting V3 type mapping;
- examples/tests as acceptance tests.

Not directly reusable:

- gm2-specific [T] result syntax without a V3 parser addition;
- . . . varargs declarations without wrappers;
- gm2's 4-byte SHORTREAL assumption;
- gm2's C bool/string ABI assumptions;
- value structs whose layout depends on C padding/alignment;
- one --extra C file build model.

15.5 Porting order

1. V3 C FFI foundation:
 - foreign char * / buffer ABI;
 - true SHORTREAL/REAL32;
 - C bool;
 - [T] or stripped definitions;
 - varargs wrapper strategy;
 - EXTERNAL "c_name" and long names;
 - pkg-config build integration.
2. Port SQLite:
 - SQLite.def + SQLiteUtils;
 - in-memory tests;
 - callback tests;
 - blob/bind/column tests.
3. Port SDL3:
 - fix float/bool/string ABI;

- keep the event-buffer design or replace with a C decoder;
- dummy-driver tests.

4. Port a small GTK4 subset:

- GLib, GObject, Gio, GtkApplication, GtkWindow, GtkButton, GtkLabel, GtkBox;
- port GtkUtils, GtkClosures, m2gtkshim.c;
- add a few tests and examples;
- expand via GIR.

5. Keep SDL2 separate; do not mix SDL2 and SDL3 bindings.

6. Put bindings outside the compiler core:

```
bindings/sqlite3/
bindings/sdl3/
bindings/gtk4/
```

Each package owns its .def/.mod, C shim, tests, examples, license/provenance and V3 build manifest.

Do not vendor upstream GTK/SDL/SQLite source trees into V3; keep them as read-only references or system packages.

15.6 Assessment

- m2SQLITE: best first target, but string ABI is a prerequisite.
- m2SDL3: solid second target after float32, C bool and string ABI.
- m2-GTK4: excellent long-term asset; do not port all 432 modules before the FFI/parser work.

The efficient route is: fix the C FFI, port SQLite, port SDL3, then port a small GTK4 subset using the existing closure/shim architecture.

16. GTK4 OOP facade: limits and recommended layering

V3 classes can make GTK4 application code look much more like Modula-2, but they should be an **optional high-level facade**, not the implementation mechanism for the binding itself. Directly mapping V3 classes onto GObject classes is not viable: V3's Clarion-form classes and GObject's C object model have incompatible layouts and lifecycles.

16.1 Three layers

The binding should distinguish three layers:

1. raw GTK4 C binding -- ADDRESS handles and free procedures
2. typed/opaque wrapper -- Modula-2 types and module procedures
3. optional OOP facade -- CLASS Window, CLASS Button, methods

The existing m2-GTK4 is layer 1. That is the right canonical low-level layer: generated or reviewed FOR "C" modules with opaque ADDRESS handles.

V3 classes belong at layer 3, for application code. A typed opaque layer can sit between them, especially where class fields would expose implementation details.

16.2 Why V3 classes are not GObject classes

V3 classes:

- are records with fields;
- have a hidden THIS receiver in methods;
- have a hidden V3 vtable pointer for virtual methods;
- use V3 method-call and subtype rules;

- have value semantics when used as values;
- have no destructors or finalizers;
- support single inheritance only.

GObject has:

- its own GObject header, type system and class/instance structures;
- reference counting and floating references;
- signals and properties;
- interfaces;
- GObjectClass/GtkWidgetClass vfunc tables;
- C-side construction and finalization.

A V3 class instance cannot be passed to C as a GtkWidget *. The V3 wrapper and the GTK object must remain separate:

```
CLASS Window;
  handle : ADDRESS;    (* the real GtkWidget *, not the V3 object *)
  ...
END Window;
```

C never sees the V3 object. It receives only the handle stored inside it. Direct GObject subclassing still requires C shims such as the existing m2gtkshim.c.

16.3 Ownership and reference counting

A V3 class copy copies its fields. If a wrapper contains a handle and an owner flag, copying the wrapper can create two apparent owners of the same GObject and lead to a double Unref.

Use pointer wrappers and explicit lifetime operations:

TYPE

```
CLASS Object;
  handle : ADDRESS;
  owned   : BOOLEAN;
  PROCEDURE Ref;
  PROCEDURE Unref;
END Object;

CLASS Widget (Object);
  PROCEDURE SetVisible (on : BOOLEAN);
END Widget;

CLASS Window (Widget);
  PROCEDURE SetTitle (s : ARRAY OF CHAR);
  PROCEDURE Present;
END Window;

WindowPtr = POINTER TO Window;
```

Constructors should return pointer wrappers, for example NewWindow (app) : WindowPtr.

Rules:

- distinguish owned from borrowed handles;
- sink floating references where necessary;
- DISPOSE frees only the V3 wrapper, never the GObject;
- call Unref/Close explicitly before disposing the wrapper;
- do not rely on V3 BEGIN ... END as a per-object constructor: a class initialisation body runs once at startup, not for each object;
- keep wrapper objects on the main GTK thread.

V3 has no destructors, so ownership must be documented and tested, not inferred from scope exit.

16.4 Encapsulation

V3 classes have no private fields. A class declared in a definition module exposes its fields to clients, including the GTK handle and owner flag. This weakens the abstraction.

Options:

1. accept the exposed handle and document it as an internal field;
2. use an opaque handle type plus module procedures for the most sensitive objects;
3. add private class fields or opaque classes to V3 later.

A true private-field facility is compiler work and should not block the first GTK4 facade. For a small facade, exposing handle as an implementation detail may be acceptable; for a public library API, an opaque wrapper is cleaner.

16.5 Inheritance and GTK interfaces

V3 single inheritance can model GTK's primary class chain:

```
GObject -> GtkWidget -> GtkWindow -> GtkApplicationWindow
GObject -> GtkWidget -> GtkButton
GObject -> GtkWidget -> GtkBox
```

It cannot model GTK's interfaces. GTK and Gio use interfaces such as `GtkAccessible`, `GtkBuildable`, `GtkNative`, `GtkRoot`, `GtkShortcutManager`, `GListModel` and `GtkTreeModel`.

Do not encode interfaces as extra V3 parents: V3 rejects multiple parents. Provide interface capabilities as separate modules or free-function groups:

```
Accessible.SetLabel
Buildable.AddFromString
Native.GetSurface
```

Only the main widget chain should use class inheritance. Services and non-widget objects such as `GtkApplication`, `GtkBuilder`, models and file dialogs can use their own wrapper roots or opaque handles.

16.6 Constructors and properties

GTK construction is C-style:

```
gtk_application_window_new (app)
g_object_new (type, "property", value, ...)
```

V3 has no named-property arguments or general varargs. Constructors in the facade should be ordinary module procedures:

```
PROCEDURE NewWindow (app : Application) : WindowPtr;
PROCEDURE NewButton (label : ARRAY OF CHAR) : ButtonPtr;
```

Property-heavy construction should use C wrappers or `g_object_new_with_properties`, not an invented V3 constructor syntax. Keep `gtk_*_new` and `g_object_new` wrappers in the raw binding; the OOP facade should call them rather than duplicate them.

16.7 Signals and callbacks

Do not pass a V3 instance method directly to `g_signal_connect_data`. A V3 method has a hidden receiver and is not a C function pointer with GObject's callback ABI.

Use the existing `GtkClosures` pattern:

- one top-level trampoline per callback signature;
- a wrapper/context pointer as `gpointer` user data;
- an explicit destroy handler;
- optional V3 virtual dispatch inside the trampoline.

For example, a clicked handler can receive the wrapper pointer as user data, cast it back to `ButtonPtr`, and invoke a V3 method or virtual method. The trampoline must still be a top-level procedure with the C callback signature.

The same rule applies to `GtkApplication` activate/startup handlers, `GtkDrawingArea` draw functions and model callbacks.

16.8 GObject subclassing

Custom widgets, custom `GdkPaintable` implementations, custom `GtkLayoutManagers` and custom `GListModel`s still require C:

- register the `GObject` type;
- provide `class_init/instance_init`;
- override vfuncs;
- handle finalization and properties.

V3 classes cannot do this. Keep `m2gtkshim.c` for subclassing. The OOP facade can wrap the resulting C object afterward, but it cannot replace the shim.

16.9 Recommended package layout

<code>bindings/gtk4/raw/</code>	FOR "C" modules, ADDRESS handles
<code>bindings/gtk4/wrap/</code>	optional typed wrappers / opaque types
<code>bindings/gtk4/oop/</code>	optional CLASS facade
<code>bindings/gtk4/shim/</code>	<code>m2gtkshim.c</code> , <code>GtkClosures</code>

The raw binding remains canonical. The OOP facade should cover only a small, useful application subset:

`Object`, `Widget`, `Window`, `ApplicationWindow`,
`Button`, `Label`, `Box`, `Entry`, `ListBox`, `DrawingArea`,
`Application`, `Builder`

Do not create a parallel class hierarchy for all 432 GTK/GDK/Gsk/Gio/Adw modules. Keep uncommon APIs at the raw or typed-wrapper layer. The GIR generator can help produce raw modules; the OOP facade should be hand-curated because ownership and API shape need judgement.

16.10 Is it more “Modula-2”?

Not necessarily. Classic Modula-2 style is primarily:

- modules as namespaces;
- opaque types;
- explicit procedures;
- strong typing;
- explicit ownership.

V3 CLASS is a Clarion-form extension, not ISO Modula-2 OO. An OOP facade can be nicer for application code, but it is not automatically more standard. The most idiomatic design is probably:

opaque handle types + module procedures

with classes as an optional convenience layer for users who prefer:

```
win^.SetTitle("Demo");  
btn^.OnClicked(OnQuit);
```

16.11 Bottom line

- Use V3 OOP for a high-level GTK4 facade, not for the raw binding.
- Keep ADDRESS-based FOR "C" modules as the canonical layer.
- Use POINTER TO CLASS wrappers holding a C handle and ownership flag.

- Add explicit Ref/Unref/Close; do not rely on DISPOSE.
- Model only GTK's primary single-inheritance chain with classes.
- Keep GTK interfaces as separate module capabilities.
- Use GtkClosures/top-level trampolines for signals; never pass V3 methods directly to C.
- Keep m2gtkshim.c for GObject subclassing.
- Wrap a small common subset first; do not mirror all 432 modules.

This gives a more Modula-2 application API without pretending that V3 classes are GObject classes.

17. Suggested order of future work

1. TBITSIZE (small, standard-facing).
2. ADDADR/SUBADR/DIFADR (small, ISO SYSTEM).
3. Set ordering (bounded gm2 compatibility; fixes largeset5).
4. Complex types: choose ISO built-ins versus a nonstandard value-record library; remove the misleading COMPLEX* aliases.
5. Pragmas: native < * . . . * > parser/IR plus a low-risk semantic subset; later add the TopSpeed adapter and ABI/layout pragmas.
6. UTF-8 string/character libraries: fix Utf8.Encode, add byte-string and UString modules, then UTF-8 input.
7. Collections: Vector, Deque/Stack, HashMap, then optional BTreeMap; persistent BTreeFile last.
8. Terminal emitters: TermOut, VT100, VT52, Ansi, optional Regis; Tty and headless tests.
9. C FFI foundation and bindings: underscore policy, link names, float32, size_t, C strings/buffers, struct/callback wrappers, pkg-config; SQLite, then SDL3, then GTK4.
10. Port the existing gm2 bindings: SQLite first, SDL3 second, GTK4 subset last; keep them as separate bindings/ packages.
11. Add an optional GTK4 OOP facade over the raw binding: small Object/Widget hierarchy, pointer wrappers, explicit Ref/Unref/Close, GtkClosures trampolines; no direct GObject subclassing.
12. Communications: Serial/Transport first; protocols and terminal emulation optional.
13. Network: Net/Transport, TCP/UDP, Poll, then optional TLS/HTTP.
14. Database: SQLite binding or typed B-tree table layer; dBase III only if required.
15. ISO COROUTINES subset, then interrupt/protection layer.
16. Dynamic SET CONST and deep composite copies.
17. Parser error recovery and diagnostics.

External assembly needs no compiler work; document it as the supported low-level escape hatch.