

Programmer's Conversion Guide

Better scalar scope, wider loop completion, bounded UTF-8 file I/O, and the refined PySide6 workspace.



WORKS / CONVERTED

DOES NOT WORK / REJECTED

REFERENCE PROFILE

CONVERTER CONTRACT 0.16.0
SOURCE GRAMMAR Python 3.11
TARGET deterministic portable ISO C11 source
GUIDE EDITION scope, loop, file-I/O, and PySide6 release

Document map

Major sections are bookmarked and indexed below. Every numbered source unit is also identified as converted or deliberately rejected.

RESULT	MEANING
WORKS / CONVERTED	Accepted source with complete ISO C11 and no diagnostics.
DOES NOT WORK / REJECTED	Rejected source with its exact primary PYC diagnostic.

Purpose and reading contract

What changed in 0.16.0

Part I - Branch-aware scalar scope

Part II - Loop completion and wider iteration

Part III - Bounded UTF-8 file I/O

Part IV - File status, ownership, and resource contract

Resource ceilings

Part V - Diagnostics quick reference

Part VI - Stable conversion envelope

Part VII - PySide6 workspace improvements

Release validation

Final checklist

3

4

5

12

17

30

31

32

33

34

34

35

Purpose and reading contract

This guide is the Programmer's Conversion Guide for PyCForge 0.16.0. It explains the newly promoted conversion surfaces and the boundaries that keep generated C deterministic, bounded, and independently auditable.

- Branch-aware publication of compatible scalar locals after exhaustive conditionals.
- while, range, fixed-list, fixed-tuple, and fixed-dictionary loop else suites.
- Exact ownership of break, continue, and return relative to loop completion.
- Bounded UTF-8 whole-file reads and exact UTF-8 writes with deterministic cleanup.
- Explicit C ABI, resource ceilings, diagnostics, and caller responsibilities.
- PySide6 editor font controls, auto-indent, and readable generated-C mapping display.

IMPORTANT

Treat every numbered example as a separate source file. PyCForge rejects the entire conversion unit when any function in it is unsupported; it never publishes partial C.

Validation model

The guide examples are machine-checked through the converter without executing the submitted Python, compiling generated C, linking it, or loading it. The committed PDF is also checked against a deterministic rebuild.

What changed in 0.16.0

AREA	NEW CONVERSION SURFACE	FAIL-CLOSED BOUNDARY
Scalar scope	Compatible locals defined on every reachable if / elif / else arm may survive the branch.	Partial definition, read-before-store, conflicting representations, containers, records, and loop mutation reject.
Loop completion	else is supported for while, range, and fixed list / tuple / dictionary loops.	Only an owned break suppresses else; unsafe zero-iteration target reads reject.
File I/O	One exact read or write inside with <code>open(..., encoding='utf-8', newline='')</code> .	No append/binary mode, partial read, handle escape, dynamic write expression, or nested control-flow session.
Workspace	Required PySide6 UI, 8-48 point editor sizing, auto-indent, and passive generated-C text.	Generated C remains read-only; explicit mapping navigation remains available.

The new surfaces are carried by active 0.16.0 plan, C-IR, renderer, generated-C, decision-trace, summary, fact-table, runtime-policy, and rule-set identities. Historical 0.14.3 conversion remains available only through its explicit compatibility profile.

The PySide6 worker boundary uses `pycforge.worker-protocol/0.2` and preserves the active runtime-policy identity in its authenticated request envelope.

Part I - Branch-aware scalar scope

PyCForge may now publish a new scalar local after an exhaustive conditional. The proof is reachability-aware and independent of lowering.

The rule is intentionally selective. A candidate must have one compatible scalar representation, a store on every reachable publishing arm, and no earlier read or unsupported lifetime interaction.

- Supported candidate categories: proved bool, int, float, and str scalars.
- Containers, records, nested capture, global/nonlocal directives, and opaque uses are excluded.
- Unsafe source receives a dedicated PYC41xx diagnostic before C-IR publication.

W01 Exhaustive branch-local scalar

WORKS / CONVERTED

NO DIAGNOSTICS

Both reachable branches define the same integer representation, so PyCForge proves one function-scoped C local and publishes it after the branch.

PYTHON SOURCE

```
def choose(flag: bool) -> int:
    if flag:
        value = 41
    else:
        value = 1
    return value
```

Implementation notes

- The declaration is hoisted in C; Python evaluation order is unchanged.
- The binding becomes readable only after the exhaustive branch completes.

W02 Exhaustive if / elif / else scalar

WORKS / CONVERTED

NO DIAGNOSTICS

Every reachable arm defines result as an integer before the final read.

PYTHON SOURCE

```
def classify(value: int) -> int:
    if value < 0:
        result = -1
    elif value == 0:
        result = 0
    else:
        result = 1
    return result
```

Implementation notes

- Nested conditional arms are checked for complete reachability.
- Compatible bool, int, float, and str scalar representations are considered.

W03 New scoped scalar beside an unrelated loop

WORKS / CONVERTED

NO DIAGNOSTICS

An unrelated bounded loop does not prevent a later exhaustive scalar proof.

PYTHON SOURCE

```
def total_or_code(flag: bool, count: int) -> int:
    total = 0
    for index in range(count):
        total = total + index
    if flag:
        code = 1
    else:
        code = 2
    return code
```

Implementation notes

- The candidate scalar itself must not be mutated inside a loop.
- Loop targets retain their own bounded lifetime rules.

N01 Non-exhaustive branch-local scalar

DOES NOT WORK / REJECTED

PRIMARY DIAGNOSTIC PYC2940

The false path reaches return without storing value.

PYTHON SOURCE

```
def incomplete(flag: bool) -> int:
    if flag:
        value = 1
    return value
```

Implementation notes

- Add an else arm or initialize value before the conditional.

N02 Conflicting scalar representations

DOES NOT WORK / REJECTED

PRIMARY DIAGNOSTIC PYC2943

The branches disagree about the proved C representation.

PYTHON SOURCE

```
def conflict(flag: bool) -> int:
    if flag:
        value = 1
    else:
        value = True
    return value
```

Implementation notes

- Use the same scalar category in every publishing branch.

N03 Container as a branch-hoisted candidate

DOES NOT WORK / REJECTED

PRIMARY DIAGNOSTIC PYC3403

Phase 16 scope publication is deliberately limited to proved scalar bindings.

PYTHON SOURCE

```
def containers(flag: bool) -> int:
    if flag:
        value = [1]
    else:
        value = [2]
    return 0
```

Implementation notes

- Declare fixed containers outside conditional control flow.

Part II - Loop completion and wider iteration

Loop else suites now lower for while, range, and bounded fixed-container loops. The analyzer assigns every break and continue to its exact owning loop.

A completion flag exists only when the loop owns a break. Natural exhaustion or a false while condition enters else; continue preserves completion; return bypasses it.

- Fixed list, tuple, and dictionary iteration remains bounded and immutable.
- Nested loop breaks never suppress the wrong else suite.
- range lowering checks the signed 64-bit increment boundary before adding the step.

W04 while ... else on natural completion

WORKS / CONVERTED

NO DIAGNOSTICS

The else suite runs when the condition naturally becomes false.

PYTHON SOURCE

```
def drain(count: int) -> int:
    while count:
        count = count - 1
    else:
        return 1
    return 0
```

Implementation notes

- No completion flag is emitted when the loop owns no break.
- A return inside the loop bypasses the post-loop else naturally.

W05 range ... else with an owned break

WORKS / CONVERTED

NO DIAGNOSTICS

Only a break owned by this loop suppresses its else suite.

PYTHON SOURCE

```
def find_three(limit: int) -> int:
    result = -1
    for index in range(limit):
        if index == 3:
            result = index
            break
    else:
        result = 0
    return result
```

Implementation notes

- PyCForge emits a completion flag only because this loop owns a break.
- Breaks inside nested loops do not clear the outer loop's flag.

W06 Fixed-list iteration with else

WORKS / CONVERTED

NO DIAGNOSTICS

Fixed list, tuple, and dictionary iteration share bounded completion proofs.

PYTHON SOURCE

```
def sum_then_mark() -> int:
    values = [1, 2, 3]
    total = 0
    for value in values:
        total = total + value
    else:
        total = total + 1
    return total
```

Implementation notes

- continue preserves natural completion and does not suppress else.
- range increments are hardened against signed 64-bit overflow.

N04 Loop target read from else

DOES NOT WORK / REJECTED

PRIMARY DIAGNOSTIC PYC2941

A zero-iteration range never binds index, so the else read is unsafe.

PYTHON SOURCE

```
def last_index(limit: int) -> int:
    for index in range(limit):
        continue
    else:
        return index
    return 0
```

Implementation notes

- Store a separately initialized result when an empty range is possible.

Part III - Bounded UTF-8 file I/O

Phase 16 adds a deliberately narrow, ownership-complete file profile. A supported with statement owns one fresh handle, one direct operation, and one exactly-once close.

The generated helpers use an explicit non-null `int64_t` status pointer. Reads return unique owned storage to the C caller; writes borrow their path and text operands.

- Exact context: `open(path, 'r' or 'w', encoding='utf-8', newline='')`.
- Read form: direct return, or assignment followed by immediate return after close.
- Write form: one direct string Name or string literal; the Python byte count is discarded.
- Path and write text C strings must be non-null, NUL-terminated, and valid UTF-8.

W07 Direct bounded UTF-8 read

WORKS / CONVERTED

NO DIAGNOSTICS

The generated entry point returns a uniquely owned char pointer and reports file failures through a required non-null `int64_t` status pointer.

PYTHON SOURCE

```
def load(path: str) -> str:
    with open(path, 'r', encoding='utf-8', newline='') as handle:
        return handle.read()
```

Implementation notes

- The C caller frees the returned buffer after use.
- Invalid UTF-8, embedded NUL content, and reads over the configured limit fail.

W08 Assigned read followed by immediate transfer

WORKS / CONVERTED

NO DIAGNOSTICS

The owned read result crosses the close boundary exactly once and is immediately returned.

PYTHON SOURCE

```
def load_named(path: str) -> str:
    with open(path, 'r', encoding='utf-8', newline='') as handle:
        data = handle.read()
    return data
```

Implementation notes

- Aliasing or observing data before its return is rejected.
- The handle is closed exactly once on every helper outcome.

W09 Exact UTF-8 write from a string parameter

WORKS / CONVERTED

NO DIAGNOSTICS

The text is borrowed, written exactly, and the Python write count is deliberately discarded.

PYTHON SOURCE

```
def save(path: str, text: str) -> int:
    with open(path, 'w', encoding='utf-8', newline='') as handle:
        handle.write(text)
    return 0
```

Implementation notes

- write accepts a direct proved string Name or a string literal.
- The caller supplies NUL-terminated valid UTF-8 text with no embedded NUL.

W10 Exact UTF-8 write from a literal

WORKS / CONVERTED

NO DIAGNOSTICS

A direct string literal is an accepted stable write operand.

PYTHON SOURCE

```
def mark_ready(path: str) -> int:
    with open(path, 'w', encoding='utf-8', newline='') as handle:
        handle.write('ready')
    return 0
```

Implementation notes

- A literal containing an embedded NUL is rejected before lowering.

W11 Sequential file sessions

WORKS / CONVERTED

NO DIAGNOSTICS

Multiple sessions are supported when each one has a closed, single-operation lifecycle.

PYTHON SOURCE

```
def save_two(first: str, second: str, text: str) -> int:
    with open(first, 'w', encoding='utf-8', newline='') as left:
        left.write(text)
    with open(second, 'w', encoding='utf-8', newline='') as right:
        right.write(text)
    return 0
```

Implementation notes

- The default operation ceiling is explicit and independently validated.
- No handle may escape, alias, rebind, or be used after its with suite.

W12 File output followed by branch-local result selection

WORKS / CONVERTED

NO DIAGNOSTICS

File effects and later exhaustive scalar publication compose when their ownership regions do not overlap.

PYTHON SOURCE

```
def save_code(path: str, text: str, flag: bool) -> int:
    with open(path, 'w', encoding='utf-8', newline='') as handle:
        handle.write(text)
    if flag:
        code = 1
    else:
        code = 2
    return code
```

Implementation notes

- A file with block itself remains a top-level function statement.
- No source Python or generated C is executed during conversion validation.

N05 Append, binary, or implicit text options

DOES NOT WORK / REJECTED

PRIMARY DIAGNOSTIC PYC3903

The Phase 16 profile accepts only literal r or w mode with exact UTF-8 and newline settings.

PYTHON SOURCE

```
def append(path: str, text: str) -> int:
    with open(path, 'a', encoding='utf-8', newline='') as handle:
        handle.write(text)
    return 0
```

Implementation notes

- Use mode 'w', encoding='utf-8', and newline="" exactly.

N06 Two operations in one with suite

DOES NOT WORK / REJECTED

PRIMARY DIAGNOSTIC PYC3901

One file session owns exactly one direct read or write operation.

PYTHON SOURCE

```
def twice(path: str, text: str) -> int:
    with open(path, 'w', encoding='utf-8', newline='') as handle:
        handle.write(text)
        handle.write(text)
    return 0
```

Implementation notes

- Use separate sequential with statements when multiple operations are needed.

N07 Sized read or readline

DOES NOT WORK / REJECTED

PRIMARY DIAGNOSTIC PYC3905

The supported operation is a zero-argument bounded read of the whole file.

PYTHON SOURCE

```
def partial(path: str) -> str:
    with open(path, 'r', encoding='utf-8', newline='') as handle:
        return handle.read(10)
```

Implementation notes

- readline, iteration over a handle, seek, and partial reads remain outside the profile.

N08 Effectful write argument

DOES NOT WORK / REJECTED

PRIMARY DIAGNOSTIC PYC3905

A write operand is not evaluated before context entry unless it is a direct proved Name or literal.

PYTHON SOURCE

```
def make_text(flag: bool) -> str:
    while flag:
        continue
    return 'x'

def save_call(path: str, flag: bool) -> int:
    with open(path, 'w', encoding='utf-8', newline='') as handle:
        handle.write(make_text(flag))
    return 1
```

Implementation notes

- Compute and prove the text in an earlier statement, then pass its direct name.

N09 Handle use after close

DOES NOT WORK / REJECTED**PRIMARY DIAGNOSTIC PYC3906**

The file handle is context-owned and cannot escape or be used after its close boundary.

PYTHON SOURCE

```
def reuse(path: str, text: str) -> int:
    with open(path, 'w', encoding='utf-8', newline='') as handle:
        handle.write(text)
    handle.write(text)
    return 0
```

Implementation notes

- Aliases, explicit close calls, rebinding, and returning the handle are also rejected.

N10 Calling a source-defined file-effect function

DOES NOT WORK / REJECTED

PRIMARY DIAGNOSTIC PYC3908

The initial explicit-status ABI exposes file-effect functions as generated C entry points, not Python-to-Python calls.

PYTHON SOURCE

```
def load(path: str) -> str:
    with open(path, 'r', encoding='utf-8', newline='') as handle:
        return handle.read()

def use(path: str) -> str:
    return load(path)
```

Implementation notes

- Call the generated entry point from C and pass a non-null `int64_t` status pointer.

Part IV - File status, ownership, and resource contract

Generated file-effect entry points add a required non-null `int64_t * status` parameter. After the null guard, each entry point initializes status to zero before any source-controlled return or evaluation; a helper failure overwrites it with the exact nonzero class.

STATUS	MEANING	OWNERSHIP / CLEANUP
0	success	Read buffer transfers to caller; write has no owned result.
1	open error	No open handle or owned result remains.
2	allocation error	Any partial allocation is released.
3	read error	Handle close is still attempted exactly once.
4	configured read limit exceeded	No oversized result is published.
5	embedded NUL in read content	Read allocation is released.
6	invalid UTF-8 in read content	Read allocation is released.
7	close error	No second close attempt is emitted.
8	exact write error	Borrowed inputs remain caller-owned.

C caller contract

A path argument must be a non-null, NUL-terminated valid UTF-8 C string. Write text has the same preconditions and must contain no embedded NUL. A successful read returns a unique char pointer that the caller must free exactly once.

Path and write-text validity are external caller preconditions; helpers cannot safely inspect an invalid pointer or bytes beyond its first terminator. Runtime statuses 5 and 6 describe embedded-NUL and UTF-8 validation of bytes read from a file, not path validation.

Resource ceilings

A request may choose lower limits, but it cannot raise a limit above the trusted hard ceiling. Coherent tampering with both a request and its proof is rejected by independent validation.

RESOURCE	DEFAULT	TRUSTED HARD CEILING
Maximum file read bytes	1,048,576	16,777,216
Maximum live owned bytes	2,097,152	16,777,217
Maximum simultaneously open files	8	8
Maximum file operations	1024	1024
Scope proof nodes	request bounded	100,000
Scope proof edges	request bounded	400,000
Scope bindings	request bounded	50,000
Scope hoisted locals	request bounded	10,000
Loop/scope traversal depth	request bounded	128

Read allocation invariant

`max_live_owned_bytes` must cover `max_file_read_bytes` plus its terminating NUL byte. Invalid configurations reject with PYC1005 before conversion begins.

Part V - Diagnostics quick reference

CODE	AREA	PRIMARY MEANING
PYC2940	loop else	Unsafe while-body local read from the else suite.
PYC2941	loop else	Loop target may be unbound after zero iterations.
PYC3901	file shape	Unsupported with statement or session structure.
PYC3902	open binding	open is shadowed or not context-owned.
PYC3903	open arguments	Mode, encoding, newline, path, or literal path is outside profile.
PYC3904	handle binding	Handle target or rebinding violates fresh ownership.
PYC3905	operation	read/write method, arguments, mode, or result use is unsupported.
PYC3906	lifetime	Handle or owned read value escapes its proved lifecycle.
PYC3907	resource	Requested file-operation count exceeds its configured limit.
PYC3908	effect ABI	Source-defined call to a file-effect function is unsupported.
PYC2940	scalar scope	A reachable read occurs before a proved local store.
PYC2943	scalar scope	Publishing branches use conflicting C representations.
PYC3403	container scope	A fixed container binding is outside its proved lifetime.

Diagnostics carry a source span and remediation. Internal proof validation errors fail closed and do not publish generated C. Cancellation remains a distinct propagated control path.

Part VI - Stable conversion envelope

SUPPORTED CORE	SELECTED BOUNDARIES
Typed functions and explicit returns	No dynamic typing, decorators, generators, async execution, or exceptions.
bool, signed 64-bit int, double, borrowed strings	No implicit int/float coercion or general dynamic string operations.
if / elif / else and short-circuit Boolean expressions	Condition representations must be proved and comparison categories compatible.
range and fixed list / tuple / dictionary loops	Containers are immutable, statically bounded, and not generally comparable.
Immutable automatic static records	No general methods, mutation, dynamic fields, or record returns.
Positional, keyword, positional-only, required keyword-only calls	No defaults, variadics, argument unpacking, recursion, or indirect calls.
Bounded floor division and modulo	Numeric facts must prove divisor and overflow safety.
Phase 16 file entry points	No arbitrary Python file API; use the exact profile in Part III.

Whole-unit publication

PyCForge publishes generated C only after parsing, normalization, semantic analysis, independent proof validation, C-IR validation, rendering, and conformance checks all pass.

Part VII - PySide6 workspace improvements

The desktop workspace is now a required PySide6 application. The editor changes address the most immediate usability obstacles while keeping conversion evidence visible.

- Editor font size is configurable from 8 through 48 points and rebinds existing buffers.
- Python source auto-indent after block headers and preserves indentation on new lines.
- Generated C remains read-only and no longer receives passive full-line mapping overlays.
- Explicit source-to-C mapping navigation remains available when requested.
- The release smoke test constructs and closes the real MainWindow offscreen.

Reading generated C

The right editor shows syntax highlighting without painting every mapped line. Use explicit mapping navigation to identify correspondence instead of treating the entire output as selected.

Release validation

The Phase 16 release gate checks active identities, decision-trace schema, independent validator architecture, fact-table tampering, file ABI and resource ceilings, historical custody, real PySide6 behavior, and exact deterministic guide reconstruction.

Final checklist

- Use Python 3.11 syntax and explicit annotations for every supported function surface.
- Keep branch-published locals scalar, exhaustive, representation-compatible, and loop-independent.
- Use loop else only when natural completion has a clear meaning; initialize separate results for empty loops.
- Use the exact UTF-8 file context and one direct operation per with statement.
- Honor generated C status, ownership, NUL, UTF-8, and free responsibilities.
- Treat every rejection as a whole-unit outcome; no partial C is safe to consume.

The governing rule

If PyCForge cannot prove the source, its resource use, its ownership transfers, and its generated representation inside the declared contract, it rejects the unit instead of guessing.

PyCForge 0.16.0

Python 3.11 to deterministic portable ISO C11 source