# TEST SUITE AND CODE COVERAGE

Coenraad Marinissen

*c.marinissen@nikhef.nl*


June 24, 2026

Amsterdam

# TESTING

- Form has a test suite in the `check/` directory
  - Includes examples from the manual, new features, scripts reproducing (fixed) bugs.
  - You can run the default tests locally with `make check`
  - Runs on GitHub's CI runners on commit: `Ubuntu`, `macOs` and `Windows`
    - ➡ Also `form` and `tform` under valgrind

# TESTING

- Form has a test suite in the `check/` directory
  - Includes examples from the manual, new features, scripts reproducing (fixed) bugs.
  - You can run the default tests locally with `make check`
  - Runs on GitHub's CI runners on commit: `Ubuntu`, `macOs` and `Windows`
    - ➡ Also `form` and `tform` under valgrind

- Good for robustness
  - Validate new features without breaking existing ones
  - Confirms that Form behaves as expected under different conditions
    - ➡ Different builds with different features enabled
    - ➡ Different operating systems: `Ubuntu`, `macOs` and `Windows`

# ADD OWN TESTS

Add your own tests!

- coverage.frm
- diagrams.frm
- examples.frm
- features.frm
- fixes.frm
- polynomial.frm
- user.frm

Package authors should add tests!

See `check/extra` directory

- color.h
- forcer.h
- series.h
- fmft.h

# ADD OWN TESTS

Add your own tests!

- coverage.frm
- diagrams.frm
- examples.frm
- features.frm
- fixes.frm
- polynomial.frm
- user.frm

Package authors should add tests!

See `check/extra` directory

- color.h
- forcer.h
- series.h
- fmft.h

- Add fold containing the code
- In particular tests with tricky performance optimisations, rarely used features etc.
- Should be fast running, a few seconds at most, 30s under valgrind
- See also `check/README.md` to see what assertions you can use

```
*--#[ Square :
S x;
L F = (1+x)^2;
P;
.end
assert succeeded?
assert result("F") =~ expr("1+2*x+x^2")
*--#] Square :
```

# RUN TESTS LOCALLY

- Run default tests

```
Make check
```

- Run specific test(s) and binaries (saves time)

```
./check.rb [options] [--] [binname] [files|tests]
```

Run `./check.rb --help` to see the options

```
./check.rb /path/to/form
```

```
./check.rb Issue8
./check.rb 'divmod_*'
./check.rb examples.frm
```

- Run tests in `check/extra`

```
./check.rb -C extra
```

# COVERAGE

- How well does the test suite do?

  Click on this icon in the **`README.md`** to see the full coverage report

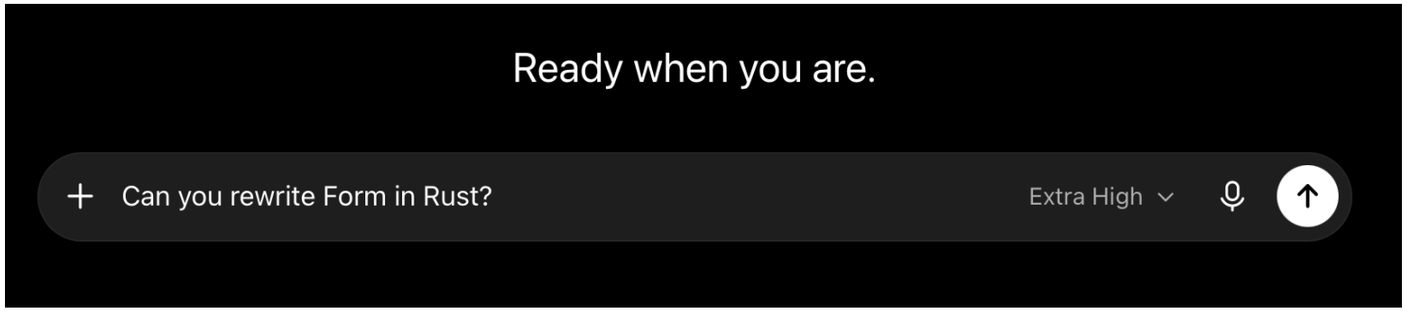
  → Code coverage


- We use **`gcov`** to collect the coverage statistics and **`Coveralls`** on Github for the coverage report


- Coverage increased in the last year from **49.26%** (June 9) to **63.16%** (June 22)
- Can still use many more tests! → Good coverage is >90%

- Not only for robustness,

  → also to make it feature proof



# COVERAGE

- We would appreciate if the result of this workshop is that many new tests are added and coverage is increased.
  - ➡ See the **exercises** for the hands-on session
  - ➡ For example: are all statements from the manual covered?
  - ➡ Also a good way to learn the source code
  - ➡ Can AI write good tests? Of course, this still need proper human review...!
  - ➡ The coverage report can also be created locally:

Script to fetch the libraries in `check/extra`:
https://gist.github.com/jodavies/49da8f04e7fe488fdc86df35a5714bd1

Script that creates the coverage report:
https://gist.github.com/jodavies/a99272b9dbadb6b74c06465a3edd4e14

Not macOS friendly (yet),

let me know if you want to run it on macOS

# UNREACHABLE CODE

- What about parts of the code that cannot be reached from a regular Form script? Such as unfinished features (`#setflag, dimension_, #namespace, Multibracket,...`), and obsolete statements.
  - ➡ Comment these out?
  - ➡ Hide them from the coverage statistic with a preprocessor variable?
    E.g. make it a build option.
  - ➡ Found out this morning:

```
/* LCOV_EXCL_START */
int unfinished_feature(WORD *term) {
    /* implementation */
    return 0;
}
/* LCOV_EXCL_STOP */
```