BLUE - A GENERIC BUILD-SYSTEM CRAFTED ENTIRELY IN GUILE

Sergio Pastor Pérez, Olivier Dion

TABLE OF CONTENTS

WHAT IS BLUE?

A BUILD-SYSTEM

```
(blueprint
 (configuration blue-configuration)
 (buildables
  (append
   blue-docs
   blue-modules
   blue-templates))
 (testables
  blue-tests)
 (commands
  (list
   check-install-command
   coverage-command
   install-command
   guix-build-command)))
```

SUPPORTING PROJECT CONFIGURATION

```
(blueprint
 (configuration blue-configuration)
 (buildables
  (append
   blue-docs
   blue-modules
   blue-templates))
 (testables
  blue-tests)
 (commands
  (list
   check-install-command
   coverage-command
   install-command
   guix-build-command)))
```

GOAL-DRIVEN DECLARATIONS

```
(blueprint
 (configuration blue-configuration)
 (buildables
  (append
   blue-docs
   blue-modules
   blue-templates))
 (testables
  blue-tests)
 (commands
  (list
   check-install-command
   coverage-command
   install-command
   guix-build-command)))
```

AND COMMAND BASED ACTIONS

```
(blueprint
 (configuration blue-configuration)
 (buildables
  (append
   blue-docs
   blue-modules
   blue-templates))
 (testables
  blue-tests)
 (commands
  (list
   check-install-command
   coverage-command
   install-command
   guix-build-command)))
```

IT'S ALSO A COMMAND DISPATCHER

```
 1 (define-command (hello-command files)
 2   ((invoke "hello")
 3    (category 'hello)
 4    (synopsis "Say hello")
 5    (help "FILE ...\nPrint hello to FILE."))
 6   (for-each
 7    (lambda (file)
 8      (with-output-to-file file
 9        (lambda () (display "Hello\n"))))
10    files))
11
12 (blueprint
13  (commands
14   (list hello-command)))
```

WITHOUT BOILER PLATE

```
(define-command (hello-command files)
  ((invoke "hello")
   (category 'hello)
   (synopsis "Say hello")
   (help "FILE ...\nPrint hello to FILE."))
  (for-each
   (lambda (file)
     (with-output-to-file file
       (lambda () (display "Hello\n"))))
   files))

(blueprint
 (commands
  (list hello-command)))
```

BUT MORE IMPORTANTLY...

A FRAMEWORK
TO MAKE YOUR OWN BUILD SYSTEM

```
(define-blue-class <tarball> (<buildable>)
  (executable
   #:getter tarball-executable
   #:init-value #%~#%?TAR
   #:init-keyword #:executable))

(define-method (ask-build-manifest
                (this <tarball>) (inputs <list>) (output <string>))
  (let ((relative-path (buildable-inputs-relative-to this)))
    (make-build-manifest
     (string-append "TAR\t" output)
     (cons*
      (tarball-executable this) "czf" output
      "-C" relative-path
      (map
       (lambda (path)
         (string-replace-substring path (string-append relative-path "/") ""))
       inputs)))))
```

DEFINE YOUR OWN BUILDABLE OBJECTS

```
1  (define-blue-class <tarball> (<buildable>)
2    (executable
3     #:getter tarball-executable
4     #:init-value #%~#%?TAR
5     #:init-keyword #:executable))
6
7  (define-method (ask-build-manifest
8                  (this <tarball>) (inputs <list>) (output <string>))
9    (let ((relative-path (buildable-inputs-relative-to this)))
10     (make-build-manifest
11      (string-append "TAR\t" output)
12      (cons*
13       (tarball-executable this) "czf" output
14       "-C" relative-path
15       (map
16        (lambda (path)
17          (string-replace-substring path (string-append relative-path "/") ""))
18        inputs)))))
```

AND LET BLUE KNOW WHAT TO DO WITH THEM

```
1  (define-blue-class <tarball> (<buildable>)
2    (executable
3     #:getter tarball-executable
4     #:init-value #%~#%?TAR
5     #:init-keyword #:executable))
6
7  (define-method (ask-build-manifest
8                  (this <tarball>) (inputs <list>) (output <string>))
9    (let ((relative-path (buildable-inputs-relative-to this)))
10     (make-build-manifest
11      (string-append "TAR\t" output)
12      (cons*
13       (tarball-executable this) "czf" output
14       "-C" relative-path
15       (map
16        (lambda (path)
17          (string-replace-substring path (string-append relative-path "/") ""))
18        inputs)))))
```

WHY ANOTHER BUILD-SYSTEM?

Because BLUE is

MINIMALIST AND SELF-CONTAINED

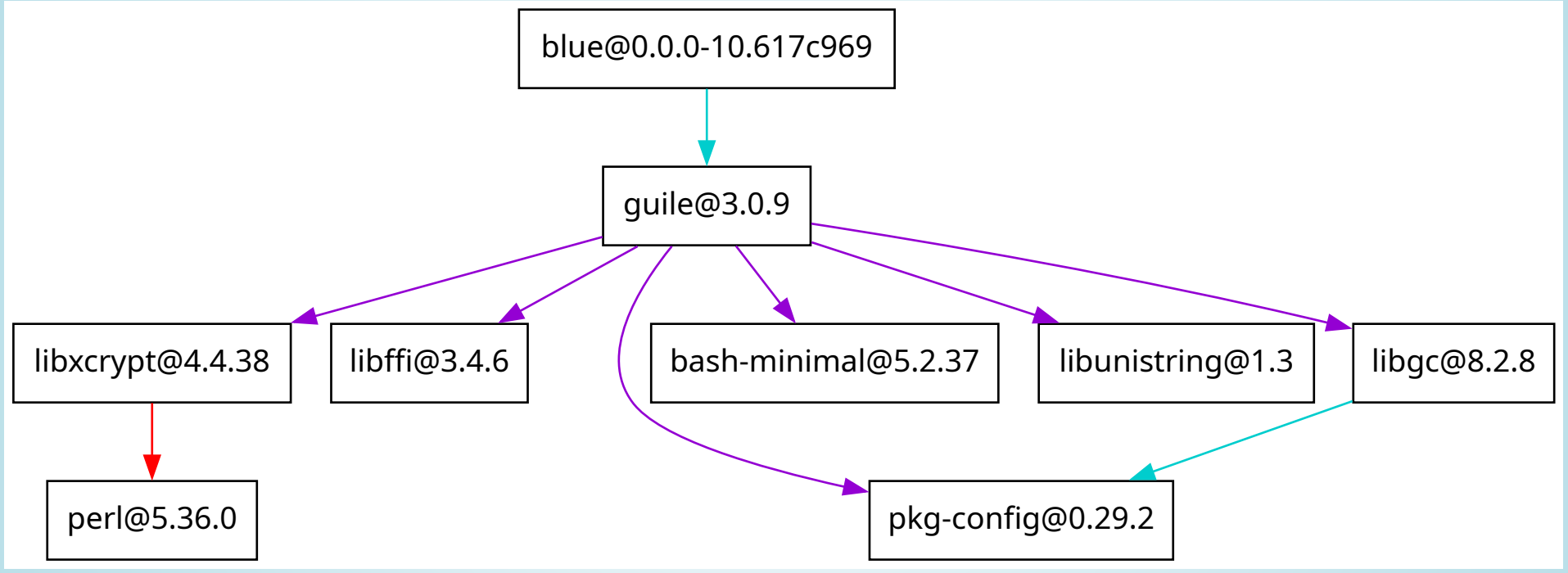


Figure 1: Dependencies of the BLUE package

GOAL-BASED

Ask what you want.

```
(c-binary
 (inputs '("prog.c"))
 (external-dependencies #%~(list #%?dep:sdl2))
 (outputs "prog"))
```

GOAL-BASED

Ask what you want.

```
(c-binary
 (inputs '("prog.c"))
 (external-dependencies #%~(list #%?dep:sdl2))
 (outputs "prog"))
```

BLUE will figure it out ⬇️

```
$ gcc -MMD -I<prefix>/include -I<prefix>/include/SDL2 -fPIE \
  -g -O2 -Wall -Wextra -c -o prog.o prog.c
$ gcc -g -O2 -Wall -Wextra -L<prefix>/lib -o prog prog.o -lSDL2
```

EXTENSIBLE

EXTENSIBLE

GOOPS

EXTENSIBLE

GOOPS

- Method overriding / redefining
- Inheritance

EXTENSIBLE

GOOPS

- Method overriding / redefining
- Inheritance

User defined commands

EXTENSIBLE

GOOPS

- Method overriding / redefining
- Inheritance

User defined commands

- Override built-in commands

EXTENSIBLE

GOOPS

- Method overriding / redefining
- Inheritance

User defined commands

- Override built-in commands

Plugin system (TODO)

EXTENSIBLE

GOOPS

- Method overriding / redefining
- Inheritance

User defined commands

- Override built-in commands

Plugin system (TODO)

- Custom build scheduler

EXTENSIBLE

GOOPS

- Method overriding / redefining
- Inheritance

User defined commands

- Override built-in commands

Plugin system (TODO)

- Custom build scheduler

Stencil system

EXTENSIBLE

GOOPS

- Method overriding / redefining
- Inheritance

User defined commands

- Override built-in commands

Plugin system (TODO)

- Custom build scheduler

Stencil system

EXTENSIBLE

GOOPS

- Method overriding / redefining
- Inheritance

User defined commands

- Override built-in commands

Plugin system (TODO)

- Custom build scheduler

Stencil system

Project serialization

EXTENSIBLE

GOOPS

- Method overriding / redefining
- Inheritance

User defined commands

- Override built-in commands

Plugin system (TODO)

- Custom build scheduler

Stencil system

Project serialization

EXTENSIBLE

GOOPS

- Method overriding / redefining
- Inheritance

User defined commands

- Override built-in commands

Plugin system (TODO)

- Custom build scheduler

Stencil system

Project serialization

Autocomplete engine

EXTENSIBLE

GOOPS

- Method overriding / redefining
- Inheritance

User defined commands

- Override built-in commands

Plugin system (TODO)

- Custom build scheduler

Stencil system

Project serialization

Autocomplete engine

- Shell integration

HELLO BLUE

LAYOUT

blueprint.scm

A container for gathering a BLUE project description.

```
1 ;;; Modules
2
3 ;;; Buildables
4
5 ;;; Commands
6
7 ;;; Blueprint
8
```

IMPORT STENCILS

Stencil

A set of buildable types and utilities to simplify the description of a common pattern.

```
1 ;;; Modules
2 (use-modules (blue stencils c))
3
4 ;;; Buildables
5
6 ;;; Commands
7
8 ;;; Blueprint
9
```

DECLARE BUILDABLES

Buildable

An object that BLUE knows how to build.

```
;;; Modules
(use-modules (blue stencils c))

;;; Buildables
(define libhello
  (c-binary
   (inputs "hello.c")
   (outputs "libhello.a")
   (library? #t)
   (shared? #f)))

(define hello
  (c-binary
   (inputs (list "main.c" libhello))
   (outputs "hello")))
```

DEFINE COMMANDS

Command

An action that BLUE can execute.

```
1  ;;; Modules
2  (use-modules (blue stencils c)
3               (blue subprocess)
4               (blue types command))
5
6  ;;; Buildables
7  (define libhello
8    (c-binary
9     (inputs "hello.c")
10    (outputs "libhello.a")
11    (library? #t)
12    (shared? #f)))
13
14 (define hello
15   (c-binary
```

DEFINE BLUEPRINT

Blueprint

An object that describes a BLUE project.

```
;;; Modules
(use-modules (blue stencils c)
             (blue subprocess)
             (blue types blueprint)
             (blue types command))

;;; Buildables
(define libhello
  (c-binary
   (inputs "hello.c")
   (outputs "libhello.a")
   (library? #t)
   (shared? #f)))

(define hello
```

LET'S BUILT IT

UPS... I FORGOT TO CONFIGURE

The C stencil requires a configuration step.

```
λ blue build
Traceback (most recent call last):
0: (_) at blue/command.scm:80:8

    77|      (cond
    78|       (result result)
    79|       (required?
>   80|        (raise-exception
              ^
    81|         (make-exception
    82|          (make-external-error)
    83|          (make-exception-with-origin

Error while running command `build':
 &external-error
 origin:
  blue/types/configuration.scm:543:2 → blue/types/configuration.scm:130:6
 irritants:
  (#:configuration-hash
   "57134f29eccd79ff5dfa0f71a493ce1d021c6b41f2ec4c91230638eceae1cc65f6b8e024b2cf00f39")
 message:
  Could not find the configuration in the store.
 hint:
  This is usually because the command invoked requires a configuration,
  but no configuration exists in the store matching the configuration
  manifest of the blueprint.

  This can happen if no configuration has been done or that the
  configuration manifest has changed.

  Try to configure the build directory with the built-in command:

    blue configure [ARGS]...
```

Figure 2: Missing BLUE configuration

Commands optionally can be chained with '- -'.

```
λ blue configure -- build
starting phase `parse-command-line-arguments'
    parsing 0 arguments:
starting phase `resolve-host'
    guessing host target
        OK x86_64-unknown-linux-gnu
starting phase `resolve-dependencies'
    resolving 0 dependencies
starting phase `expand-variables'
    guessing build target
        OK x86_64-unknown-linux-gnu
    guessing C toolchain
        looking for binary x86_64-unknown-linux-gnu-gcc
            OK x86_64-unknown-linux-gnu-gcc → /gnu/store/pdsp8rh…-gcc-15.2.0/bin/x86_64-unknown-linux-gnu-gcc
configuration summary
    0 warnings
    0 errors
    CC  /home/pastor/projects/lapislazuli/fosdem2026-presentation/hello-blue/build/hello.o
    CC  /home/pastor/projects/lapislazuli/fosdem2026-presentation/hello-blue/build/main.o
    LINK    /home/pastor/projects/lapislazuli/fosdem2026-presentation/hello-blue/build/libhello.a
    LINK    /home/pastor/projects/lapislazuli/fosdem2026-presentation/hello-blue/build/hello
```

Figure 3: Command chaining

COMMAND CAN TAKE ARGUMENTS

Here we pass the arguments directly to Guix.

```
(define-command (guix-build-command arguments)
  ((invoke "guix-build")
   (category 'guix)
   (load-configuration-policy 'no))
  (zero? (popen "guix"
                (append
                 `("time-machine"
                   "-C"
                   ,(string-append #%?srcdir "/channels.scm")
                   "--"
                   "build"
                   "-f"
                   ,(string-append #%?srcdir "/guix.scm"))
                 arguments)
                #:working-directory #%?srcdir)))
```

WHAT IF I MY PROJECT NEEDS STATEFUL DATA?

INTRODUCING THE CONFIGURATION

Configuration

An object that describes stateful data.

```
1 (define hello-configuration
2   (configuration))
```

HERE YOU CAN DEFINE VARIABLES FOR YOUR PROJECT

```
(define hello-configuration
  (configuration
   (variables
    (list
     (variable
      (name "VERSION")
      (value "0.0.1"))
     (variable
      (name "CFLAGS")
      (value
       (lambda* (#:key host #:allow-other-keys)
         (match-glob host
           (("aarch64*") "-moutline-atomics")
           (else ""))))))))))
```

AND ALSO YOUR PROJECT DEPENDENCIES

```
(define hello-configuration
  (configuration
   (variables
    (list
     (variable
      (name "VERSION")
      (value "0.0.1"))
     (variable
      (name "CFLAGS")
      (value
       (lambda* (#:key host #:allow-other-keys)
         (match-glob host
           (("aarch64*") "-moutline-atomics")
           (else ""))))))))
   (dependencies
```

BLUE WILL ENSURE THE DEPENDENCIES ARE MEET

```
λ blue configure
starting phase `parse-command-line-arguments'
    parsing 0 arguments:
starting phase `resolve-host'
    guessing host target
        OK x86_64-unknown-linux-gnu
starting phase `resolve-dependencies'
    resolving 1 dependency
        NO sdl2 (???)
starting phase `expand-variables'
    guessing C toolchain
        looking for binary x86_64-unknown-linux-gnu-gcc
            OK x86_64-unknown-linux-gnu-gcc → /gnu/store/pdsp8rh…-gcc-15.2.0/bin/x86_64-unknown-linux-gnu-gcc
    guessing build target
        OK x86_64-unknown-linux-gnu
configuration summary
    0 warnings
    1 error
        &error
        origin:
         #{<c-dependency>: blueprint.scm:33:5 → blue/dependency/c.scm:63:8}
        caused-by:
         &external-error
         origin:
          $ pkg-config —exists sdl2
         message:
          Package sdl2 was not found in the pkg-config search path.
          Perhaps you should add the directory containing `sdl2.pc'
          to the PKG_CONFIG_PATH environment variable
          No package 'sdl2' found
```

Figure 4: Unmet dependency exception

LET'S MAKE A CONFIGURATION EXAMPLE

A WHITE CANVAS...

```
1 (use-modules (blue types blueprint))
2
3 (blueprint)
```

LET'S ADD A BUILDABLE

```
1 (use-modules (blue stencils c)
2              (blue types blueprint))
3
4 (blueprint
5  (buildables
6   (list
7    (c-binary
8     (inputs '("prog.c"))
9     (outputs "prog")))))
```

LET'S CONFIGURE A VARIABLE

CFLAGS

default to "-g -O2 -Wall -Wextra" (C stencil)

```
(use-modules (blue stencils c)
             (blue types blueprint)
             (blue types configuration)
             (blue types variable))

(blueprint
 (configuration
  (configuration
   (variables
    (list
     (variable
      (name "CFLAGS")
      (value "-O3"))))))
 (buildables
  (list
```

AND A DEPENDENCY

```
(use-modules (blue dependency c)
             (blue stencils c)
             (blue types blueprint)
             (blue types configuration)
             (blue types variable))

(blueprint
 (configuration
  (configuration
   (variables
    (list
     (variable
      (name "CFLAGS")
      (value "-O3"))))
   (dependencies
```

AND A DEPENDENCY

```
11      (list
12       (variable
13        (name "CFLAGS")
14        (value "-O3"))))
15     (dependencies
16      (list
17       (c-dependency
18        (name "sdl2")
19        (required? #t))))))
20   (buildables
21    (list
22     (c-binary
23      (inputs '("prog.c"))
24      (external-dependencies #%~(list #%?dep:sdl2))
25      (outputs "prog")))))
```

THAT WAS EASY... BUT HOW DO WE USE THE CONFIGURATION?

LET'S DEFINE A NEW COMMAND TO THAT USES THE VARIABLES

```
 1 (use-modules (blue dependency c)
 2              (blue stencils c)
 3              (blue types blueprint)
 4              (blue types configuration)
 5              (blue types variable)
 6              (blue types command))
 7
 8 (define-command (print-info-command arguments)
 9   ((invoke "info"))
10   (format #t "-----
11 Source directory: ~a
12 Build directory: ~a
13 Prefix: ~a
14 CFLAGS: ~a
15 sdl2: ~a
```

LET'S DEFINE A NEW COMMAND TO THAT USES THE VARIABLES

```
30        (name "CFLAGS")
31        (value "-O3"))))
32     (dependencies
33      (list
34       (c-dependency
35        (name "sdl2")
36        (required? #t))))))
37   (buildables
38    (list
39     (c-binary
40      (inputs '("prog.c"))
41      (external-dependencies #%~(list #%?dep:sdl2))
42      (outputs "prog"))))
43   (commands
44    (list print-info-command)))
```

LET'S BUILT IT

```
λ blue configure --prefix=/tmp/hello-install -- build -- info
starting phase `parse-command-line-arguments'
    parsing 1 argument: --prefix=/tmp/hello-install
starting phase `resolve-host'
    guessing host target
        OK x86_64-unknown-linux-gnu
starting phase `resolve-dependencies'
    resolving 1 dependency
        OK sdl2 (2.30.8)
starting phase `expand-variables'
    guessing C toolchain
        looking for binary x86_64-unknown-linux-gnu-gcc
            OK x86_64-unknown-linux-gnu-gcc → /gnu/store/pdsp8rh…-gcc-15.2.0/bin/x86_64-unknown-linux-gnu-gcc
    guessing build target
        OK x86_64-unknown-linux-gnu
configuration summary
    0 warnings
    0 errors
    CC  /home/pastor/projects/lapislazuli/fosdem2026-presentation/hello-configuration/prog.o
    LINK    /home/pastor/projects/lapislazuli/fosdem2026-presentation/hello-configuration/prog
─────
Source directory: /home/pastor/projects/lapislazuli/fosdem2026-presentation/hello-configuration
Build directory: /home/pastor/projects/lapislazuli/fosdem2026-presentation/hello-configuration
Prefix: /tmp/hello-install
CFLAGS: -O3
sdl2: -lSDL2
─────
```

Figure 5: Configure with command arguments

DELAYED COMPUTATIONS

#%? ... WHAT IS THAT?

```
 1 (define-command (print-info-command arguments)
 2   ((invoke "info"))
 3   (format #t "-----
 4 Source directory: ~a
 5 Build directory: ~a
 6 Prefix: ~a
 7 CFLAGS: ~a
 8 sdl2: ~a
 9 -----
10 "
11           #%?srcdir
12           #%?builddir
13           #%?prefix
14           #%?CFLAGS
15           (c-dependency-libs-only-l #%?dep:sdl2)))
```

BLUE DELAYED COMPUTATIONS

Delayed computation

An expressions evaluated at run-time within a dynamic computation environment. Its application is memoized.

Expression	Meaning
#%~FORM	Create a delayed expression from FORM.
#%?SYMBOL	Ask for the value of SYMBOL within the current computation.
#%,FORM	Undelay the expression in FORM within a delayed expression.

LET'S DELAY AN EXPRESSION

```
1 #%~"guile"
```

LET'S DELAY AN EXPRESSION

```
1 #%~"guile"
```

Expands to ⬇️

```
1 (make-computation (lambda () "guile"))
```

LET'S RETRIEVE A CONFIGURATION VALUE

```
1 #%~(if (string=? #%?libdir "/usr/local/lib")
2        #%,special-location
3        (path->string
4         (path-join #%?libdir "guile")))
```

LET'S RETRIEVE A CONFIGURATION VALUE

```
#%~(if (string=? #%?libdir "/usr/local/lib")
       #%,special-location
       (path->string
        (path-join #%?libdir "guile")))
```

Expands to ⬇️

```
(make-computation
 (let* ((t-1dff1b83541ce327-295 special-location))
   (lambda ()
     (let ((t-1dff1b83541ce327-294
            (delay (computation-ask 'libdir))))
       (if (string=? (force t-1dff1b83541ce327-294)
                     "/usr/local/lib")
           t-1dff1b83541ce327-295
           (path->string
            (path-join
             (force t-1dff1b83541ce327-294) "guile")))))))
```

LET'S DRAW SOME SIMILARITIES

Makefile (lazy evaluation)

```
PREFIX = /usr/local
BINDIR = ${PREFIX}/bin
```

G-Expressions

```
#~(string-append #$output "/bin")
```

BLUE

```
#%~(string-append #%?prefix "/bin")
```

ONE LAST REMARK

Delayed computations bring

ONE LAST REMARK

Delayed computations bring

- Much more than variable expansion.

ONE LAST REMARK

Delayed computations bring

- Much more than variable expansion.
- The power and expressivity of Guile.

CAN BE SERIALIZED TO DISK!

```
1  #%~(string-append #%?prefix "/bin")
```

Expands to ⬇️

CAN BE SERIALIZED TO DISK!

```
#%~(string-append #%?prefix "/bin")
```

Expands to ⬇️

```
(let
    ((the-computation
      ((@ (blue computation) make-computation)
       (let ()
         (let ()
           (define proc
             (lambda ()
               (let
                   ((t-5c05c27ebb17590-ab
                     ((@@ (guile) make-promise)
                      (lambda ()
                        ((@@ (blue computation) computation-ask) (quote prefix))))))
                 ((@ (guile) string-append)
                  ((@@ (blue computation) force) t-5c05c27ebb17590-ab) "/bin"))))
           ((@@ (blue utils hash) set-object-property!) proc (quote blue-hash)
            754606235298532093936125085570906851521)
           proc))
       (quote ((line . 91) (column . 20)))
       (quote (string-append (~computation-ask (quote prefix)) "/bin"))
       (delay
         (quote
          (let ()
```

BACKTRACES

MANY OF US MAKE MISTAKES...

WORRY NOT...
WE'VE GOT PRETTY BACKTRACES!

LET'S SAY YOU MAKE A TYPO

```
 1 (use-modules (blue dependency c)
 2              (blue stencils c)
 3              (blue types blueprint)
 4              (blue types configuration)
 5              (blue types variable)
 6              (blue types command))
 7
 8 (define-command (print-info-command arguments)
 9   ((invoke "info"))
10   (format #t "-----
11 Source directory: ~a
12 Build directory: ~a
13 Prefix: ~a
14 CFLAGS: ~a
15 sdl2: ~a
```

THIS IS THE BLUE BACKTRACE YOU WILL SEE

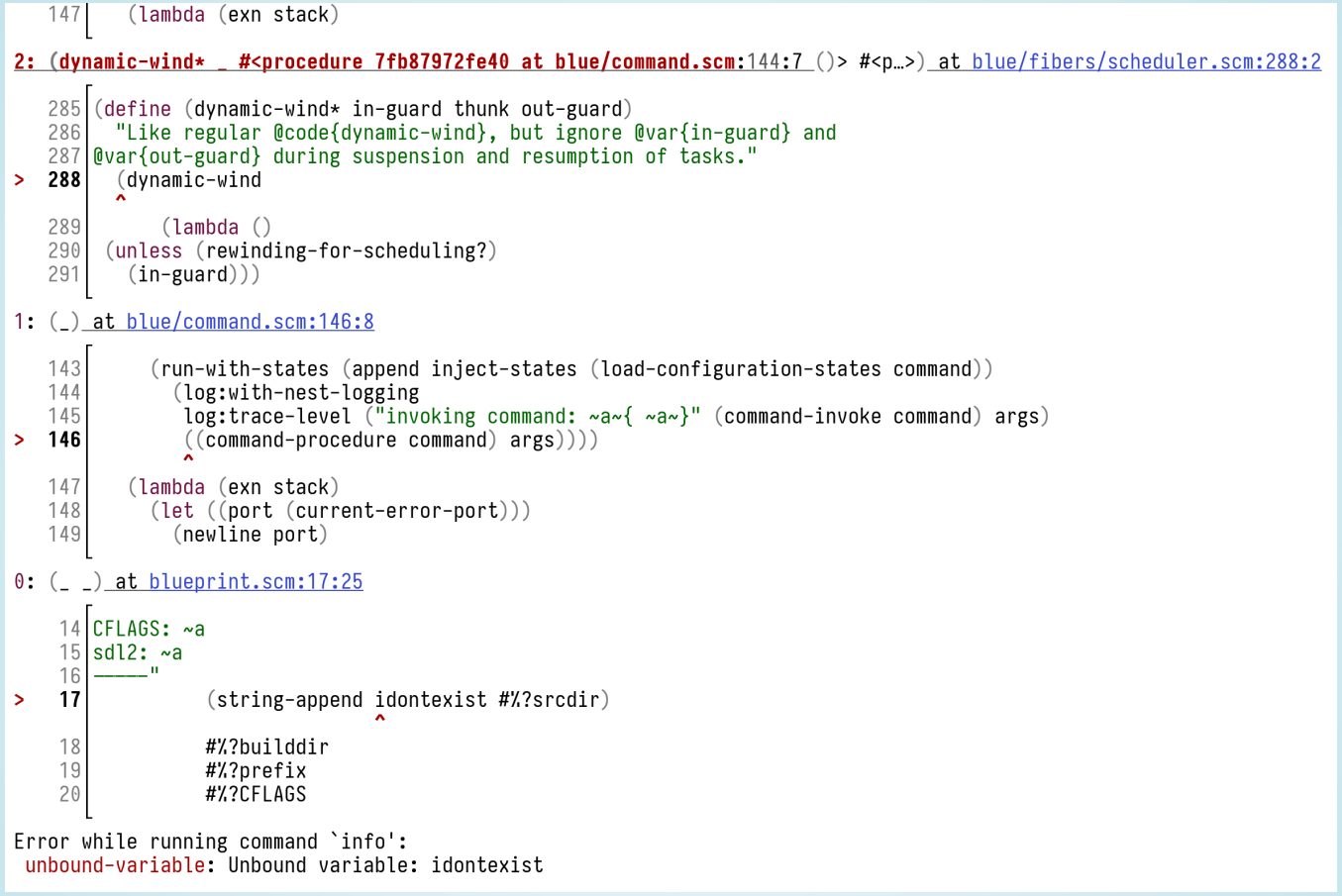
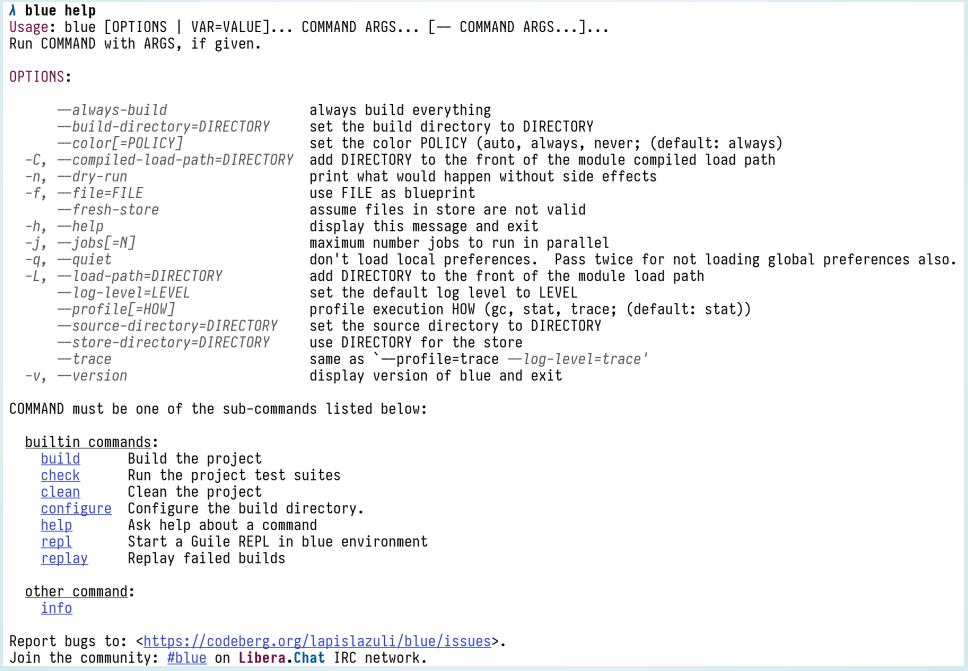


Figure 6: Unbound variable BLUE backtrace

HELP MENUS

BLUE HAS DYNAMIC AND INTERACTIVE HELP MENUS



```
λ blue help
Usage: blue [OPTIONS | VAR=VALUE]... COMMAND ARGS... [— COMMAND ARGS...]...
Run COMMAND with ARGS, if given.

OPTIONS:

      —always-build                 always build everything
      —build-directory=DIRECTORY    set the build directory to DIRECTORY
      —color[=POLICY]               set the color POLICY (auto, always, never; (default: always)
  -C, —compiled-load-path=DIRECTORY add DIRECTORY to the front of the module compiled load path
  -n, —dry-run                      print what would happen without side effects
  -f, —file=FILE                    use FILE as blueprint
      —fresh-store                  assume files in store are not valid
  -h, —help                         display this message and exit
  -j, —jobs[=N]                     maximum number jobs to run in parallel
  -q, —quiet                        don't load local preferences.  Pass twice for not loading global preferences also.
  -L, —load-path=DIRECTORY          add DIRECTORY to the front of the module load path
      —log-level=LEVEL              set the default log level to LEVEL
      —profile[=HOW]                profile execution HOW (gc, stat, trace; (default: stat))
      —source-directory=DIRECTORY   set the source directory to DIRECTORY
      —store-directory=DIRECTORY    use DIRECTORY for the store
      —trace                        same as `—profile=trace —log-level=trace'
  -v, —version                      display version of blue and exit

COMMAND must be one of the sub-commands listed below:

  builtin commands:
    build      Build the project
    check      Run the project test suites
    clean      Clean the project
    configure  Configure the build directory.
    help       Ask help about a command
    repl       Start a Guile REPL in blue environment
    replay     Replay failed builds

  other command:
    info

Report bugs to: <https://codeberg.org/lapislazuli/blue/issues>.
Join the community: #blue on Libera.Chat IRC network.
```

Figure 7: BLUE help menus

REPLAY

Since mistakes are bound to happen…

```
λ blue replay
origin: #{<c-object>/home/pastor/projects/lapislazuli/fosdem2026-presentation/hello-configuration/prog.o: blue/stencils/c.scm:296:8}
replay: blue replay da0015b3ea84a59
log: blue replay --log da0015b3ea84a59
inputs:
+ /home/pastor/projects/lapislazuli/fosdem2026-presentation/hello-configuration/prog.c
outputs:
+ /home/pastor/projects/lapislazuli/fosdem2026-presentation/hello-configuration/prog.o
error:
 origin:
  #{<c-object>/home/pastor/projects/lapislazuli/fosdem2026-presentation/hello-configuration/prog.o: blue/stencils/c.scm:296:8}
 caused-by:
  irritants:
   (#:exit-value
    1
    #:subprocess
    ("gcc"
     "-MMD"
     "-I/gnu/store/74szk5l…-sdl2-2.30.8/include"
     "-I/gnu/store/74szk5l…-sdl2-2.30.8/include/SDL2"
     "-fdiagnostics-color=always"
     "-fPIE"
     "-O3"
     "-D_REENTRANT"
     "-c"
     "-o"
     "/home/pastor/projects/lapislazuli/fosdem2026-presentation/hello-configuration/prog.o"
     "/home/pastor/projects/lapislazuli/fosdem2026-presentation/hello-configuration/prog.c"))
  message:
   SUBPROCESS invocation failed.
```

Figure 8: BLUE replay command

PREFERENCE SYSTEM

Preference

A setting that configures some aspect of BLUE

Projects, and extensions can also define their own preferences that users can customize.

DEFINE THEM GLOBALLY

```
1 ;;; In $XDG_CONFIG_HOME/blue/config.scm
2 (set-preference! blue.ui:jobs 4)
3 (set-preference! blue.ui:color-policy 'always)
```

PROJECT LOCALLY

```
1 ;;; In $%?srcdir/.blue-config.scm
2 (set-preference!
3  blue.command.builtin.configure:flags
4  '("--fast-install"))
```

OR IN A ENVIRONMENT VARIABLE

```
1 export BLUE_PREFERENCES=blue.stencils.c.configuration:cc=clang
```

EXTENSIBILITY

INHERITANCE

```
1 (define-blue-class <texinfo> (<buildable>))
```

ADDING ADDITIONAL SLOTS

```
(define-blue-class <texinfo> (<buildable>)
  (executable
   #:getter info-executable
   #:init-value "makeinfo"
   #:init-keyword #:executable
   #:type string?)
  (flags
   #:getter info-flags
   #:init-value '()
   #:init-keyword #:flags
   #:type (list-of? string?)))
```

TYPED SPECIFICATION FOR SLOTS

```
 1 (define-blue-class <texinfo> (<buildable>)
 2   (executable
 3    #:getter info-executable
 4    #:init-value "makeinfo"
 5    #:init-keyword #:executable
 6    #:type string?)
 7   (flags
 8    #:getter info-flags
 9    #:init-value '()
10    #:init-keyword #:flags
11    #:type (list-of? string?)))
```

METHOD OVERLOADING

```
 1 (define-method (ask-build-manifest
 2                 (this <texinfo>)
 3                 (input <string>)
 4                 (output <string>))
 5   (make-build-manifest
 6    (string-append "MKINFO\t" output)
 7    (list (info-executable this)
 8          (info-flags this)
 9          (string-append "--output=" output)
10          input)))
```

BUILDABLE

```
1  (define-method (ask-build-manifest
2                  (this <texinfo>)
3                  (input <string>)
4                  (output <string>))
5    (make-build-manifest
6     (string-append "MKINFO\t" output)
7     (list (info-executable this)
8           (info-flags this)
9           (string-append "--output=" output)
10          input)))
```

INPUT

```
1  (define-method (ask-build-manifest
2                  (this <texinfo>)
3                  (input <string>)
4                  (output <string>))
5    (make-build-manifest
6     (string-append "MKINFO\t" output)
7     (list (info-executable this)
8           (info-flags this)
9           (string-append "--output=" output)
10          input)))
```

OUTPUT

```
1  (define-method (ask-build-manifest
2                  (this <texinfo>)
3                  (input <string>)
4                  (output <string>))
5    (make-build-manifest
6     (string-append "MKINFO\t" output)
7     (list (info-executable this)
8           (info-flags this)
9           (string-append "--output=" output)
10          input)))
```

MANIFEST

```
(define-method (ask-build-manifest
                (this <texinfo>)
                (input <string>)
                (output <string>))
  (make-build-manifest
   (string-append "MKINFO\t" output)
   (list (info-executable this)
         (info-flags this)
         (string-append "--output=" output)
         input)))
```

HEADER

Progress indicator that BLUE will print.

```
(define-method (ask-build-manifest
                (this <texinfo>)
                (input <string>)
                (output <string>))
  (make-build-manifest
   (string-append "MKINFO\t" output)
   (list (info-executable this)
         (info-flags this)
         (string-append "--output=" output)
         input)))
```

ACTION

The code that generates the artifact.

```
(define-method (ask-build-manifest
                (this <texinfo>)
                (input <string>)
                (output <string>))
  (make-build-manifest
   (string-append "MKINFO\t" output)
   (list (info-executable this)
         (info-flags this)
         (string-append "--output=" output)
         input)))
```

BLUE.EL

The Emacs interface for BLUE

blue-run-command

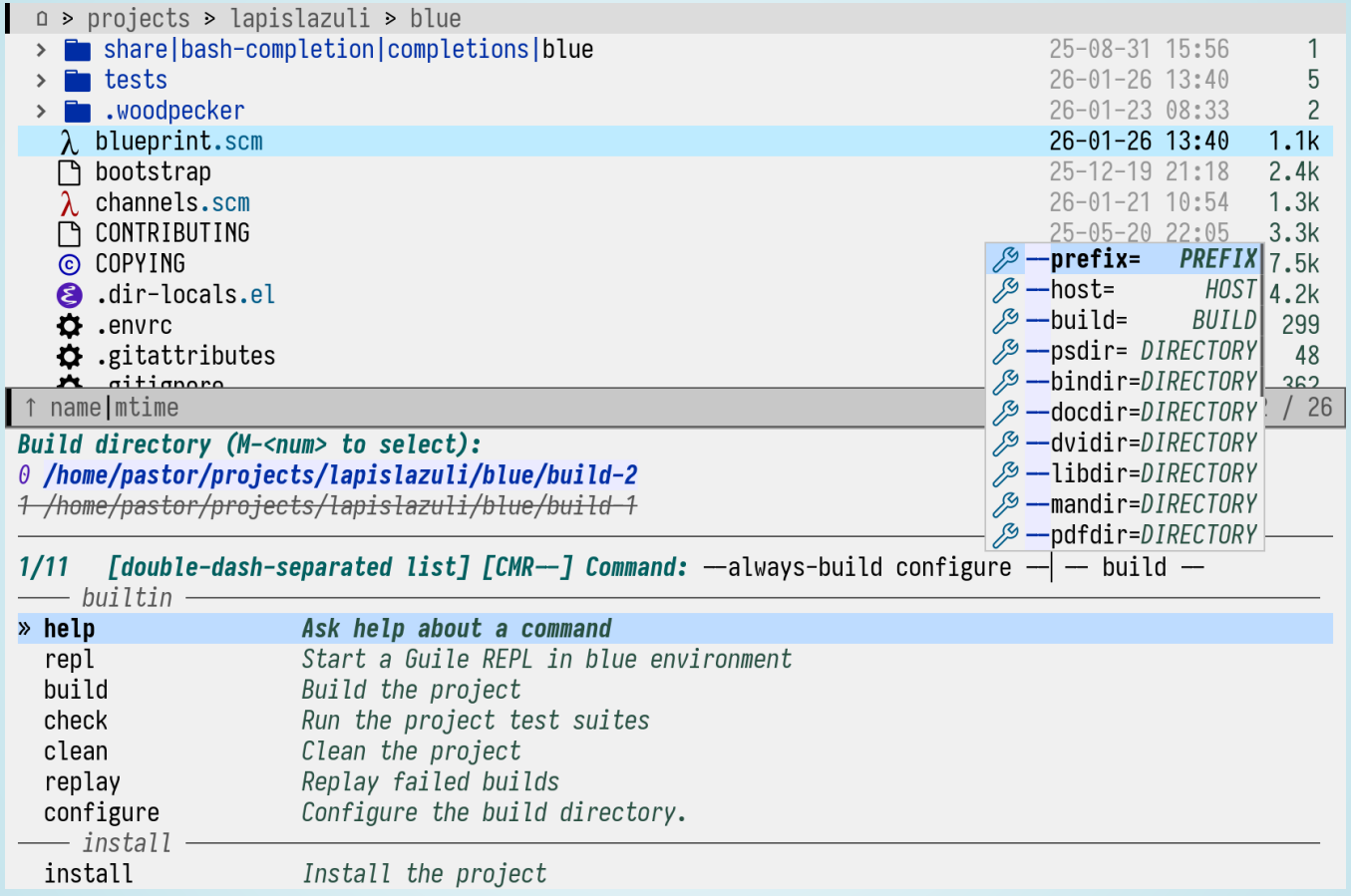


Figure 9: BLUE run command Emacs interface

blue-transient

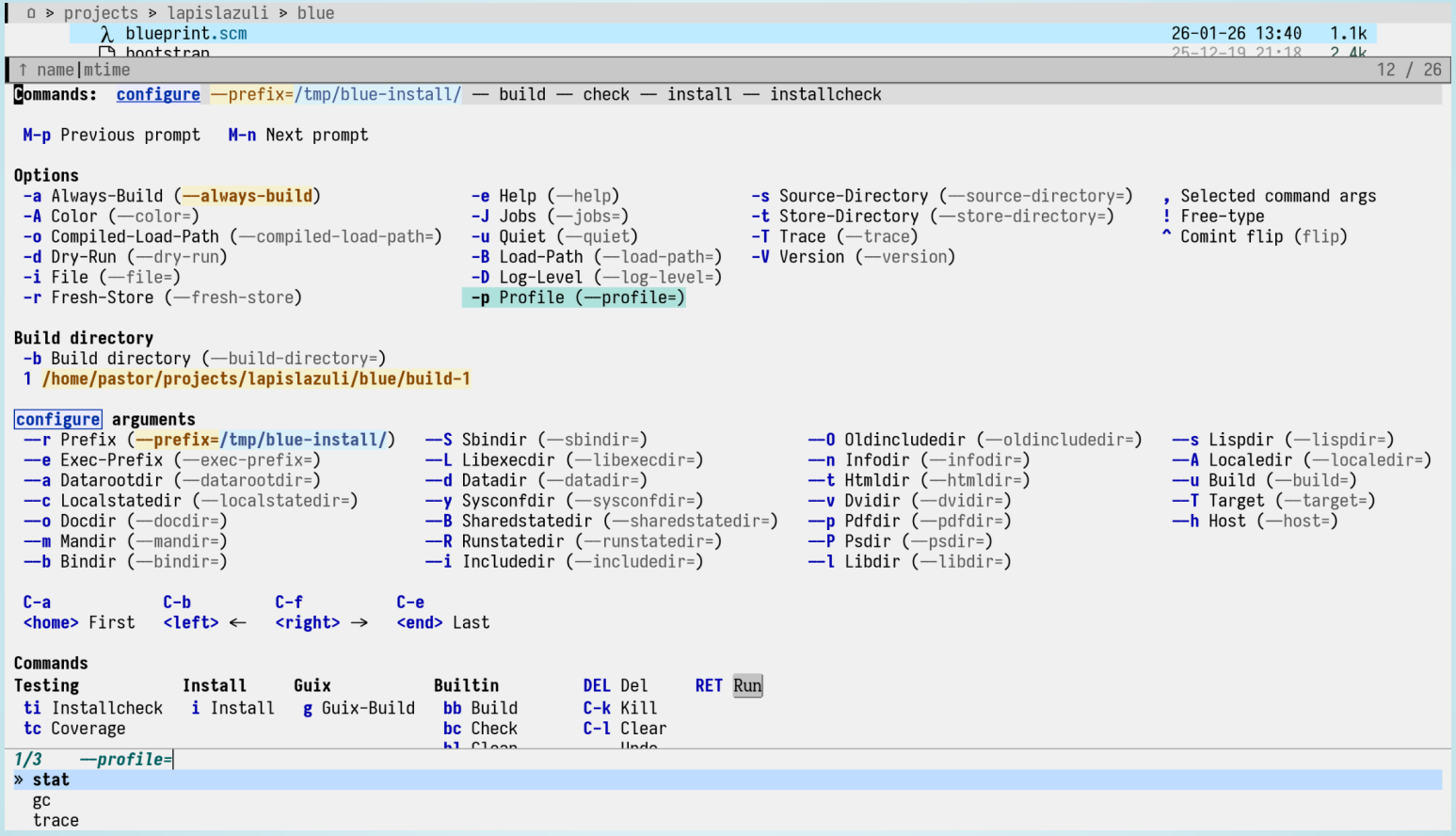


Figure 10: BLUE transient Emacs interface

blue-replay

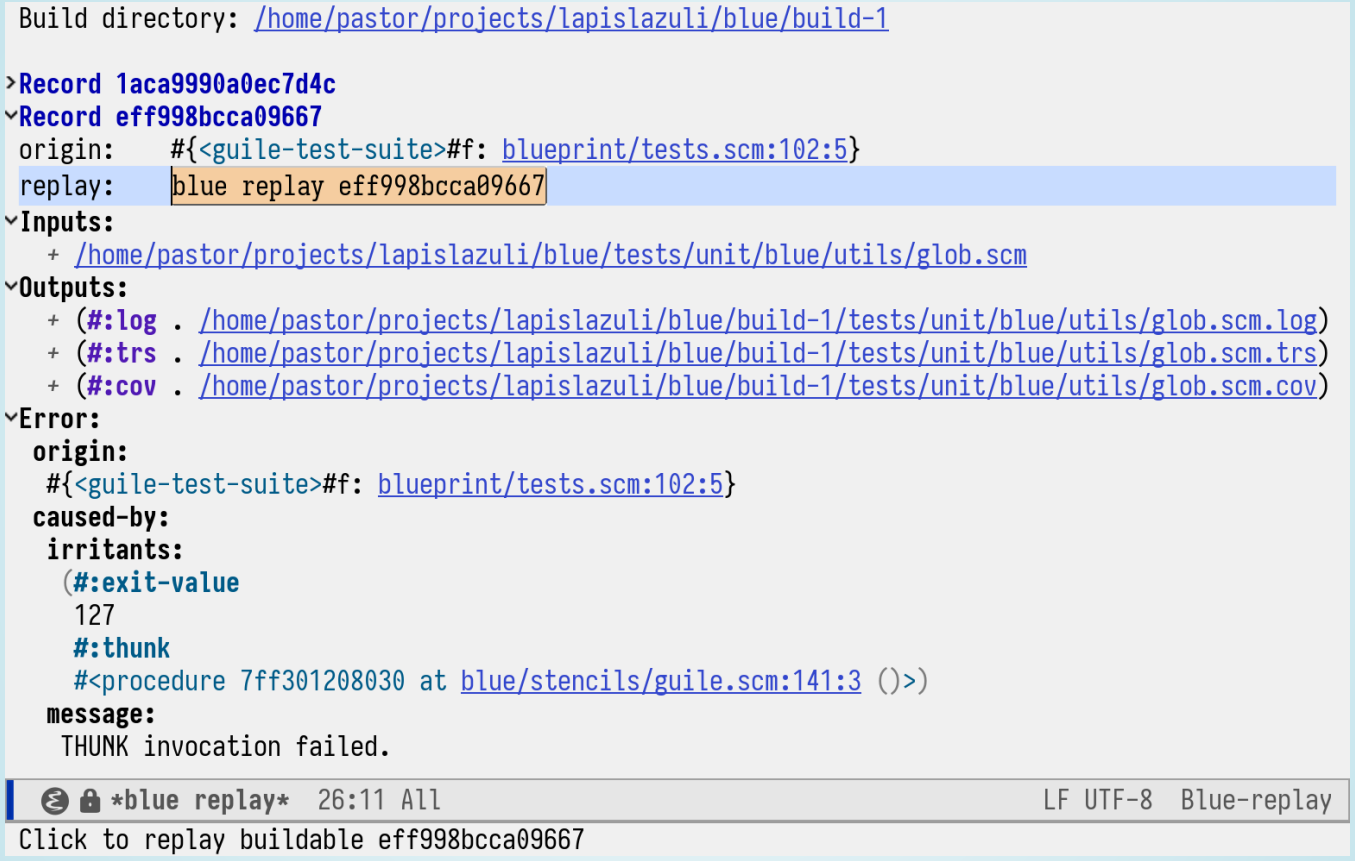


Figure 11: BLUE replay Emacs interface

GUIX SUPPORT

Bluebox

A Guix channel for BLUE.

blue-build-system

```
(define-public blue
  (let ((commit "4ef3c82de00da039ad255d458becc01d9cb65507")
        (revision "12"))
    (package
      (name "blue")
      (version (git-version "0.0.0" revision commit))
      (source (origin
                (method git-fetch)
                (uri (git-reference
                      (url "https://codeberg.org/lapislazuli/b
                      (commit commit)))
                (file-name (git-file-name name version))
                (sha256
                 (base32
                  "1hv9x7002nal967ywh73dpcakygpb1arxyzjmxd93jq
```

IT CAN BOOTSTRAP BLUE

```
(define-public blue
  (let ((commit "4ef3c82de00da039ad255d458becc01d9cb65507")
        (revision "12"))
    (package
      (name "blue")
      (version (git-version "0.0.0" revision commit))
      (source (origin
                (method git-fetch)
                (uri (git-reference
                      (url "https://codeberg.org/lapislazuli/b
                      (commit commit)))
                (file-name (git-file-name name version))
                (sha256
                 (base32
                  "1hv9x7002nal967ywh73dpcakygpb1arxyzjmxd93jq
```

SUBPHASES

Subphase

Auto-generated Guix phase for BLUE.

```
(define-public blue
  (let ((commit "4ef3c82de00da039ad255d458becc01d9cb65507")
        (revision "12"))
    (package
      (name "blue")
      (version (git-version "0.0.0" revision commit))
      (source (origin
                (method git-fetch)
                (uri (git-reference
                      (url "https://codeberg.org/lapislazuli/b
                      (commit commit)))
                (file-name (git-file-name name version))
                (sha256
                 (base32
                  "1hv9x7002nal967ywh73dpcakygpb1arxyzjmxd93jq
```

SUBPHASES LOOK LIKE THIS

```
guix build -L. blue --check --no-grafts
The following derivation will be built:
  /gnu/store/jqbskg7sx076vc66vbngd9xvscx6a1g4-blue-0.0.0-13.0e
building /gnu/store/jqbskg7sx076vc66vbngd9xvscx6a1g4-blue-0.0.
starting phase `separate-from-pid1'
starting phase `set-SOURCE-DATE-EPOCH'
starting phase `set-paths'
starting phase `install-locale'
starting phase `unpack'
starting phase `bootstrap'
starting phase `patch-usr-bin-file'
starting phase `patch-source-shebangs'
starting phase `bootstrap-blue'
starting phase `configure'
starting phase `patch-generated-file-shebangs'
```

ROADMAP

- Pre-alpha (we are here)
- Alpha
- Beta
- Release

CURRENT FEATURES

- Pure Guile (no additional dependencies)
- Replay functionality
- Configurable preferences
- Autocomplete engine
- Blueprint export to JSON
- Extensible primitive types (through GOOPS)
- Hash-based tracking of buildables
- Time-based tracking of inputs/outputs
- Configuration based on system discoverability

NEXT FEATURES

- DAG traversal API
- DAG exporter/viewer (hoot?)
- Custom build scheduler
- WISP support
- Exporters
  - Ninja
  - Compilation database

LAPISLÁZULI

FOR YOUR FREEDOM AND OURS

Our projects are GPL compliant

MAIN REPOSITORIES

LGPL

- BLUE - Build Language User Extensible

GPL

- Bluebox - A Guix channel for BLUE
  (Implements the `blue-build-system`)
- blue.el - The Emacs interface for BLUE

PORTS

Guile

- Shepherd

Suckless

- ST
- DWM

Demos

- Hello world
- Hello world with embedded BLUE

END

Thanks for you attention