OMG! GERBILS!

WHAT IS GERBIL?

- Gerbil is an opinionated dialect of Scheme with a state of the art macro and module system.
- Inspired by Racket, but geared for Systems Programming -- Scheme in your server!
- But of course it is a great general purpose programming language.
- Gambit Scheme inside!

THE GERBIL DISTRIBUTION

- Gerbil includes a comprehensive standard library and a package manager for external packages.
- R7RS small is supported as a custom language prelude.
- R7RS large: currently Red edition, Tangerine coming soon.

GERBIL MODULES

- The fundamental unit of computation in Gerbil is the module.
- Each module has a prelude which provides initial bindings; default is the gerbil dialect.
- Modules export bindings that they want to make accessible to other modules.
- Modules can import other modules (at any phase in the meta-syntactic tower) to extend their namepsace and functionality.

AN EXAMPLE MODULE

vyzo switches to emacs, shows `:std/net/request`.

EXECUTABLE MODULES

- Excutables are just normal modules that export `main`.
- Compiled with the `-exe` (`-static`) option they generate binaries.

vyzo switches to emacs, shows `gxpkg.ss`.

THE PRELUDE

- The prelude is just another module!
- You can define your own custom language with a prelude that can even change the surface syntax.
- `:scheme/r7rs` is the R7RS prelude.
- There is a special `:<root>` at the base for empty modules that only have the core expander bindings.

A CUSTOM PRELUDE

vyzo switches to emacs, shows `gerbil-libp2p:libp2p/pb/identify` and `:std/protobuf/proto`.

THE CORE PRELUDE

vyzo switches to emacs, shows `:gerbil/core` and discusses the core dialect.

GERBIL MACROLOGY

- Preludes are full of macros!
- The macro system is based on `quote-syntax` (see Racket).
- `syntax-case` is implemented as a macro on top and is the canonical way to process syntax.
- Complex procedural macros have full access to the syntactic environment (see `syntax-local-value`).
- Expander API: `<expander-runtime>`.

MACROLOGY EXAMPLES

vyzo switches to emacs and shows macros relevant to actors.

THE STANDARD LIBRARY

- Batteries included for practical systems programming.
- generics, actors, http client and server, databases, key-value stores, cryptography, json, yaml, xml, OS interface, etc ...
- Lots of srfis.
- Philosophy: If something is useful for practical programming, we add it!

COMPILATION TO GAMBIT

- The Gerbil compiler compiles modules to idiomatic Gambit Scheme code.
- Compiler optimizes Gerbil idioms (direct dispatch, arity checking, match optimizations etc...)
- Gambit namespaces are used to control naming; each module corresponds to a namespace.
- Programmer can drop to raw Gambit code with `begin-foreign` special form.

COMPILER ARTIFACTS

- Compiler artifacts for a module `module` in package `package`:
  - `package/module__rt`: the runtime loader.
  - `package/module__0`: the runtime code.
  - `package/module__{n}`: the expansion time code (compiled macros!)
  - `package/module.ssi`: the module interface.
  - `package/module.ssxi.ss`: the optimizer interface.

COMPILER ARTIFACTS (2)

vyzo switches to emacs, shows compiler artifacts and generated Gambit code for `:std/net/request`.

COMPILING EXECUTABLES

- Dynamic executables:
  - Executable stub that loads the module runtime and executes `main`.
  - Require Gerbil distribution.

COMPILING EXECUTABLES (2)

- Static executables:
  - All dependent code is compiled together to produce a binary.
  - Full program optimization with `(declare (optimize-dead-definitions))`.
  - Binaries are independent of Gerbil distribution and can be shipped to your server.

FIND OUT MORE!

- Documentation: https://cons.io
- Source code: https://github.com/vyzo/gerbil
- IRC: `#gerbil-scheme` on freenode