SCRAPSCRIPT

COMING 2024

```
greet ◁ person:ron 3

. greet :: person → text =
  | :cowboy → "howdy"
  | :ron n → "hi " ++ a ++ "ron" , a = text/repeat n "a"
  | :parent :m → "hey mom"
  | :parent :f → "greetings father"
  | :friend n → "yo" ▷ list/repeat n ▷ string/join " "
  | :stranger "felicia" → "bye"
  | :stranger name → "hello " ++ name

. person =
  : cowboy
  : ron int
  : parent s , s = (: m : f)
  : friend int
  : stranger text


"hi aaaron"
```

Scrapscript is best understood through a few perspectives:

- “it’s JSON with types and functions and hashed references”
- “it’s tiny Haskell with extreme syntactic consistency”
- “it’s a language with a weird IPFS thing”

Scrapscript solves *the software sharability problem*.

Modern software breaks at boundaries. APIs diverge, packages crumble, configs ossify, serialization corrupts, git tangles, dependencies break, documentation dies, vulnerabilities surface, etc.

To make software safe and sharable, scrapscript combines existing wisdom in new ways:

- all expressions are content-addressible “scraps”

- all programs are data
- all programs are “platformed”

These simple guarantees produce new paradigms:

- Content-Addressible Everything
  - Worldwide Collaborative Namespace
  - No Broken Dependencies
  - Expression-Level Versioning
  - Time-Travel Interpreter
- The “Platform” Paradigm
  - Designed For Embedded DSLs
  - Self-Documenting Typed Configs
- Large As A Language; Small As A Message
  - All Valid Programs Return Valid Programs
  - Send Arbitrary Types Over The Wire
  - Send Unevaluated Sandboxed Programs
  - Comes With “Flat” Binary Representation
  - Magic Compression
- First-Class Network Requests
  - Serialization-Free Experience
  - Typecheck Across Network Bounds
- Tooling From First-Principles
  - Optimized For AI & Autocomplete
  - Snippets On Steroids
  - Seamlessly Publish And Partake
  - Hosting, Accounts, And Payments
  - Brand New Browser

Content-Addressible Everything

Any chunk of the language can be replaced with a hash.

These chunks are called “scraps”.

Scraps are stored/cached/named/indexed in global distributed “scrapyards”.

Worldwide Collaborative Namespace

```
31 ▷ janedoe91/fibonacci

1346269
```

```
31 ▷ #sha1$e4caecf0d6f84d4ad72e228adce6c2b46a0328f9$0

1346269
```

Scrapscript rejects traditional package-management. Instead, “scrapyards” combine features from Smalltalk, Hackage, IPFS, GitHub, and StackOverflow. This new paradigm empowers devs to safely collaborate in live environments.

No Broken Dependencies

Every scrap carries its own immutable dependencies.

The language itself forms merkle trees; VCS tools like `git` are optional. Every expression is independently version-controlled through the global namespace.

Expression-Level Versioning

```
pair
(spaceq/is-planet@2005 "pluto")
```

```
(spaceq/is-planet@2006 "pluto")
```

```
pair true false
```

Every expression in the ecosystem can be independently spliced and “time-travelled”.

To avoid giant updates, scrapscript tooling can incrementally upgrade your code. Any chunk of code can be pinned independently to upgrade at a later time.

Time-Travel Interpreter

```
$ echo 'spaceq/is-planet "pluto"' | scrap eval --t="2005-01-01"
```

```
true
```

```
$ echo 'spaceq/is-planet "pluto"' | scrap eval --t="2006-12-31"
```

```
false
```

Easily inspect code regressions. Execute code with dependencies from a specific point in time.

The “Platform” Paradigm

Scrapscript acts as an algebra for performant “platforms”.

By embracing “managed effects” (like Elm and Roc), scrapscript stays small and simple.

Designed For Embedded DSLs

```
h1 [] [ text "hello world" ]
. { h1, text } = luffy88/html-tags
```

```
| "/home" → q → res:success ◁ "<p>howdy " ++ name ++ "</p>"
  , name = q ▷ dict/get "name" ▷ maybe/default "partner"
| "/contact" → _ → res:success "<a href="mailto:hello@example.c
| _ → _ → res:notfound "<p>not found</p>"
. res = : success text : notfound text
```

Platforms are flexible! Use scrapscript as a web server, templating language, shell, compilation target, tiny embedded OS, query language, or anything imaginable.

Self-Documenting Typed Configs

```
my-org:my-config
{ name = "my-server-001"
, cpus = :4
, mem  = :16
}
. my-org = : my-config
  { name = text
  , cpus = : 1 : 2 : 4 : 8
  , mem  = : 1 : 2 : 4 : 8 : 16 : 32
  }
```

Large As A Language; Small As A Message

Scrapscript is a full programming language designed to be sent over the wire with type-safety in mind.

All Valid Programs Return Valid Programs

```
$ echo 'my-type:left . my-type = : left : right' \
> | scrap eval \
> | scrap eval \
> | scrap eval


my-type:left
. my-type =
  : left
  : right




$ echo 'ok (42 + 1)' \
> | scrap eval --result \
> | scrap eval --result \
> | scrap eval --result


ok 43




$ echo 'ok (42 + "apple")' \
> | scrap eval --result \
> | scrap eval --result \
> | scrap eval --result


err [ eval/type-error "+" "int" "text" ]
```

Scrapscript is small enough to be its own complete datatype.

Every scrap carries its own custom types. Stale references are simply impossible.

Programs can be chained and transformed in completely new ways. Pass your scraps through linters and optimizers in simple pipelines.

Send Arbitrary Types Over The Wire

```
animal:horse "Lucy"
. animal =
  : horse text
  : zebra int
```

Let the computers communicate which types they're using.

Don't waste engineering hours juggling types and serialization across different machines.

Send Unevaluated Sandboxed Programs

```
quang77/nth-digit-pi 420000000000
```

Scrapscript programs are safe to send around.

Many client/server relationships can be radically simplified by skipping serialization.

Comes With "Flat" Binary Representation

```
$ echo '3 * 5' | scrap eval | scrap flat | hexdump -C

0F


$ echo 'true' | scrap flat | hexdump -C

C3


$ echo '[ false, true ]' | scrap flat | hexdump -C
```

```
92C2C3
```

Put programs into JSONB-sized packages. Scrapscript fits into msgpack.

Magic Compression

```
$ echo 'sarahsmith65/very-large-video' | scrap flat | hexdump -C
```

```
D8A196C4BC3A1139B2413CBE2EBECA8F3B754166450E
```

Instead of sharing large dumps of data, you can send references to any data anywhere.

By sending references, other machines can opt to pull the data from cache or high-speed CDNs.

First-Class Network Requests

By leveraging scraps-as-messages, scrapscript explores new networking paradigms.

Scrapyards enable new compile-time primitives for verifying type-safety across network boundaries. “Contracts” are automatically inferred and enforced between clients, servers, and external APIs.

Serialization-Free Experience

```
$ echo "@hucksternews/frontpage 3" | scrap platform task
```

```
[ "https://ciechanow.ski/mechanical-watch/"
, "http://www.paulgraham.com/todo.html"
, "https://sive.rs/hellyeah"
]
```

Scrapscript automatically serializes and deserializes scraps across any API boundaries. The system doesn't care whether you use IPC, HTTP, QUIC, email, etc.

Typecheck Across Network Bounds

```
$ echo "@rebbit/users 42" | scrap eval
```

```
error: @rebbit/users expects type rebbit/users-request
```

The scrapscript compiler tells you when remote APIs differ from the code. And if the API changes while the code is running, scrapscript offers a series of graceful handling options.

Tooling From First-Principles

Scrapscript vertically integrates editors, VCS, configuration, platforms, payments, data, and cloud infrastructure.

Optimized For AI & Autocomplete

```
f a b
. f = | x → y → x * y
```

```
. a = 1
. b = 2
```

Scrapscript encourages wishful thinking.

Declare your goal up-front, and let your tooling make educated guesses about how to get there.

Snippets On Steroids

```
$ echo "`633b327df5e54bb626300a19a459b7bd81cce3ad13f72aa395df41e0
| scrap save "my-key"
```

Save your scraps in scrapbooks to privately sync across your devices.

Use team scrapbooks to collaborate on code in a live environment.

Seamlessly Publish And Partake

```
@yard/publish my-key "greet" "| _ → \"hello\""

task:success ()


connie2036/greet "hi"

"hello"


@yard/get "connie2036/greet"
```

```
task:success "greet" "| _ → \"hello\""


@yard/delete my-key "greet"

task:success ()
```

Scrapyards store scraps in an IPFS-like system with name and versioning information.

Hosting, Accounts, And Payments

Development doesn’t need to be difficult. Scraplab will offer the best features of the following services in an integrated experience:

- Stripe/Gumroad/Patreon
- Netlify/Fly.io
- GitHub

Brand New Browser

The scrapland browser turns every scrap into its own interactive page.

Scrapscript was designed by Taylor Troesh.

Say hello if you want to receive updates or join the team.