Software Interlocking Blocks

SWIBs and Receptors and Black Boxes and the Future of Programming

Paul Tarvydas, May 2024

Goals

- Simple Expressivity for Programmers, incl. Systems Programmers
- Breaking Free of the Function-Based Paradigm
- Abstraction, Namespaces, Concurrency, Black Boxes, etc., etc.
- Multiple notations, not just a single notation.
- Hardware focus instead of functional notation focus
- LEGO®-like software blocks
- Compiling diagrams to code is easy, why stick to text-only notations?

Textual Notation

- Excellent for expressing computation, *i.e. calculators*
- No need to replace text with any other kind of notation
- Too restrictive for non-computation, *e.g. distributed computing, internet, robotics, blockchain, etc.*
- FoP instead of FoPoP … wanted - *Future of Programming vs. Future of Past of Programming ; different problems (biases) in Past than Now and Future (old fashioned ideas about time-sharing, memory sharing vs. today's inexpensive CPUs and inexpensive memory)*

Textual Notation - Restrictions

- Ad-hoc blocking - *caller must wait for callee*
- Routing - *callee must send value back to caller, only*
- Excessive coupling - *functions call other functions (instead of sending messages)*
- 1:1 - 1 input, 1 output - *destructuring only creates the illusion of multiple inputs & outputs*

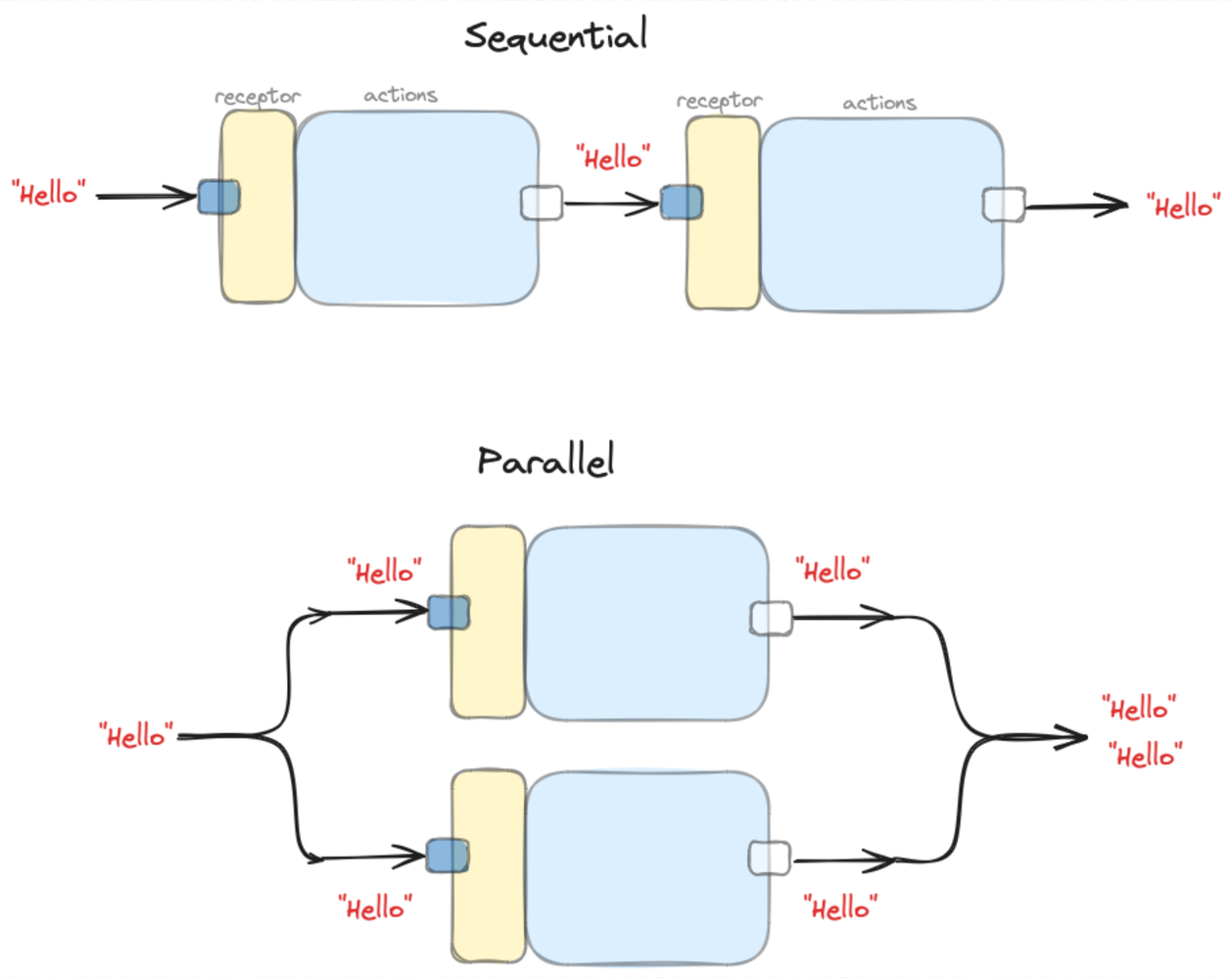
Textual Notation - Restrictions (2)

- Everything included in single notation - *GPL*
- Time dimension ignored - *how to express sequencing instead of computing?*
- Ignores hardware in lieu of notation worship - *hardware can do more than textual notation allows for*
- Text-based “async” is not true async - *step-wise simultaneity is not asynchronous*
- Backwards - *notation for analysis drives programming, instead of programming driving analysis*

Textual Notation - Restrictions (3)

- Single technique for abstraction - *lambdas and parameterization - everything looks the same regardless of semantic importance - prefer to use different notations appropriate to each semantic concern*
- Inheritance OK for expressing structure of data
- Inheritance Not OK for expressing control flow - *need parental authority, not inheritance, children must not override behaviour of parents ; original intent of COND was to express conditional values of functions, not as a mixin with variables to create varied control flows*
- Original intent of CPU hardware was single threading - use multiple CPUs if multiple threads needed

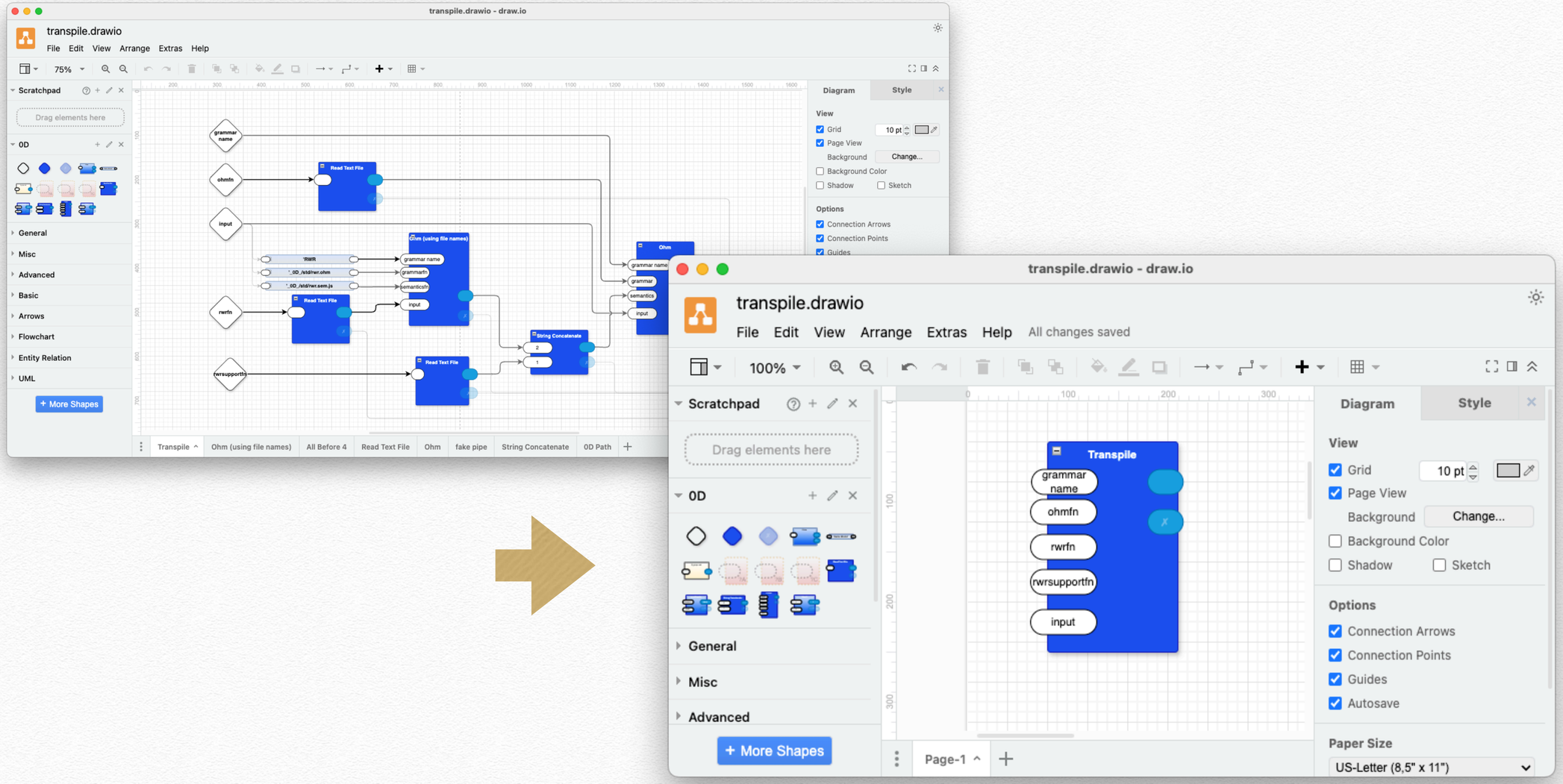
Concurrency is Simple



Concurrency is Simple Continued

- Don’t need thread libraries
- Anonymous functions, e.g. closures, are enough
- FIFO queues instead of LIFO callstack for concurrency
- Loops & recursion don’t make sense in async paradigm

Abstraction



Abstraction (2)

- Abstraction requires fan-out *avoided when analysis drives programming*
- Enables layered design

Prerequisites for SWIBs

- Message passing that doesn't use the callstack
- Hierarchical, recursive message dispatcher (0D)
- Extreme Isolation - data isolation *and* control-flow isolation

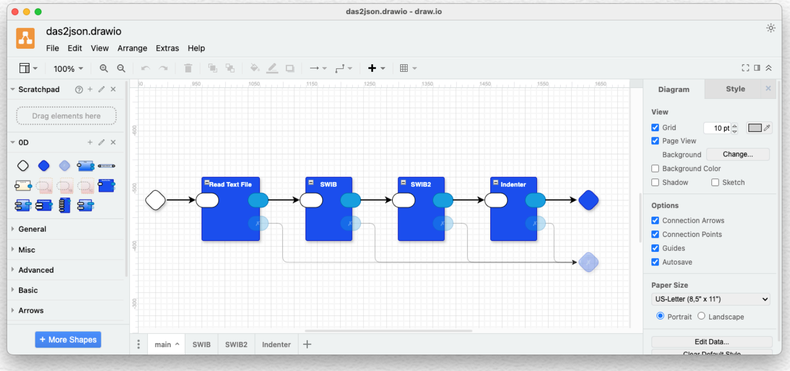
Designing DPLs

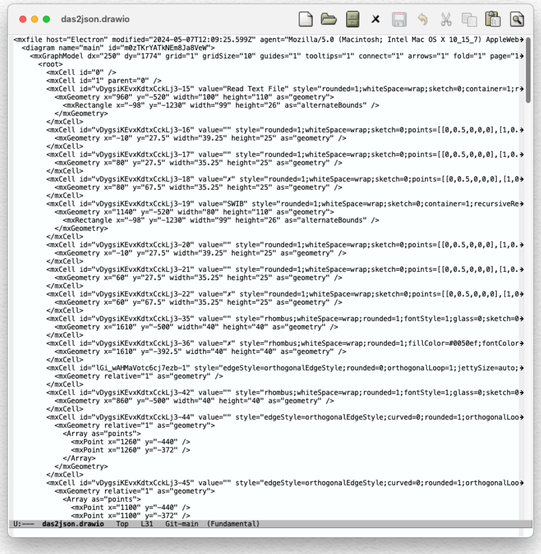
- Choose a small subset of simple graphics, akin to using only a subset of English for textual programming languages
- Rectangles, ellipses, arrows, text, grouping
- If editor does not provide convenient semantic info, use simple math and infer the semantic info
  - e.g. *intersection*, *containment*, *bigger*, *smaller*, *left-of*, etc. for-loops, dictionaries, factbases, Prolog, etc.

Compiling SWIBs to Code

- Use graphic editors that save diagrams in text format, e.g. XML, graphML
- Use text parsing tools to parse XML, e.g. OhmJS
- Feed parse to existing languages, e.g. Python, then let existing language compiler do the rest of the work
- t2t - text to text translation using PEGs *or* macros
  - e.g. XML ☞ Python, *or*, XML ☞ .json ☞ diagram interpreter

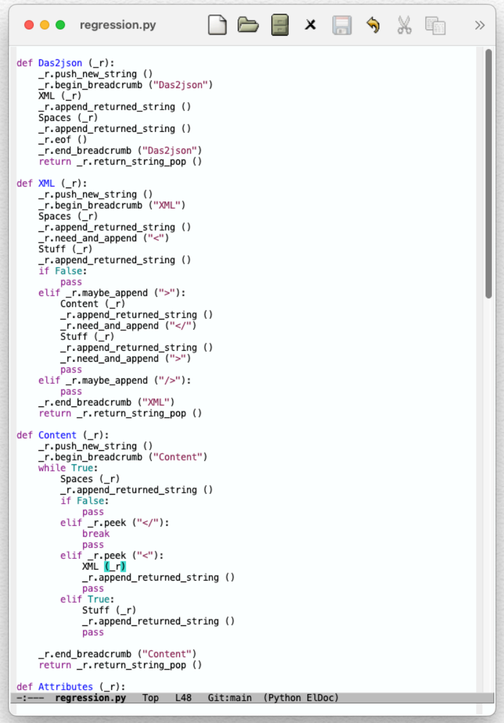
Example: Diagram Compiler





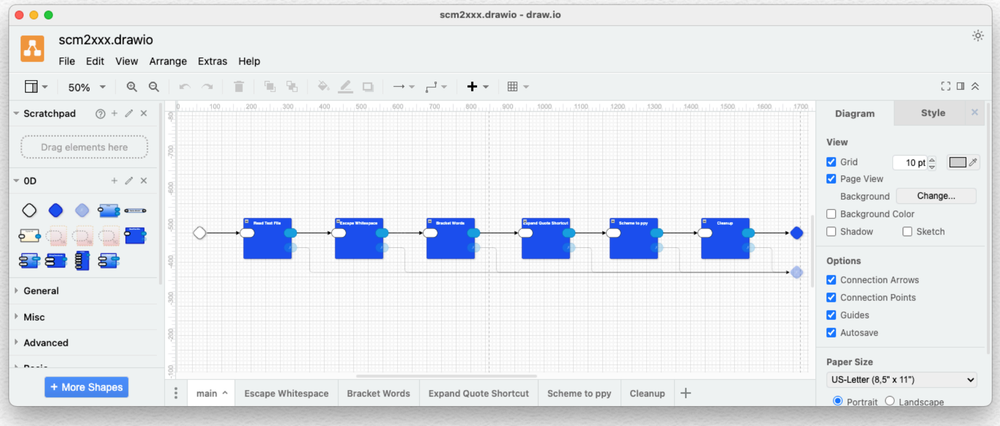
OhmJS
OhmJS+rwr
etc.

Python

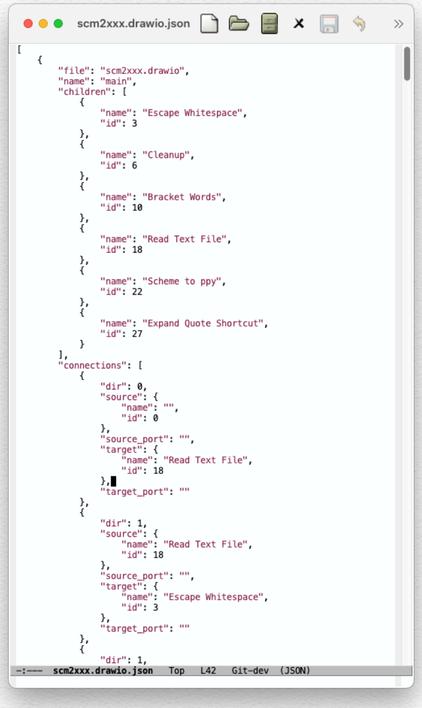


WIP https://github.com/guitarvydas/das2json

Example: Diagram Interpreter



JSON



odin0D
py0D

WIP https://github.com/guitarvydas/scm2js0d

Das2json Compiler

swib.ohm

```
swib {
  main = spaces rule rule*
  rule = "<" spaces rulename "=" spaces pattern+
  pattern = stringMatch | endop | cond | peekcond | cycle | rulecall

  endop = "_end" spaces

  rulename = name spaces

  rulecall = name spaces

  stringMatch = string spaces

  string = dq notdq* dq
  dq = "\""
  notdq =
    | "\\" any -- escaped
    | ~dq any -- raw

  cond = "[" spaces condClause+ "]" spaces
  peekcond = "[+" spaces peekCondClause+ "]" spaces
  peekCondClause = "|" spaces peekCondMatch ":" spaces action*
  condClause = "|" spaces condMatch ":" spaces action*
  condMatch =
    | string -- string
    | endop -- endop
    | "*" -- else
  peekCondMatch =
    | string -- string
    | endop -- endop
    | "*" -- else
  action = break | acceptAndAppend | pattern
  break = "_break" spaces
  acceptAndAppend = "." spaces

  cycle = "<<<" spaces pattern+ ">>>" spaces

  name = firstLetter moreLetter*
  firstLetter = letter | "_"
  moreLetter = digit | firstLetter
}
```

swib.rwr

```
swib {
  main [spaces firstRule moreRule*] = '+firstRule+moreRule+

import receptor
_r = receptor.Receptor ()
[+firstRule+] (_r)
s = _r.pop_return_value ()
print (s)

  rule [_colon spaces rulename _eq spaces2 pattern+] ='
def +rulename+ (_r):(-
    _r.push_new_string ()
    _r.begin_breadcrumb ("+rulename+")
    +pattern+_r.end_breadcrumb ("+rulename+")
    return _r.return_string_pop ()-)\n'

  pattern [p] = '+p+'

  endop [_end spaces1] = '_r.eof ()\n'

  rulename [name spaces1] = '+name+'

  rulecall [name spaces1] = '+name+ (_r)\n_r.append_returned_string ()\n'

  stringMatch [s ws] = '_r.need_and_append (+s+)\n'

  string [dq1 notdq* dq2] = '"+notdq+"'
  dq [c] = '\"'
  notdq_escaped [_bs c] = '+_bs++c+'
  notdq_raw [c] = '+c+'

  cond [lb spaces1 condClause+ rb spaces2] = 'if False:(-\npass-)\n+condClause+'
  condClause [_bar spaces1 condMatch _colon spaces2 action+] = 'elif +condMatch+:(-\n+action+
  condMatch_string [s] = '_r.maybe_append (+s+)'
  condMatch_endop [x] = '_r.eof ()'
  condMatch_else [x] = 'True'

  peekcond [lb spaces1 condClause+ rb spaces2] = 'if False:(-\npass-)\n+condClause+'
  peekCondClause [_bar spaces1 condMatch _colon spaces2 action+] = 'elif +condMatch+:(-\n+ac+
  peekCondMatch_string [s] = '_r.peek (+s+)'
  peekCondMatch_endop [x] = '_r.eof ()'
  peekCondMatch_else [x] = 'True'

  action [a] = '+a+'
  break [_break spaces1] = 'break\n'
  acceptAndAppend [_dot spaces1] = '_r.accept_and_append ()\n'

  cycle [lb spaces1 pattern+ rb spaces2] = 'while True:(-\n+pattern+-)\n'

  name [firstLetter moreLetter*] = '+firstLetter+moreLetter+'
  firstLetter [c] = '+c+'
  moreLetter [c] = '+c+'

}
```

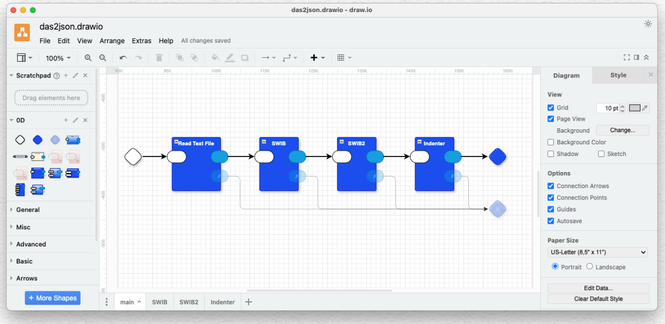
defname.ohm

```
defname {
  main = text+
  text =
    | defName -- match
    | any -- other
  defName = "/" spaces "def" spaces name spaces through<closeindent> spaces "/"
  through<s> = (~s any)+  s
  name = letter alnum* ~alnum
  closeindent = "-)"
}
```

defname.rwr

```
defname {
  main [x+] = '«x»'
  text_match [x] = '«x»'
  text_other [x] = '«x»'
  defName  [lb spaces1 _def spaces2 name spaces3 misc spaces4 rb] = '«name»'
  through [misc s] = ''
  name [letter alnum*] = '«letter»«alnum»'
  closeindent [rb] = ''
}
```

das2json.drawio



das2json.swib

das2json.swib

```
: Das2json =
  XML Spaces _end

: XML =
  Spaces "<" Stuff
  [
    | ">": Content "</" Stuff ">"
    | "/>":
  ]

: Content =
  <<<
    Spaces
    [*
      | "</": _break
      | "<": XML
      | *: Stuff
    ]
  >>>

: Attributes =
  <<<
    [*
      | ">": _break
      | "/>": _break
      | _end: _break
      | *: Stuff
    ]
  >>>

: Stuff =
  <<<
    [*
      | ">": _break
      | "<": _break
      | "/>": _break
      | _end: _break
      | *: .
    ]
  >>>

: Spaces =
  <<<
    [*
      | " ": .
      | "\t": .
      | "\n": .
      | *: _break
    ]
  >>>

: String =
  "\"" NotDquotes "\""

: NotDquotes =
-:---  das2json.swib   Top   L49   Git:main   (Fundamental)
```

0D

das2json.py

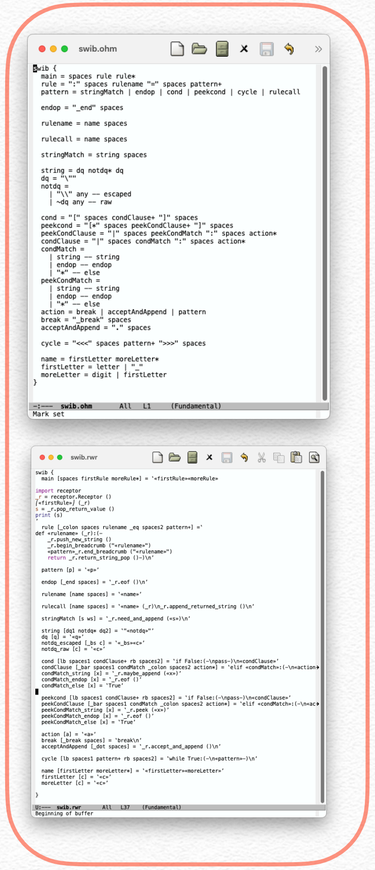
regression.py

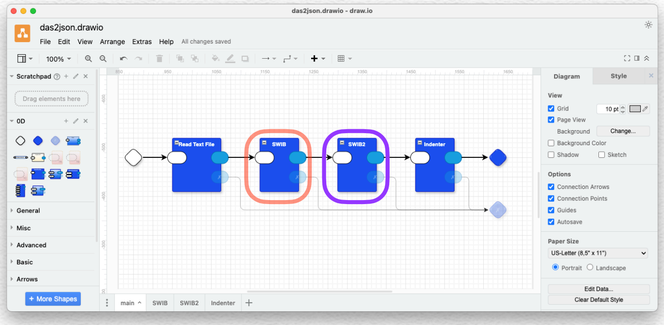
```
def Das2json (_r):
    _r.push_new_string ()
    _r.begin_breadcrumb ("Das2json")
    XML (_r)
    _r.append_returned_string ()
    Spaces (_r)
    _r.append_returned_string ()
    _r.eof ()
    _r.end_breadcrumb ("Das2json")
    return _r.return_string_pop ()

def XML (_r):
    _r.push_new_string ()
    _r.begin_breadcrumb ("XML")
    Spaces (_r)
    _r.append_returned_string ()
    _r.need_and_append ("<")
    Stuff (_r)
    _r.append_returned_string ()
    if False:
        pass
    elif _r.maybe_append (">"):
        Content (_r)
        _r.append_returned_string ()
        _r.need_and_append ("</")
        Stuff (_r)
        _r.append_returned_string ()
        _r.need_and_append (">")
        pass
    elif _r.maybe_append ("/>"):
        pass
    _r.end_breadcrumb ("XML")
    return _r.return_string_pop ()

def Content (_r):
    _r.push_new_string ()
    _r.begin_breadcrumb ("Content")
    while True:
        Spaces (_r)
        _r.append_returned_string ()
        if False:
            pass
        elif _r.peek ("</"):
            break
            pass
        elif _r.peek ("<"):
            XML (_r)
            _r.append_returned_string ()
            pass
        elif True:
-:---  regression.py   Top   L37   Git:main   (Python E
```

Re-Parsing Generated Code





Need to extract info (*name* in this example) from generated code - use 2nd parser instead of special-case data structures and return values.

WIP Projects

- Kinopio to markdown (“mind-map” automatically reorganized as markdown (for ChatGPT?)
- Dungeon Crawler game prototype (based on example game in Ceptre paper)
- Arith0D - grammar snippet compiled simultaneously to Python/Javascript/Common Lisp/WASM
- Visual SHell - VSH
- Scheme to Javascript transpiler, Scheme to Python transpiler
- LLM wrapper
- SWIB generator from diagrams
- new syntax for find-and-replace using HTML markup for colorization and fontification
- Invention of new streaming parser language Receptor for SWIBs (das2json compiler)