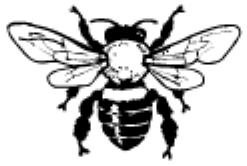
2016-12-3 computer API protocols



Guide to Distribute Services Across Top-Level Entities as Values, Methods and Variables:

§ intro project bone folding

Simple rule: for .something [something] is always a value
Something. [something] is always a variable
[.] is referred to a contextually to as 'obtect dot' 'dot to' and is a location provider forming the basic and universal and non unique or generally arbitrary pattern of thisTing.js (the naming convention is pretty standard but the implementation is arbitrary, as no protocols exists or could exist until now or resume.pdf where the object dot notation expresses a variable like the standard pattern:

Property:value

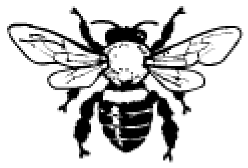
Since this pattern is not only predictable

the syntax (variation) arbitrary : we can represent this via any simple constructor
Like grammar syntax:
object : reference : action

Example: pollen dot and stage dot method and dot computer and dot host values

Such as that the pattern that objects exist prior (as in notationally a representation semantically, and or also

Wow! There are thousands of JS libraries! How is this one special/unique in any way?



What makes this one different ?
That’s easy: a simple replicable structure is only as unique as its values.

The availability of the .computed value is extremely limited, only when correspondence within a system, of meaning of key pair values are likewise dot notation object locations e.g. intelligence.computer.action.css where ‘action’ is a user assigned variable and Intelligence computer css reference standard language syntax: in this case a url:
Like: http://intelligence.computer.action.css which resolves the following:
https://pollen.computer.intelligence.computer.action.css

The repetition of the .computed value is distributed (twice referenced)

Where .computer is a commonly available value (distributed across 11 nodes, where individual nodes such as the root node: pollen.computer contan indivual / unique values)

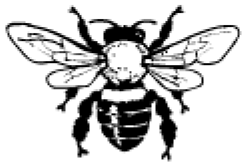
And likewise there are common variables (distributed simmilarity beteen unique variable objects like stage.email and stage.zone) and a unique distribution of expressions between values and variables and simmilar variables acroos a likewise (or, unlike) set of values. (See figures of this document)

As far as I can tell this is a completely unique and un used method, the availability of which corresponds solely on the proximity to top certain top-level domain objects,

For instance:
If .com and .org are the only available top-level objects it makes no sense semantically to take advantage of the . notation method, it would make no sense if .com was a value compared to .org

But when . email is a value and .stage is a common variable it makes sense for an API to use these to create sophisticated methods, because is is easy for the user to understand the .computer or .email value over the mostly 100% arbitrary .com value.

2016-12-3 computer API protocols



I will sow distributed work can be done across a unique system of urls I will describe: And I will argue the efficacy of system is only as good or as usable as its key pair parameters makes sense to the user and constitute a legible and productive pattern of objects, methods, attributes, values and actions.

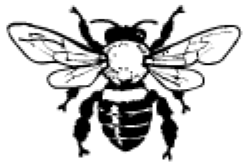
In these sense the pollen.computer project is extendable due to the robustness of the key terms whears the pattern is fecundable: the meaning, via the obvious and usability of the pattern is not: that makes this API uniue as well as unwritable in a more meaninful way: (ccounting for the rarity of the for which no more avalible meaningful patters exist, such as for the avaliability of the .computer value

So, the values in key pairs are always top level objects, (like .zone or .cloud the question is which one (see figures in this document)

So what I have done is removed a layer, attributing top level objects to values (when a .to object like .com is the attribute of a value key pair [usually variable:value, here sometimes variable.value])
So yes I am changing the syntax: but we can just pretend the ^upper . is there for : and this makes our code more readable, more obvious and usable, and I believe is conforms to the spec, why else is their object specified in the available top level domain objects if it is not to increase the productivity of our protocols and make code more readable
So if .bicycle is an available top-level domain object, and John creates a john.bicycle, an app tracking data on his bicycle, he can... If tilde press wants to use the domain: entity: tilde.press and create a more specific object for its users such as tilde.press/deadlines/submissions or even get a meaningful specified email that is not a .com, such as submissions@tilde.press they can... thanks to availiable domain objects and a little thinking.

Using this . notation entails a variable location object (commonly .stage .pollen)

So it is not a computer that is distributed over the internet, but it is a computer that runs on the internet via protocols between languages rather than a computer that is distributed across the internet, and using available languages. No it uses available



standards, which are protocols distinct between the operation of languages, other languages and often other protocols, thus this constitutes a set pattern of such and a specif of protocols (as object literals)

pollen.computer : a top-level domain distributed computer (see security.flowers), a set of protocols:
Goal(s): solve restrictions and problems of existing technologies
Solution: build new ones!
https://pollen.computer is a set domain (~50) protocols:
Each api performs 3 tiers of functions:

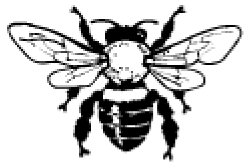
Top-level tier: is open as a resource, i.e., is a repository of its own documents, it servers secure resources to clients via the https:// protocol (http over ssl)

Mid-range tier: protocols for each API work as token of the services offered by that API and these create its methods by virtue of their own object literals used in expressions. The resolution of each api parameter depends on the use case: the mid rage use is as token operators in a distributed system.

Deep tier: the deep tier API use case is for pollen.computer specific process, such as that require the security.flowers API to function like the compose.computer method in when deep tier use of the compose.computer api method is needed.

Each API is completely open-source, under the MIT license, and free to use.

Users may wish to engage with the project in various ways such as going to the domain locations in the browser, reference their static or dynamic content like scripts, use the set of



methods via the object method pattern, use the deep tier of any or all of the APIs to build a new pattern or may create a completely unique arrangement and use.

§ Examples: A Basic Pattern:

```
https://pollen.computer.(object).(method)/{{(object)}}/{{(object)}}/action.script
example:
pollen.computer.stage.emall/{{project}}/{{directory}}  or

pollen.computer.stage.email/template/newsletter/
```

And then action.script notation such as : terminate.rby

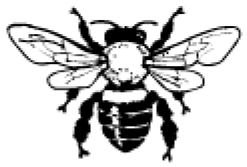
Basic pattern example 2:

Top-level use: pollen.computer/css/animate.css
Mid-level use: pollen.computer.stage.exchange
Deep use: pollen.computer.composer.computer.intension.computer
(simplifies to pollen.composer.intension or composer.intension)
intenstion.computer is the bug tracking API and composer is the build api for that unit, but there are other internsion. API like .agency, .cloud, .press so a string could look like this for accessing these API for a certain use case:

pollen.computer.composer.intension.press [object→ (action) → (object action) syntax]
So as you can see the api are chainable via this notation

So there are highly open parameters to what the computer will accept
This makes the code functional readable and secure since it is in one dimension built in with traditional code languages, like java script, in one dimension an extended browser: (failing to execute, and a 3rd dimension, secure:=> the system fails to operate without a connection

The methods define proxies in other ways not seen in coding and not as extendable as this, a native set of extensions, like a js library but different in that it is a javascript computer

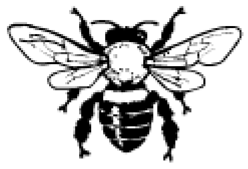


§ **APIs** : and available methods for pollen.computer
(in no particular order)

vermont.computer person.computer candy.computer intension.computer intelligence.computer intellegence.computer speak.computer number.computer artifact.computer beethoven.computer bach.computer maozart.computer	tilde.party tilde.press tilde.host tilde.cloud pollen.agency pollen.zone pollen.cafe pollen.ltda pollen.blue pollen.pink pollen.healthcare pollen.help pollen.host pollen.solutions pollen.tips intension.agency intension.cloud intension.press	security.flowers mutation.technology respolyn.xyz respolyn.com rhamses.xyz polytec.xyz stage.email stage.exchange stage.healthcare stage.legal stage.lgbt stage.zone stage.cafe stage.ltda union.codes union.institute science.codes mutation.codes svg.codes

Demo method use - get script function:

```
$.getScript(https://pollen.computer.stage.email),
//Or
$.getScript(https://pollen.computer.mutation.technology/access.xml),
```

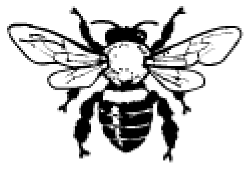


§ API Structure

bind method-object:
security.flowers

API GROUPS:

Objects and actions: **(location API on pollen.computer)**	**Unique Objects and actions alias**	**Methods:** **(actions on pollen.computer)**
intelligence.computer speak.computer number.computer artifact.computer pollen.solutions pollen.tips stage.cafe composer.computer	intension.computer intension.agency intension.cloud **(Data and repository:)** beethoven.computer bach.computer mozart.computer	pollen.help pollen.host pollen.zone stage.zone stage.email stage.exchange intension.press (composer.computer alias)

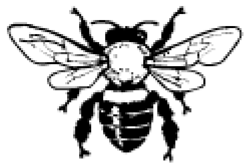


§ API TYPES:
Actions oriented methods: pollen. /pollen{{.script}}
Expression Patterns:
pollen.object.pollen.object
object.computer.object.computer
object.computer.pollen.object

Listed A - Z

{{}}.attribute **(# indicates frequency)**	**component.{{}}**
.agency (3) .blue .cafe (2) .cloud (2) .computer (11) .codes (4) .email .estate .exchange .flowers .healthcare (2) .host (2) .help	artifact. beethoven. bach. composer. intension. (4) intelligence. mozart. mutation. number. pollen. (12) respolyn. rhamses. speak. stage. (8)

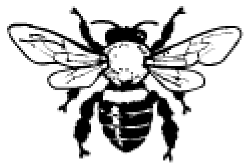
2016-12-3 computer API protocols



.ltda (2) .lgbt .party .pink .press (2) .solutions .tips .town (to come potentially) (5) .technology .xyz (3) .zone (2)	security. science. svg. tilde. (5) union. vermont. (3)

{{}}.attribute (# indicates frequency)	**Variable Component.{{}}**
.computer (11) .town (5) .codes (4) .agency (3) .xyz (3) .cafe (2) .cloud (2) .healthcare (2) .host (2) .ltda (2) .press(2) .zone (2) .blue .email .estate .exchange .flowers .help .lgbt .party .pink .solutions	pollen. (12) stage. (8) tilde. (5) intension. (4) vermont. (3) artifact. beethoven. bach. composer. intelligence. mozart. mutation. number. respolyn. rhamses. speak. security. science. svg. union.

2016-12-3 computer API protocols



.tips .technology	

Fig: API method distribution

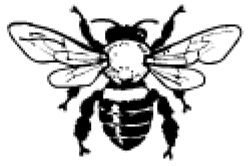
Important .properties	Important objects.	Notable top-level entities
.computer (11) .codes (4) .agency (3) .cloud (2) .host (2) .press(2) .zone (2) .email .exchange .help .solutions	pollen. (12) stage. (8) intension. (4) artifact. composer. intelligence. mutation. number. speak. security. svg. union.	pollen.computer pollen.agency pollen.host pollen.help pollen.solutions stage.cloud stage.zone stage.email stage.exchange intension.computer intension.agency intension.cloud internsion.press

Chaining acceptable parameters examples:

```
stage.email.press
svg.pollen.help
```

```
.security.press.svg.codes.artifact.intelligence
```

(calls security from composer with .press on the ‘svg.codes’ object (creates a record with .artifact and assigns a deep tier .intelligence record/alias (since .security and composer are working on deep tier)



stage.exchange.computer.composer.press

This exchanges files on the stage via the attachment of the .composer alias (creating a record) with .press (pressed,/ or passed back to to the bug tracker API (composer.computer)

§ composer.computer API

composer.computer uses deep tier access to these APIs

Data and repository: pairwise redundant	**Bug Tracking and QA resources**
beethoven.computer bach.computer mozart.computer	intension.computer intension.agency intension.cloud intenstion.press

The composer. API method is available without deep access

§ Dot Notation

Standard dot notation uses the space before the dot as a location/space parameter.
The pollen.computer protocol uses dot notation to access specific versions of a certain function that is defined in a different class or a different module.

The **.computer** property is the most common across the system with 11 property locations Likewise **pollen.** and **stage.** are the most referenced object entities with 12 and 8 method locations respectively. (see: Fig: API method distribution (above)

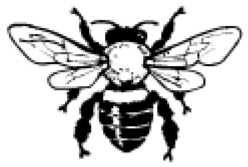
Seven other APIs have high entities (2-4 location each) These, primary APIs support important methods for of the stage. object -- the pollen. object and the composer.computer (the build API)

§ Definitions:

pollen.computer.security.flowers [method]

pollen.computer.artifact.computer
pollen.computer.speak.computer
Pollen.computer.number.computer

2016-12-3 computer API protocols



pollen.computer.beethoven.computer
pollen.computer.bach.computer
pollen.computer.mozart.computer

pollen.computer.mutation.codes
pollen.computer.svg.codes
pollen.computer.mutation.technology

pollen.computer.pollen.solutions
pollen.computer.stage.zone
pollen.computer.stage.healthcare
pollen.computer.stage.exchange
pollen.computer.stage.email
pollen.computer.intension.cloud
pollen.computer.pollen.tips
pollen.computer.intelligence.computer
pollen.computer.intellegence.computer

In some cases an object literal will be passed in --- >

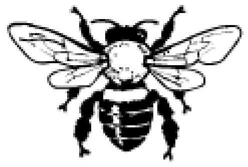
$.getScript(https://pollen.computer.mutation.technology/access.xml),
Is $.getScript(https://mutation.technology/access.xml),
Or $.getScript(mutation.technology/access.xml),
    $.get mutation.technology/ access

§ Overview of API Structure:

The security.flowers method is the most crucial element. (defined as a Binder method-object:)

Notice the various action methods: .zone .help .host

2016-12-3 computer API protocols



And reference methods: stage. pollen. They re all pollen to or stage to methods and one intension.press method

A hierarchy of nodes to security.flower exists so that the intension.computer .agency .cloud .press objects are objet literals of security.flowers
as is the pollen.computer ordinance of methods objects and aliases [intension is an alias object (a QA device servering and working under security.flowers]

Composer.computer si the cache entity validatior and service provider workign with permissions to run the bach. mozart. beethoven.computer protocals providing asyncronization services and data intergration (across the suite) , ststem backup and storage, and artificaial matinence such as code repositiory and other server devices → such as serving up public code...
security.flowers to provider
Composer.computer aslo handels the validations and protocols of the intenstion. methods , objects and actions.

1) Security.flowers
2) Pollen.computer & composer.computer

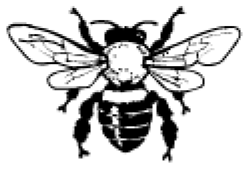
(for the bug tracking api methods and composer.computer methods)

A pollen.solution object

A structure of mutation.technology
Not a collection of domains but a technology

security.flower bind pollen:numbers.computer

2016-12-3 computer API protocols

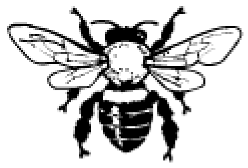


Leah this morning a catastrophic thing happened : a break through.
I have been working on this project for months and months but it finally started to take shape in large part to you lectures!
I have an idea called pollen.computer, an api : https://pollen.computer
The structure is basic
Say there are 50 additional apis : .computer or pollen. patters like: pollen.computer or pollen.host, or pollen.solutions, or numbers.computer, artifact.computer
The names of the API correspond to a .reference or a reference. notation that is conditionally in a system of methods and objects, a pattern, and as system themself, each a pattern or specifically a set of libraries at the host level, for example : https://pollen.help, is available as a service of pollen.computer via (https://pollen.computer.pollen.help) and, as such it can be rendered as a secured unsecured local service as in a local pollen.help service or as a secured service of pollen.computer via secuirty.flower

security.flower is an api that authenticaties and secures token for services to and from pollen.computer exchanges, such as for databse queries, unless the pollen.computer api is itself hosted as a proxi (without security.flower) (such as for the purposes of testing or building).
With security.flower the composer.computer hosted systems are instored, but obviously https://composer.computer is a public and open souce api composer.computer can become a prpriatoy token when authenticated via https://security.flowers an open souce api, security.flower services are avaliable to run or host locally (as a proxy) and are accassable via the .computer objects and methods.
However only security.flower.composer.computer methods are valid (or some other valid method system) (for the bug tracking api methods and composer.computer methods)

It's not a new language but it’s an api, with a set of methods and objects, which correspond to a pattern which are top level domain entities, whose names are function carriers distributed across that pattern of objects and methods, constituting their organization, functional, programmatically and importantly, the peer wise and hierarchically locations of each api into a body of documents and protocols, all accessible via a standard pattern of
opject → https://stage.email (go to this website and use its API user services)
object.method → stage.email (use this API’s methods and a .object or method. notation)

2016-12-3 computer API protocols



object.method.object/action → stage.email/killPermissions.exe (chaniable syntax).
{dependening on the operator's set paramaters} (chiain this object to higher functions such as pollen.computer functions like security.flowers api or the bug tracking and data api composer.computer with redundant (distributed) component services: and one avaliable .press method:
Intension.press

security.flowers and composer.computer both are a method/object and are a build api

composer.computer services

(composer.computer API services)
pollen.computer.beethoven.computer
pollen.computer.bach.computer
pollen.computer.mozart.computer
Pollen.computer.polllen.js (attribute.object.attribute.object)

https://pollen.computer.(object).(method)/{{(object)}}/{{(object)}}/action.script

.object or object.
.method or method. expression syntax (which are exchange with their api at various level) via https:// for api documents and record objects, via .object method via proxy or via security.flowers api for pollen.computer services like composer.computer and bug tracking

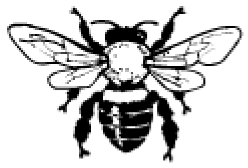
pollen.help api: A helper api. Including reference documents on a top level and helper proxy functions and services such as polyfills and patches, and can observe component crossover properties such as data migrations or asynchronous operations requiring a sophisticated proxy

```
pollen.computer.stage.email/{{here}}.pollen.help(composer.computer.stage.exchange)
intension.cloud → composer.computer
internsion.computer → composer.computer
```

(so in this example, above, I staged an email in a location and used pollen.help API to sync reference objects. Then added a backup configuration API and a bug tracker API)

```
intension.press.pollen.computer → intension.agency.pollen.computer.composer.computer
```

(And then extract a report or report those methods to utility api via a watch api.)



§ DOM Structure:

Pollen.computer uses .xml
“XML is a software- and hardware-independent tool for storing and transporting data.”
XML, the eXtensible Markup Language, is not HTML! (XML is self-descriptive)
XML is recommended by the W3C (quoted above)

The DOM structure includes this .xml reference material to primary APIs, such as JavaScript, the Browser, and the DOM.

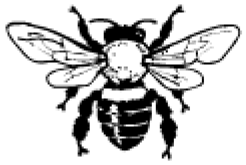
Where possible pollen.computer will simplify by concatinating into a single reference entitity appendable to any DOM as a JavaScript file which will extend into the necessary structure by appending static or dynamic resources like stylesheets and additional .js or .xml resources
Creating a regular pattern of:
pollen.computer.pollen.js
(appends)
→ pollen.css
→ pollen.xml

§ Glossary:

.value
.press value (a method ususall of higher level api access or as their assigned variable or value
.press value service

vermont.computer (look for vermont. methods in 2018 such as vermont.town methods for utility services like vermont.computer.vermont.town.brattleboro.town (an attribiture location prefix method we pre defined utility sevices))
person.computer
candy.computer
intension.computer ---> a bug tracker and QA api
intension.agency ---> watch api
Intelligence.computer ---> utility api
Intellegence.computer ---> deep intelligence.computer api analogue (unit testing)
speak.computer

2016-12-3 computer API protocols

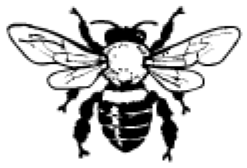


number.computer
artifact.computer

composer.computer utility api and service of security.flowers
beethoven.computer
bach.computer
mozart.computer

Science.codes interface of objects:
Mutation.codes : watcher observer ser of methods for .tecnology value
Svg.codes : an object API for entity, semantic, and programatic reference are to be determoined by W3C, SVG authorities, and this project
Tilde.press : an object literal of the tilde method as location, also a .press value service and a tilde variable
tilde.host
tilde.cloud
pollen.agency
pollen.zone
pollen.cafe
pollen.ltda
pollen.blue
pollen.pink
pollen.healthcare
pollen.help
pollen.host
pollen.solutions
pollen.tips
intension.agency
intension.cloud → a backup configuration API activating composer.computer at a low and secured level (and is an open souce api and library or service object at the top level
intension.press
respolyn.xyz
respolyn.com
rhamses.xyz
polytec.xyz
stage.email
stage.exchange
stage.healthcare
stage.legal
stage.lgbt

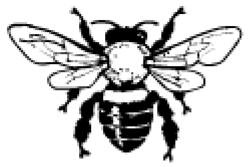
2016-12-3 computer API protocols



stage.zone
stage.cafe
stage.ltda
union.codes
union.institute

artifact.
beethoven.
bach.
composer.
intension. (4)
intelligence.
mozart.
mutation.
number.
pollen. (12)
respolyn.
rhamses.
speak.
stage. (8)
security.
science.
svg.
tilde. (5)
union.
vermont. (3)

.agency (3)
.blue
.cafe (2)
.cloud (2)
.computer (11)
.codes (4)
.email
.estate
.exchange
.flowers
.healthcare (2)
.host (2)
.help



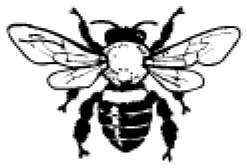
.ltda (2)
.lgbt
.party
.pink
.press (2)
.solutions
.tips
.town (to come potentially) (5)
.technology
.xyz (3)
.zone (2)

§ notes

Ok so you have this neat syntax but what can I do with it? How useful is it?

Like : what does the vermont. location do?
What do the science. Or speak. Locations do?

2016-12-3 computer API protocols



What is the function of the .codes parameter? Or the .lgbt parameter ? I don't know

But it may be important to keep some of these open, rather than excluding them from the foundation. (conservative in what you do and liberal in what you accept)
So let's keep the .pink .blue .party methods

Test case:
Say you had these objects:

.computer (X) ← most common method: most attributes
.codes (Z)
.exchange (S)
.host (K)
pollen. (Y) ← most common object: most attributes
stage. (D)
svg. (P)

And various other objects with various values representing sets of attributes
Knowing pollen. + .computer have the most attributes