



Services, Distributed Methods Across Top-Level Domain Entities

§ intro

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 range 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.email/{{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`)

`intension.computer` is the bug tracking API and `composer` is the build api for that unit, but there are other `intension`. 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.ltida 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.ltida 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),
```

§ 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

. Attribute . {{}} (# indicates frequency)	{{}}. compent.
.agency (3)	artifact.
.blue	beethoven.
.cafe (2)	bach.
.cloud (2)	composer.
.computer (11)	intension. (4)
.codes (4)	intelligence.
.email	mozart.
.estate	mutation.
.exchange	number.
.flowers	pollen. (12)
.healthcare (2)	respolyn.
.host (2)	rhamases.
.help	speak.
.ltda (2)	stage. (8)
.lgbt	security.
.party	science.
.pink	svg.
.press (2)	tilde. (5)
.solutions	union.
.tips	vermont. (3)
.town (to come potentially) (5)	
.technology	
.xyz (3)	



.zone (2)	
-----------	--

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



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

§ Definitions:

Pollen.computer.security.flowers [method]

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

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.intelligence.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 structure:

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

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

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 validation and service provider workign with permissions to run the bach. mozart. beethoven.computer protocols providing asynchrization services and data intergration (across the suite) , ststem backup and storage, and artificiaial matinance such as code repository 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

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. patterns 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 themselves, 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 security.flower

security.flower is an api that authenticates and secures token for services to and from pollen.computer exchanges, such as for database queries, unless the pollen.computer api is itself hosted as a proxy (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 source api composer.computer can become a proprietary token when authenticated via <https://security.flowers> an open source api, security.flower services are available to run or host locally (as a proxy) and are accessible 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
 object → <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)
 object.method.object/action → stage.email/killPermissions.exe (changeable syntax).
 {depending on the operator's set parameters} (chain 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 available .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

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

EX>

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.

§ Glossary:

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

number.computer

artifact.computer

composer.computer utility api and service of security.flowers

Beethoven.computer

bach.computer

mozart.computer

science.codes

mutation.codes

svg.codes

security.flowers

mutation.technology

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 → a backup configuration API activating composer.computer at a low and secured level (and is an open source 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
stage.zone
stage.cafe
stage.ltda
union.codes
union.institute

The MIT License (MIT)

Copyright (c) 2016 Jordan Lafland

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute,



sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.