Mike Hahn – hahnbytes@gmail.com

Jovelyzer

Jovelyst is a programming language used for developing desktop apps which run in your web browser. Whenever an end-user wishes to run a Jovelyst desktop app, she launches the Jovelyst web launcher (which in turn starts up the embedded, open source Jetty web server) and then points her web browser to localhost/appname, where "appname" is the name of the desired Jovelyst app. Jovelyst rhymes with novelist and is implemented in Java.

Business Model

Jovelyzer.com is a website which includes a hosting service (app center) for Jovelyst desktop apps which can be uploaded by developers and downloaded by end-users. Eventually this hosting service will also host web-based apps in the form of server-side Jovelyst code. Client-side Jovelyst code is converted to JavaScript when uploaded to the hosting service. Developers pay a subscription fee of $10/year for uploading privileges, and will eventually pay web hosting fees for their web-based apps (sample URL: appname.jvlst.com) which include server-side Jovelyst code. End-users who wish to share any of their Jovelyst-related data using Google Drive must also pay a subscription fee of $10/year.

Joppineez

Joppineez is both a tool which lets you organize your image folders, and the first sample Jovelyst app. Joppineez 2.0 supports image sharing functionality, using Google Drive. The Java version of Joppineez will be implemented even before Jovelyst is implemented.

Lystagger Syntax

Screen layout definition language. Doubles as HTML replacement for Jovelyst web development framework.

Tags:

- (tag)
- (tag; body)
- (tag (fld val)*; body)

Body:

- text
- (; text)*
- ((fld val)*; text)*

Notation:

- * repeats
- a; b; c = (a)(b)(c)

Android Keyboard

1	2	3	4	5	6	7	8	9	0
Q	W	E	R	T	Y	U	I	O	P
A	S	D	F	G	H	J	K	L	;
SH	Z	X	C	V	B	N	M	&	BK
SP	(	)	-	"	\	#	:	FN	CR

SH - Shift
SP - Space
BK - Backspace
CR - Enter
FN - Functions
& - Symbols

Note: Android is not currently targeted as a Jovelyst platform, but will be eventually.

Search Engine

Jovelyzer features a built-in search engine which indexes all web-based apps hosted with Jovelyzer. The user can enter a word or phrase to search for, or perform Advanced Search, specifying various search criteria. The user can also specify ranking of search results, such as: relevance, popularity, alphabetical order, price.

The search engine can display search-based ads on the right-hand side of the browser window based on keywords typed in by the user. Web pages can display content-based ads (not just banner ads) at the location(s) determined by the web developer. Content-based ads are displayed when certain keywords chosen by the advertiser appear on the same web page. Ad revenue is split between Jovelyzer and the owner of the web-based app (the client): 30 percent to Jovelyzer and 70 percent to the client.

Billing

Clients of Jovelyzer web hosting, whose web-based apps which usually include Jovelyst server-side code are uploaded to Jovelyzer.com, are billed monthly for each of their web-based apps (or "apps" for short). Each app falls into one of 6 tiers, depending on how heavily it consumes resources. The bottom (first) tier includes all apps in the bottom 20 percent. The second tier includes all apps above the first tier but below the median. The third tier includes all apps above the median but below the top 30 percent. The fourth tier includes all apps below the top 10 percent but above the bottom 70 percent. The fifth tier includes the top 10 percent of all apps. The sixth tier includes only those apps having an unusually high consumption of resources. The pricing per month for all 6 tiers is as follows:

1. $10
2. $15
3. $25
4. $40
5. $60
6. $60 + high-usage fee

The high-usage fee is proportional to the resources consumed in excess of the 95th percentile, and is generally capped at $240. In extreme cases it can go as high as $440. The pricing per month for a given app cannot jump by more than 2 tiers, above or below the previous month, unless the new pricing is in the sixth tier. Apps displaying a banner ad at the top of the client area of the user's web browser benefit by dropping 2 tiers: a tier-5 app drops down to tier-3, a tier-3 app drops down to tier-1, and tier-2 apps and below drop down to tier-zero, which is free for a given month. Tier-6 apps displaying a banner ad get a discount of $35, but don't drop down 2 tiers.

Resource Calculation

Resources consumed per month, per app, are based on the length of a vector (X, Y), where X is the no. of nodes in RAM created, and Y is the product of a constant K and the no. of words of graphics data uploaded/downloaded. (A node is generally 10 bytes and a word is 4 bytes.) The value of K is chosen so that $X_{AVG} = KY_{AVG}$ (the average values are calculated over all apps hosted by Jovelyzer). The value X is a rough approximation of how CPU-intensive a given app is in any given month. Resource consumption per app is also measured on a daily basis. The 4 highest daily resource consumption values in a given month are ignored for the purposes of doing the monthly resource consumption calculation, for a given app.

Classes of Users

Users of Jovelyzer web-based apps fall into 4 categories: unranked, bronze, silver, and gold. Unranked users do not possess memberships of any apps, and do not pay a subscription fee. Many apps make use of a freemium business model, where most users pay no fees but power users elect to pay an annual subscription fee. Bronze users are members of a single app. Silver users are members of up to 4 apps. Gold users are members of an unlimited no. of apps. The annual subscription fees for the 3 ranked user classes are as follows:

1. Bronze: $10
2. Silver: $25
3. Gold: $50

Every client of Jovelyzer web hosting who has at least one freemium app hosted with at least one member on any given day receives a micro-share of the total subscription fees collected by Jovelyzer. That micro-share is proportional to the pricing of the web hosting for that app in the current month. So a tier-5 app would receive a micro-share 6 times higher than a tier-1 app. Every month the micro-shares are added up for all daily users and all apps belonging to a given client, who receives a payment offsetting the web hosting prices paid to Jovelyzer. Those payments are split between Jovelyzer and each client: 30 percent to Jovelyzer and 70 percent to the client.

Business Plan Outline

1. Jovelyst SDK:
    1. Implement Jovelyst Compiler
    2. Integrate Lystagger with Jovelyst
    3. Learn Java for Web Applications coding
    4. Implement Joppineez app
    5. Apply to join the Ryerson DMZ tech incubator
    6. If successful then:
        1. Hire 2 recent com-sci Ryerson grads
        2. Work takes place at DMZ downtown
    7. Else:
        1. Hire 2 autistic programmers using Specialisterne
        2. They find IT jobs for those on the autism spectrum
        3. Both of them work in my condo
    8. Write business plan
    9. Seek angel investor
    10. Junior Programmer:
        1. Implement Converter: Lystagger-to-HTML
        2. Implement Converter: Jovelyst-to-JavaScript
    11. Senior Programmer:
        1. Implement Jovelyst IDE
        2. Implement Lystagger IDE
        3. Implement WYSIWYG (Lystagger)
    12. Support data-sharing using Google Drive
    13. Expand Jovelyst Library
    14. License the Jovelyst SDK to web hosting firms and others
2. Website:
    1. Hire web programmer if angel investor found
    2. Design app center website
    3. Handle credit cards/PayPal
    4. Handle sales tax
    5. If angel investor not found:
        1. Design website in-house
        2. Outsource production version
    6. Implement web-based app hosting post-launch
    7. Handle billing of web-based apps
    8. Handle disbursement of subscription fees
    9. Implement search engine:
        1. Basic search
        2. Advanced search
        3. Search-based ads
    10. Sell banner ads to advertisers
    11. Sell content-based ads to advertisers
3. Launch:
    1. Hire CEO if angel investor found
    2. Else:
        1. Mike is CEO
        2. Programmers laid off after 6 to 12 months
        3. Jovelyzer becomes open source
    3. Purchase Google AdWords ads
    4. Jovelyzer.com goes live
    5. Consider using Web Design Pros
    6. Use SEO to increase traffic

4. Android:
    1. Convert Joppineez to Java
    2. Implement Joppineez Android app
    3. Develop Jovelyst SDK for Android

History

I am Mike, the founder, and got the idea for Jovelyst on Canada's 150th birthday. I was at my brother's cottage and used the memo feature of my phone to make a to do list. (Prior to April 2017 I had never owned a cell phone.) Realizing that a phone can actually do more than just call/text, I then got the idea of using the programming language I had previously designed and begun implementing, as it was ideal for a small screen. I then resurrected my old games program idea (make your own games), just for Android.

About Me

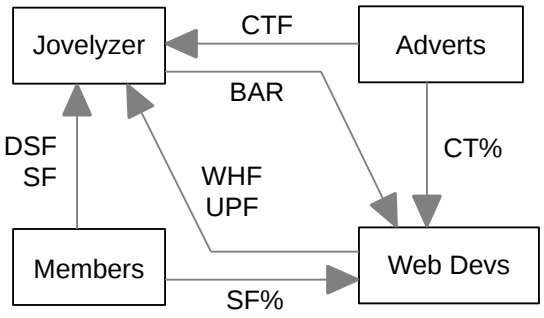
I am Mike Hahn, the founder of Jovelyzer.com. I was previously employed at Brooklyn Computer Systems as a Delphi Programmer and a Technical Writer (I worked there between 1996 and 2013). At the end of 2014 I quit my job as a volunteer tutor at Fred Victor on Tuesday afternoons, where for 5 years I taught math, computers, and literacy. I'm now a volunteer math/computer tutor at West Neighbourhood House. My hobbies are reading quora.com questions/answers and the news at cbc.ca. About twice a year I get together with my sister Cathy who lives in Victoria. She comes here or I go out there usually in the summer. Prior to starting Jovelyst I used to lie on the couch a lot, not being very active. Now I'm busy most of the time. I visit my brother Dave once a month or so and I also visit my friends Main and Steph once or twice a month.

Contact Info

Mike Hahn
Founder, Jovelyzer.com
515-2495 Dundas St. West
Toronto, ON M6P 1X4

Country: Canada
Phone: 416-533-4417
Email: hahnbytes (AT) gmail (DOT) com
Web: www.hahnbytes.com

Cash Flow Diagram



Payments Table

Abbr.	Fees	Amounts
WHF	Web Hosting	$10 - 60/month
SF	Subscriptions	$10 - 50/year
SF%	Subscriptions	70%
UPF	Uploading Privileges	$10/year
DSF	Data Sharing	$10/year
BAR	Banner Ad Rebate	$10 - 35/month
CTF	Click Thru	varies
CT%	Click Thru	70%

Language Grammar

Jovelyst is a Python dialect in which all operators precede their operands, and parentheses are used for all grouping (except string literals, which are delimited with double quotes). Jovelyst code is often accompanied by Lystagger screen definition files. Lystagger is similar to HTML, except open tags begin with an open parenthesis and a keyword, and the closing tag is simply a close parenthesis. Any text enclosed in a tag is preceded by a semicolon. File extensions include .LYST (source code), .LSTA (assembler code), .LSTC (compiled code), and .LSTG (Lystagger).

Keyboard Aid

This optional feature enables hyphens, open parentheses, and close parentheses to be entered by typing semicolons, commas, and periods, respectively. When enabled, keyboard aid can be temporarily suppressed by using the Ctrl key in conjunction with typing semicolons, commas, and periods (no character substitution takes place). By convention, hyphens are used to separate words in multi-word identifiers, but semicolons are easier to type than hyphens. Similarly, commas and periods are easier to type than parentheses. Typing semicolon converts previous hyphen to a semicolon, and previous semicolon to a hyphen (use the Ctrl key to override this behaviour). Typing semicolon after close parenthesis simply inserts semicolon.

Special Characters

- `()` grouping
- `-` used in identifiers
- `;` end of stmt.
- `:` dot operator
- `"` string delimiter
- `\` escape char.
- `#` comment

More Characters

- `_` used in identifiers
- `$` string prefix char.
- `*` public switch
- `{}` block comment

Differences from Python

- Parentheses, not whitespace
- Integration with Lystagger
- Operators come before their operands
- Information hiding (public/private)
- Single, not multiple inheritance
- Adds interfaces ("scool" defs.)
- Drops iterators and generators
- Adds lambdas
- Adds quote and list-compile functions, treating code as data

Grammar Notation

- Non-terminal symbol: <symbol name>
- Optional text in brackets: [*text*]
- Repeats zero or more times: [*text*]…
- Repeats one or more times: <symbol name>…
- Pipe separates alternatives: *opt1* | *opt2*
- Comments in *italics*
- Advanced features flagged as **

Compiler and Assembler

The Jovelyst Compiler translates source code into compiled code, and optionally into assembler code. Assembler code is an intermediate language which is much simpler than source code, although both source code and assembler code are in the form of text files. During Jovelyst development, the developer uses the Jovelyst Assembler, which converts assembler code (hand-written by the developer) into compiled code. This is a necessary step enabling the JovelYst Runtime Environment (JYRE) to be tested prior to the development of the Jovelyst Compiler.

Jovelyst Grammar

White space occurs between tokens (parentheses and semicolons need no adjacent white space, also any semicolon before a close parenthesis may be omitted):

<source file>:
- [<use>] [* <vars>] [<vars>] [do <block>] [<def>]… [<class>]… [do <block>]

<use>:
    use (<import-semi>...)

<import-semi>:
    <import-stmt> **;**

<import stmt>:
    import <module>
    import (<module>…)
    from <rel module> import <mod list>
    from <rel module> import all

<module>:
    <name>
    (<name> as <name>)
    (**:** <name>… [as <name>])

<mod list>:
    <id as>
    (<id as>…)

<id as>:
    <mod id>
    (<mod id> as <name>)

<mod id>:
    <mod name>
    <class name>
    <func name>
    <var name>

<rel module>:
    (**:** [<num>] [<name>]...)
    <name> *// ?*

<class>:
- ([*] <cls typ><name> [<base class>] [<does>] [* <vars>] [<vars>] <def>…)
- ([*] scool <name> [<does>] [<const list>] [<def hdr>]...)
- ([*] enum <name><elist>)

<cls typ>:
    class
    abclass

<does>:
    does (<scool name>...)

<scool name>:
<base class>:
    <name>
    (**:** <name><name>…)

<const list>:
    const (<const pair>...)

<const pair>:
    (<name><const expr>)

<def hdr>:
    (<defun><name> ([<parms>]) [<dec>])

<def>:
- ([*] <defun><name> ([<parms>]) [<vars>] [<dec>] do <block>)

<defun>:
    def
    abdef

<vars>:
    var (<id>...)

<parms>:
    <parm>... [(* <id>)] [(** <id>)]

<parm>:
    <id>
    (tuple <id>...)
    (<set op><id><expr>)
    (<set op> (tuple <id>...) <expr>)

<dec>:
    decor <call expr>...

<block>:
    ([<stmt-semi>]…)

<stmt-semi>:
    <stmt> **;**

<jump stmt>:
    <continue stmt>
    <break stmt>
    <return stmt>

<pjump stmt>:
    return <expr>
    ** <raise stmt>

<raise stmt>:
    raise [<expr> [from <expr>]]

<stmt>:
    <open stmt>
    <closed stmt>

<open stmt>:
    <if stmt>
    <while stmt>
    <for stmt>
    ** <try stmt>
    <pjump stmt>
    <pcall stmt>
    <asst stmt>
    <del stmt>

<closed stmt>:
    <jump stmt>
    <call stmt>
    <print stmt>
    <lstg tag>

<lstg tag>:
    (lstg **;** ...)

<call expr>:
- (<name> [<arg list>])
- (**:** <obj expr> [<colon expr>]…
  (<method name> [<arg list>]))
- (call <expr> [<arg list>])

<call stmt>:
    (<name> [<arg list>])

<pcall stmt>:
- **:** <obj expr> [<colon expr>]…
  (<method name> [<arg list>])
- call <expr> [<arg list>]

<colon expr>:
    <name>
    (<name> [<arg list>])

<arg list>:
    [<expr>]... [(<set op><id><expr>)]...

<asst stmt>:
    <asst op><target expr><expr>
    <set op> (tuple <target expr>...) <expr>

<asst op>:
    set | addset | minusset | mpyset | divset |
    idivset | modset |
    shlset | shrset | shruset |
    andbset | xorbset | orbset |
    andset | xorset | orset |
    = | += | -= | *= | /= |
    //= | %= |
    <<= | >>= | >>>= |
    &= | ^= | '|=' |
    &&= | ^^= | '||='

<set op>:
    set | =

<target expr>:
    <name>
    (**:** <name> [<colon expr>]… <name>)
    (slice <arr><expr> [<expr>])
    (slice <arr><expr> all)

<arr>: *// string or array/lyst*
    <name>
    <expr>

<obj expr>:
    <name>
    <call stmt>

<if stmt>:
- if <expr> do <block> [elif <expr> do <block>]… [
  else <block>]

<while stmt>:
    while <expr> do <block>
    do <block> while <expr>

<for stmt>:
    for <name> in <expr> do <block>

<try stmt>:
- try <block> <except clause>… [else <block>]
  [finally <block>]
- try <block> finally <block>

<except clause>:
    except <name> [as <name>] do <block>

<return stmt>:
    return

<break stmt>:
    break

<continue stmt>:
    continue

<del stmt>:
    del <expr>

<paren stmt>:
    (<open stmt>)
    <closed stmt>

<qblock>:
    (quote [<paren stmt>]...)

<expr>:
    <keyword const>
    <literal>
    <name>
    (<unary op><expr>)
    (<bin op><expr><expr>)
    (<multi op><expr><expr>…)
    (<quest><expr><expr><expr>)
    <lambda>
    (quote <expr>...)
    <renum expr>
    <tuple expr>
    <lyst expr>
    <dict expr>
    <bitarray expr>
    <string expr>
    <bytezero expr>
    <bytes expr>
    <target expr>
    <obj expr>
    <cast>

<quest>:
    quest | ?

<unary op>:
    minus | notbitz | not |
    - | ~ | !

<bin op>:
    <arith op>
    <comparison op>
    <shift op>
    <bitwise op>
    <boolean op>

<arith op>:
    div | idiv | mod | mpy | add | minus |
    / | // | % | * | + | -

<comparison op>:
    ge | le | gt | lt | eq | ne | is | in |
    >= | <= | > | < | == | !=

<shift op>:
    shl | shr | shru |
    << | >> | >>>

Note: some operators delimited with single quotes for clarity (quotes omitted in source code)

<bitwise op>:
    andbitz | xorbitz | orbitz |
    & | ^ | '|'

<boolean op>:
    and | xor | or |
    && | ^^ | '||'

<multi op>:
    mpy | add | strdo | strcat |
    and | xor | andbitz | xorbitz |
    or | orbitz |
    * | + | % | + |
    && | ^^ | & | ^ |
    '||' | '|'

<const expr>:
    <literal>
    <keyword const>

<literal>:
    <num lit>
    <str lit>
    <bytes lit>

<tuple expr>:
    (tuple <expr>…)

<lyst expr>:
    (lyst [<expr>]…)

<dict expr>:
    (dict [<pair>]…)

<bitarray expr>:
    (bitarray <enum name> [<elist>])
    (bitarray <enum name><idpair>...)

<elist>:
    <id>...
    <intpair>...
    <chpair>...

<intpair>
    // integer constant
    <int const>
    (<int const><int const>)

<chpair>
    // one-char. string
    <char lit>
    (<char lit><char lit>)

<idpair>
    <idt>
    (<id><id>)

<pair>:
    // expr1 is a string
    (<expr1><expr2>)
    (<str lit><expr>)

<renum expr>
    (renumize <expr><ren id>...)
    (renumize <expr><ren int>...)
    (renumize <expr><ren ch>...)

<ren id>:
    (0 <id>)
    (1 <id>)
    (1 <id><id>)

<ren int>:
<ren ch>:
    // expr is <dec int> | <char lit>
    (0 <expr>)
    (0 <expr><expr>)
    (1 <expr>)
    (1 <expr><expr>)

<cast>:
    (cast <type><expr>)

<print stmt>:   *// built-in func*
    (print [<expr>]…)
    (echo [<expr>]…)

<lambda>:
    (lambda ([<id>]...) <expr>)
    (lambda ([<id>]...) do <block>)
    (lambda ([<id>]...) do <qblock>)
    // must pass qblock thru compile func

No white space allowed between tokens, for rest of Jovelyst Grammar

<white space>:
    <white token>...

<white token>:
    <white char>
    <line-comment>
    <blk-comment>

<line-comment>:
    # [<char>]... <new-line>

<blk-comment>:
    { [<char>]... }

<white char>:
    <space> | <tab> | <new-line>

<name>:
- [<underscore>]… <letter> [<alnum>]…
  [<hyphen-alnum>]… [<underscore>]…

<hyphen-alnum>:
    <hyphen><alnum>…

<alnum>:
    <letter>
    <digit>

In plain English, names begin and end with zero or more underscores. In between is a letter followed by zero or more alphanumeric characters. Names may also contain hyphens, where each hyphen is preceded and succeeded by an alphanumeric character.

<num lit>:
    <dec int>
    <long int>
    <oct int>
    <hex int>
    <bin int>
    <float>

<dec int>:
    [<hyphen>] 0
    [<hyphen>] <any digit except 0> [<digit>]…

<long int>:
    <dec int> L

<float>:
    <dec int><fraction> [<exponent>]
    <dec int><exponent>

<fraction>:
    <dot> [<digit>]…

<exponent>:
    <e> [<sign>] <digit>…
<e>:
    e | E

<sign>:
    + | -

<keyword const>:
    none
    true
    false

<oct int>:
    0o <octal digit>…

<hex int>:
    0x <hex digit>…
    0X <hex digit>…

<bin int>:
    0b <zero or one>…
    0B <zero or one>…

<octal digit>:
    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

<hex digit>:
    <digit>
    A | B | C | D | E | F
    a | b | c | d | e | f

<string lit>:
    [$ <str prefix>] <short long>

<str prefix>:
    r | u | R | U

<short long>:
    " [<short item>]... "
    " " " [<long item>]... " " "

<short item>:
    <short char>
    <escaped str char>

<long item>:
    <long char>
    <escaped str char>

<short char>:
    any source char. except "\", newline, or
    end quote

<long char>:
    any source char. except "\"

<bytes lit>:
    $ <byte prefix><shortb longb>

<byte prefix>: *// any case/order*
    b | br

<shortb longb>:
    " [<shortb item>]... "
    " " " [<longb item>]... " " "

<shortb item>:
    <shortb char>
    <escaped char>

<longb item>:
    <longb char>
    <escaped char>

<shortb char>:
    any ASCII char. except "\", newline, or
    end quote

<longb char>:
    any ASCII char. except "\"

<escaped char>:
    \newline
        ignore "\", newline chars.
    \\ *backslash*
    \" *double quote*
    \} *close brace*
    \a *bell*
    \b *backspace*
    \f *formfeed*
    \n *new line*
    \r *carriage return*
    \t *tab*
    \v *vertical tab*
    \ooo *octal value = ooo*
    \xhh *hex value = hh*

<escaped str char>:
    <escaped char>
    \N{name} *Unicode char. = name*
    \uxxxx *hex value (16-bit) = xxxx*