Mightier Game Development

Amazing gameplay, level design & AI without melting your brain

Language Specification

ver. 3.0 (beta 2)

[Live/current version at http://SkookumScript.com/docs/]

Conan Reis
conan@AgogLabs.com

October 16, 2014

Agog! Labs
Fun games lovingly crafted by mad scientists

Combined syntactical and lexical rules for SkookumScript in modified Extended Backus-Naur Form (EBNF). Production rules: *in-italics*. Terminals: **`coloured and in bold`**. Literal strings: '**`quoted`**'. Optional groups: []. Repeating groups of zero or more: {}. Repeating groups of n or more: {}$^{n+}$. Mandatory groups: (). Alternatives: |.

File Names and Bodies:

method-filename[1] = *method-name* '**`()`**' ['**`C`**'] '**`.skoo`**'
method-file[2] = *ws parameters* [*ws code-block*] *ws*

coroutine-filename = *coroutine-name* '**`()`**' '**`.skoo`**'
coroutine-file[3] = *ws parameter-list* [*ws code-block*] *ws*

data-filename[4] = '**`!Data`**' ['**`C`**'] '**`.skoo`**'
data-file = *ws* [*data-definition* {*wsr data-definition*} *ws*]
data-definition[5] = [*class-desc wsr*] '**`!`**' *data-name*

object-id-filename[6] = *class-name* ['**`-`**' {**`printable`**}] '**`.skoo`**' '**`-`**' | '**`~`**' '**`ids`**'
object-id-file[7] = {*ws symbol-literal* | *raw-object-id*} *ws*
raw-object-id[8] = {**`printable`**}$^{1\text{-}255}$ *end-of-line*

Expressions:

expression = *literal* | *identifier* | *flow-control* | *primitive* | *invocation*

Literals:

literal = *boolean-literal* | *integer-literal* | *real-literal* | *range-literal* | *string-literal* | *symbol-literal* | *char-literal* | *list-literal* | *closure*
boolean-literal = '**`true`**' | '**`false`**'
integer-literal[9] = ['**`-`**'] *digits-lead* ['**`r`**' *big-digit* {['**`_`**'] *big-digit*}]
real-literal[10] = ['**`-`**'] *digits-lead* V ('**`.`**' *digits-tail*) [*real-exponent*]
real-exponent = '**`E`**' | '**`e`**' ['**`-`**'] *digits-lead*
digits-lead = '**`0`**' | (*non-zero-digit* {['**`_`**'] *digit*})
digits-tail = *digit* {['**`_`**'] *digit*})
range-literal = [*expression*] '**`..`**' [*expression*] | ('**`#`**' *expression*)
string-literal = *simple-string* {*ws* '**`+`**' ws *simple-string*}
simple-string = '**`"`**' {*character*} '**`"`**'
symbol-literal = '**`'`**' {*character*}$^{0\text{-}255}$ '**`'`**'
char-literal = '**`` ` ``**' *character*
list-literal[11] = [*list-class constructor-name invocation-args*]
'**`{`**' *ws* [*expression* {*ws* ['**`,`**' *ws*] *expression*} *ws*] '**`}`**'
closure[12] = ['**`^`**' ['**`_`**' *ws*] [*expression ws*]] [*parameters ws*] *code-block*

---

[1] If optional '**`?`**' is used in query/predicate method name, use '**`-Q`**' as a substitute since question mark not valid in filename.

[2] Only immediate calls are permissible in the code block. If *code-block* is absent, it is defined in C++.

[3] If *code-block* is absent, it is defined in C++.

[4] A file name appended with '**`C`**' indicates that the file describes class members rather than instance members.

[5] *class-desc* is compiler hint for expected type of member variable. If class omitted, **`Object`** inferred or **Boolean** if *data-name* ends with '**`?`**'. If *data-name* ends with '**`?`**' and *class-desc* is specified it must be **Boolean**.

[6] Starts with the object id class name then optional source/origin tag (assuming a valid file title) – for example: Trigger-WorldEditor, Trigger-JoeDeveloper, Trigger-Extra, Trigger-Working, etc. A dash '**`-`**' in the file extension indicates an id file that is a compiler dependency and a tilde '**`~`**' in the file extension indicates that is not a compiler dependency

[7] Note: if *symbol-literal* used for id then leading whitespace, escape characters and empty symbol can be used.

[8] Must have at least 1 character and may not have leading whitespace (*ws*), single quote ('**`'`**') nor *end-of-line* character.

[9] '**`r`**' indicates *digits-lead* is (r)adix/base from 1 to 36 – default 10 (decimal) if omitted. Ex: **`2r`** binary & **`16r`** hex. Valid *big-digit*(s) vary by the radix used. See *math-operator* footnote on how to differentiate subtract from negative *integer-literal*.

[10] Can use just *digits-lead* if **`Real`** type can be inferred from context otherwise the *digits-tail* fractional or *real-exponent* part is needed. See *math-operator* footnote on how to differentiate subtract from negative *real-literal*.

[11] **`List`** type specified if the optional *list-class* and appropriate *constructor-name* is supplied (also see *instantiation*). If the type is not supplied, then a type is inferred using the initial items – if there are no initial items then **`Object`** used as default item type).

[12] [Also known as: code block, anonymous function or lambda expression.] Optional '**`^`**' or *parameters* or both must be provided (unless used in *closure-tail-args* where both optional). Optional *expression* (may not be *code-block* or *closure*) captured and used

Identifiers:

identifier[1] = *variable-identifier* | *reserved-identifier* | *class-name* | *object-id*
variable-identifier[2] = *variable-name* | ([*expression ws* '**.**' *ws*] *data-name*)
instance-name = *lowercase* {*alphanumeric*}
variable-name = *name-predicate*
data-name[3] = '**@**' | '**@@**' *variable-name*
reserved-identifier = '**nil**' | '**this**' | '**this_class**' | '**this_code**'
class-name = *uppercase* {*alphanumeric*}
object-id[4] = [*class-name*] '**@**' ['**?**' | '**=**'] *symbol-literal*
routine-name = *method-name* | *coroutine-name*
method-name[5] = *name-predicate* | *constructor-name* | *destructor-name* | *class-name*
name-predicate[6] = *instance-name* ['**?**']
constructor-name = '**!**' [*instance-name*]
destructor-name[7] = '**!!**'
coroutine-name = '**_**' *instance-name*

Flow Control:

flow-control = *code-block* | *loop* | *loop-exit* | *conditional* | *case* | *when* | *unless* | *concurrent-block*
code-block = '**[**' *ws* [*expression* {*wsr expression*} *ws*] '**]**'
loop[8] = '**loop**' [*ws instance-name*] *ws code-block*
loop-exit[9] = '**exit**' [*ws instance-name*]
conditional = '**if**' {*ws expression ws code-block*}$^{1+}$ [*ws else-block*]
case = '**case**' *ws expression* {*ws expression ws code-block*}$^{1+}$ [*ws else-block*]
else-block = '**else**' *ws code-block*
when = *expression ws* '**when**' *ws expression*
unless = *expression ws* '**unless**' *ws expression*
concurrent-block = *sync* | *race* | *branch* | *divert*
sync[10] = '**sync**' *ws code-block*
race[11] = '**race**' *ws code-block*
branch[12] = '**branch**' *ws code-block*
divert[13] = '**divert**' *ws code-block*

---

as receiver/this for *code-block* – if omitted **this** inferred. Optional '**_**' indicates it is durational (like coroutine) – if not present durational/immediate inferred via *code-block*. Parameter types, return type, scope, whether surrounding **this** or temporary/parameter variables are used and captured may all be inferred if omitted.

[1] Scoping not necessary – instance names may not be overridden and classes and implicit identifiers effectively have global scope.

[2] Optional *expression* can be used to access data member from an object – if omitted, **this** is inferred.

[3] '**@**' indicates instance data member and '**@@**' indicates class instance data member.

[4] If *class-name* absent **Actor** inferred. Optional '**?**' indicates result may be **nil** – if question mark not used and object not found at runtime then assertion error occurs. Optional equals sign '**=**' indicates it is a symbol literal validated by class type.

[5] A method using *class-name* allows explicit conversion similar to *class-conversion* except that the method is always called.

[6] Optional '**?**' used as convention to indicate predicate variable or method of return type **Boolean** (**true** or **false**).

[7] Destructor calls are only valid in the scope of another destructor's *code-block*.

[8] The optional *instance-name* identifies the loop for specific reference by a *loop-exit* which is useful for nested loops.

[9] A *loop-exit* is valid only in the code block scope of the loop that it references.

[10] 2+ durational expressions run concurrently and next *expression* executed when *all* expressions returned (result **nil**, return args bound in order of expression completion).

[11] 2+ durational expressions run concurrently and next *expression* executed when *fastest* expression returns (result **nil**, return args of fastest expression bound) and other expressions are *aborted*.

[12] 1+ durational expressions run concurrently and diverted (updated by receiver object rather than current updater – see *divert*) and the next expression executed immediately (result **nil**).

[13] Durational expressions to be updated by receiver objects rather than by the calling updater object.

Primitives:

primitive = *create-temporary* | *bind* | *class-cast* | *class-conversion*
create-temporary = '**!**' *ws variable-name* [*ws binding*]
bind = *variable-identifier ws binding*
binding = '**:**' *ws expression*
class-cast[1] = *expression ws* '**<>**' [*class-desc*]
class-conversion[2] = *expression ws* '**>>**' [*class-name*]

Invocations:

invocation = *invoke-call* | *invoke-cascade* | *apply-operator* | *invoke-operator* | *instantiation* | *index-operator* | *slice-operator*
invoke-call[3] = ([*expression ws* '**.**' *ws*] *invoke-selector*) | *operator-call*
invoke-cascade = *expression ws* '**.**' *ws* '**[**' {*ws invoke-selector* | *operator-selector*}$^{2+}$ *ws* '**]**'
apply-operator[4] = *expression ws* '**%**' | '**%>**' *invoke-selector*
invoke-operator[5] = *expression bracketed-args*
index-operator[6] = *expression* '**{**' *ws expression ws* '**}**' [*ws binding*]
slice-operator[7] = *expression* '**{**' *ws range-literal* [*wsr expression*] ws '**}**'
instantiation[8] = *class-instance* | *expression* '**!**' [*instance-name*] *invocation-args*
invoke-selector = [*scope*] *routine-name invocation-args*
scope = *class-name* '**@**'
operator-call[9] = ('**not**' *ws expression*) | (*expression ws operator-selector*)
operator-selector = *postfix-operator* | (*binary-operator ws expression*)
binary-operator = *math-operator* | *compare-op* | *logical-operator* | '**:=**'
math-operator = '**+**' | '**+=**' | '**-**' | '**-=**' | '*****' | '***=**' | '**/**' | '**/=**'
compare-op = '**=**' | '**~=**' | '**>**' | '**>=**' | '**<**' | '**<=**'
logical-operator = '**and**' | '**or**' | '**xor**' | '**nand**' | '**nor**' | '**nxor**'
postfix-operator = '**++**' | '**--**'
invocation-args[10] = [*bracketed-args*] | *closure-tail-args*
bracketed-args = '**(**' *ws* [*send-args ws*] ['**;**' *ws return-args ws*] '**)**'
closure-tail-args[11] = *ws send-args ws closure* [*ws* '**;**' *ws return-args*]
send-args = [*argument*] {*ws* ['**,**' *ws*] [*argument*]}
return-args = [*return-arg*] {*ws* ['**,**' *ws*] [*return-arg*]}
argument = [*named-spec ws*] *expression*
return-arg = [*named-spec ws*] *variable-identifier*
named-spec[12] = *variable-name* '**#**'

---

[1] Compiler *hint* that expression evaluates to specified class – otherwise error. *class-desc* optional if desired type can be inferred. If *expression* is *variable-identifier* then parser updates type context. [Debug: runtime ensures class specified is received.]

[2] Explicit conversion to specified class. *class-name* optional if desired type inferable. Ex: **42>>String** calls convert method **Integer@String()** i.e. **42.String()** - whereas **"hello">>String** generates no extra code and is equivalent to **"hello"**.

[3] If an *invoke-call*'s optional *expression* (the receiver) is omitted, '**this.**' is implicitly inferred.

[4] If **List**, each item sent call – coroutines called using **%** – **sync**, **%>** – **race** concurrency depending on delimiter. If receiver empty list or **nil** invocation is skipped. If not list or **nil** executes like normal invoke call – i.e. '**%**' is synonymous to '**.**'.

[5] Akin to **expr.invoke(**…**)** or **expr._invoke(**…**)** depending if *expression* immediate or durational – *and* if enough context is available the arguments are compile-time type-checked plus adding any default arguments.

[6] Gets item (or sets item if *binding* present) at specified index [may be negative – see *slice-operator*].

[7] Returns sub-range: **{[first] (..[last]) | (# count) [wsr step]}**. Where: **last** is inclusive; **last** and **first** may be negative with -1 last item, -2 penultimate item, etc.; **step** may be negative indicating sub-range in reverse order.

[8] *expression* used rather than *class-instance* provides lots of syntactic sugar: **expr!ctor()** is alias for **ExprClass!ctor(expr)** – ex: **num!copy** equals **Integer!copy(num)**; brackets are optional for *invocation-args* if it can have just the first argument; a constructor-name of **!** is an alias for **!copy** – ex: **num!** equals **Integer!copy(num)**; and if **expr!ident** does not match a constructor it will try **ExprClass!copy(expr).ident** – ex: **str!uppercase** equals **String!copy(str).uppercase**.

[9] Every operator has a named equivalent. For example **:=** and **assign()**. Operators do *not* have special order of precedence – any order other than left to right must be indicated by using code block brackets (**[** and **]**).

[10] *bracketed-args* may be omitted if the invocation can have zero arguments.

[11] Routines with last send parameter as mandatory closure may omit brackets '**()**' and closure arguments may be simple *code-block* (omitting '**^**' and *parameters* and inferring from parameter). Default arguments indicated via comma '**,**' separators.

[12] Used at end of argument list and only followed by other named arguments. Use compatible **List** object for group argument. Named arguments evaluated in parameter index order regardless of call order since defaults may reference earlier parameters.

Parameters:

parameters[1] = *parameter-list* [*ws class-desc*]
parameter-list = '**(**' *ws* [*send-params ws*] ['**;**' ws return-params ws] '**)**'
send-params = *parameter* {*ws* ['**,**' *ws*] *parameter*}
return-params = *param-specifier* {*ws* ['**,**' *ws*] *param-specifier*}
parameter = *unary-param* | *group-param*
unary-param[2] = *param-specifier* [*ws binding*]
param-specifier[3] = [*class-desc wsr*] *variable-name*
group-param[4] = '**{**' *ws* [*class-desc* {*wsr class-desc*} *ws*] '**}**' *ws instance-name*

Class Descriptors:

class-desc = *class-unary* | *class-union*
class-unary = *class-instance* | *meta-class*
class-instance = *class-name* | *list-class* | *invoke-class*
meta-class = '**<**' *class-name* '**>**'
class-union[5] = '**<**' *class-unary* {'**|**' *class-unary*}[1+] '**>**'
invoke-class[6] = ['**_**' | '**+**'] *parameters*
list-class[7] = **List** '**{**' *ws* [*class-desc ws*] '**}**'

Whitespace:

wsr[8] = {*whitespace*}[1+]
ws = {*whitespace*}
whitespace = *whitespace-char* | *comment*
whitespace-char = ' ' | **formfeed** | **newline** | **carriage-return** | **horiz-tab** | **vert-tab**
end-of-line = **newline** | **carriage-return** | **end-of-file**
comment = *single-comment* | *multi-comment*
single-comment = '**//**' {**printable**} *end-of-line*
multi-comment = '**/***' {**printable**} [*multi-comment* {**printable**}] '***/**'

Characters and Digits:

character = *escape-sequence* | **printable**
escape-sequence[9] = '****' *integer-literal* | **printable**
alphanumeric = *alphabetic* | *digit* | '**_**'
alphabetic = *uppercase* | *lowercase*
lowercase = '**a**' | … | '**z**'
uppercase = '**A**' | … | '**Z**'
digits = '**0**' | (*non-zero-digit* {*digit*})
digit = '**0**' | *non-zero-digit*
non-zero-digit = '**1**' | '**2**' | '**3**' | '**4**' | '**5**' | '**6**' | '**7**' | '**8**' | '**9**'
big-digit = *digit* | *alphabetic*

---

[1] Optional *class-desc* is return class – if type not specified **Object** is inferred (or **Boolean** type for predicates or **Auto_** type for closures) for nested parameters / code blocks and **InvokedCoroutine** is inferred for coroutine parameters.

[2] The optional *binding* indicates the parameter has a default argument (i.e. supplied *expression*) when argument is omitted.

[3] If optional *class-desc* is omitted **Object** is inferred or **Auto_** for closures or **Boolean** if *variable-name* ends with '**?**'. If *variable-name* ends with '**?**' and *class-desc* is specified it must be **Boolean**.

[4] **Object** inferred if no classes specified. Class of resulting list bound to *instance-name* is class union of all classes specified.

[5] Indicates that the class is any one of the classes specified and which in particular is not known at compile time.

[6] '**_**' indicates durational (like coroutine), '**+**' indicates durational/immediate and lack of either indicates immediate (like method). Class '**Closure**' matches any closure interface. Identifiers and defaults used for parameterless closure arguments.

[7] **List** is any **List** derived class. If *class-desc* in item class descriptor is omitted, **Object** is inferred when used as a type or the item type is deduced when used with a *list-literal*. A *list-class* of any item type can be passed to a simple untyped **List** class.

[8] *wsr* is an abbreviation for (w)hite (s)pace (r)equired.

[9] Special escape characters: '**n**' – newline, '**t**' – tab, '**v**' – vertical tab, '**b**' – backspace, '**r**' – carriage return, '**f**' – formfeed, and '**a**' – alert. All other characters resolve to the same character including '****', '**"**', and '**'**'.