**SLang reference manual. Version 0.62**

1. **SLang reserved word:**

|  |  |
| --- | --- |
| ***Name*** | ***Kind(comment)*** |
| **abstract** | Unit/Routine characteristic |
| **alias** | Routine characteristic |
| **and** | Boolean operation |
| **as** | Unit level/Type |
| **break** | Statement |
| **check** | Statement |
| **concurrent** | Type |
| **const** | Attribute prefix |
| **else** | Statement |
| **elsif** | Statement |
| **end** |  |
| **ensure** | Predicates |
| **extend** | Unit level |
| **external** | Routine characteristic |
| **final** | Routine characteristic |
| **hidden** | Routine characteristic |
| **if** | Statement |
| **in** | Operator |
| **init** | Routine characteristic |
| **invariant** | Predicates |
| **is** | Definition |
| **loop** | Statement |
| **new** | Statement/Expression |
| **not** | Boolean operation |
| **old** | Expression |
| **or** | Boolean operation |
| **override** | Routine characteristic |
| **pure** | Routine characteristic |
| **ref** | Type |
| **require** | Predicates |
| **return** | Statement/ Expression |
| **routine** | Type |
| **safe** | Routine characteristic |
| **super** | Statement |
| **then** | Statement |
| **this** | Expression |
| **unit** | Unit level |
| **use** | Compilation/Unit/Routine level |
| **val** | Type |
| **variant** | Predicates |
| **while** | Statement |
| **xor** | Boolean operation |

1. **SLang syntax:**

(sem+) Compilation : {[CompilationUnit](#CompilationUnit)}

(sem+) CompialtionUnit: {[UseDirective](#UseDirective)} ([AnonymousRoutine](#AnonymousRoutine)|[StandaloneRoutine](#StandaloneRoutine)|[UnitDeclaration](#UnitDeclaration))

UseDirective: **use** [UseElement](#UseElement) {“**,**”[UseElement](#UseElement)}

UseElement: [FullUnitName](#FullUnitName) [**as** [Identifier](#Identifier)]

FullUnitName: [Identifier](#Identifier) [“**[“**[UnitType](#UnitType){“**,**” [UnitType](#UnitType)}“**]**”]

(sem+) AnonymousRoutine : [StatementsList](#StatementsList)

(sem+) StatementsList: { [Statement](#Statement)[“**;**”]}

StandaloneRoutine:

[**pure**|**safe**] [Identifier](#Identifier) [[FormalGenerics](#FormalGenerics)] [[Arguments](#Arguments)] [“**:**” [Type](#Type)]   
 [[UseDirective](#UseDirective)]

[[RequireBlock](#RequireBlock)]

(“**is**” [InternalRoutineBody](#InternalRoutineBody)|**external** ) | (“**=>**”[Expression](#Expression) )  
 [[EnsureBlock](#EnsureBlock)]  
 [**end**]

Arguments : “**(**”[Argument](#Arguemnt){”**;**” [Argument](#Arguemnt)}“**)**”

Argument : ([[Identifier](#Identifier){“**,**” [Identifier](#Identifier)}“**:**”] [Type](#Type))|([Identifier](#Identifier) “**is**” [Expression](#Expression))

RequireBlock : **require** [**else**] [PredicatesList](#PredicatesList)

EnsureBlock : **ensure** [**then**] [PredicatesList](#PredicatesList)

PredicatesList : [[Predicate](#Predicate) {[”**;**”|“**,**”] [Predicate](#Predicate)}]

Predicate : [BooleanExpression](#BooleanExpression) [[DocumentingComment](#DocumentingComment)]

InternalRoutineBody : { [Statement](#Statement)[“**;**”]|[UnitRoutineDeclaration](#UnitRoutineDeclaration)}

UnitDeclaration:

[**ref**|**val**|**concurrent**|**abstract**|**extend**] **unit** [Identifier](#Identifier) [[FormalGenerics](#FormalGenerics)]  
 [[InheritDirective](#InheritDirective)]  
 [[UseDirective](#UseDirective) [**const** [FullUnitName](#FullUnitName) {“**,**” [FullUnitName](#FullUnitName)}]]

[**is**]  
 {

[FeatureSelection](#FeatureSelection)|

[ConstObjectsDeclaration](#ConstObjectsDeclaration)|

[FeatureDeclaration](#FeatureDeclaration)|

[InheritedFeatureOverriding](#InheritedFeatureOverriding)|

[UnitDeclaration](#UnitDeclaration)

}

[**invariant** [PredicatesList](#PredicatesList)]  
 **end**

InheritDirective: [Parent](#Parent) {“,” [Parent](#Parent)}

Parent : [“**~**”] [UnitTypeName](#UnitTypeName)

GenericInstantiation: “**[**”[Type](#Type) {“**,**” [Type](#Type)}“**]**”

FormalGenerics: “**[**”[FormalGeneric](#FormalGeneric) {“**,**” [FormalGeneric](#FormalGeneric)}“**]**”

FormalGeneric: [Identifier](#Identifier) [“**->**” [Type](#Type) [“**init**” [“**(**”[[Type](#Type) {“**,**” [Type](#Type)}]“**)**”]]]|[“**:**” [UnitType](#UnitType)]

FeatureSelection: **use** [UnitTypeName](#UnitTypeName)”**.**”[Identifier](#Identifier) {[“**,**”] [UnitTypeName](#UnitTypeName)**.**”[Identifier](#Identifier)}

InheritedFeatureOverriding: **override** [UnitTypeName](#UnitTypeName)”**.**”[Identifier](#Identifier)

FeatureDeclaration:

[**hidden**]

[InitDeclaration](#InitDeclaration)|([**override**] [**final**] [UnitAttribiteDeclaration](#UnitAttributeDeclaration)|[UnitRoutineDeclaration](#UnitRoutineDeclaration))

InitDeclaration:

**init** [[Arguments](#Arguments)]

[[RequireBlock](#RequireBlock)]

“**is**” [InternalRoutineBody](#InternalRoutineBody)  
 [[EnsureBlock](#EnsureBlock)]

**end**

UnitRoutineDeclaration:  
 [**pure**|**safe**] [RoutineName](#RoutineName) [[Arguments](#Arguments)] [“**:**” [Type](#Type)]   
 [[RequireBlock](#RequireBlock)]

(“**is**” [InternalRoutineBody](#InternalRoutineBody)|**abstract**|**external** ) | (“**=>**”[Expression](#Expression) )  
 [[EnsureBlock](#EnsureBlock)]

[**end**]

RoutineName : [Identifier](#Identifier)|“**()**”|“:=”|([OperatorName](#OperatorName)[ **alias** [Identifier](#Identifier)])

OperatorName : “=”|“/=”|”<=”|”>=”| “**+**”|“**-**“|“**\***”|“**/**”|“**\**”|“**^**”|**and**|**or**|**not**|**xor**|“=>”|**and then**|**or else**

**… ????** “++” | “--”**????**

ConstObjectsDeclaration : **<To Be Improved>**

**//// const** [ **use** [UnitTypeName](#UnitTypeName) {“**,**” [UnitTypeName](#UnitTypeName) } ] **is**

**const is**

[ [ConstObject](#ConstObject) { “**,**” [ConstObject](#ConstObject)} ]

**end**

ConstObject :

[Constant](#Constant) | ([Idenitifer](#Identifier) [ [CallChain](#CallChain) ])

[ “**..**” [Constant](#Constant) | ([Idenitifer](#Identifier) [ [CallChain](#CallChain) ]) ]

Statement (Starters: identifier, const, break, ?, stringConst, if, while, loop, new, this, super, return, try, check, raise)  
 : [Assignment](#Assignment)  
 | [LocalAttributeDeclaration](#LocalAttributeDeclaration)  
 | [IfCase](#IfCase)  
 | [Loop](#Loop)  
 | **break** [“:”[Label](#Label)]  
 | [FeatureCallOrCreation](#FeatureCallOrCreation)

| **?** [Identifier](#Identifier) | **check** [PredicatesList](#PredicatesList) **end**  
 |**return** [Expression](#Expression)

|[Try](#Try)

| **raise** [[Expression](#Expression)]

Label : [Identifier](#Identifier)

Try : **try**

[StatementsList](#StatementsList)

**catch (**[[Identifier](#Identifier)**:**][UnitType](#UnitType)**)**

[[StatementsList](#StatementsList)]

{**catch (**[[Identifier](#Identifier)**:**][UnitType](#UnitType)**)** [[StatementsList](#StatementsList)]}

[**else** [[StatementsList](#StatementsList)]]

**end**

Assignment  
 : [Writable](#Writable) “**:=**” [Expression](#Expression)

AttributeNamesList

: [**const**] [Identifier](#Identifier) {“**,**”[**const**] [Identifier](#Identifier)}

LocalAttributeDeclaration  
// : [AttributeNamesList](#AttributeNamesList) ([“**:**” [Type](#Type)] **is** [Expression](#Expression))|([“**:**” “**?**” [Type](#Type)] [**is** [Expression](#Expression)])

LocalAttributeDeclaration

: [AttributeNamesList](#AttributeNamesList) ([“**:**” [Type](#Type)] **is** [Expression](#Expression)) |(“**:**” “**?**”[UnitType](#UnitType))

UnitAttributeDeclaration  
 : [AttributeNamesList](#AttributeNamesList) (“**:**” [Type](#Type) [“**:=**” [[[[Arguments](#Arguments)][[RequireBlock](#RequireBlock)]  
 “**is**” [InternalRoutineBody](#InternalRoutineBody) [[EnsureBlock](#EnsureBlock)] **end**](#TupleExpression)](#OldExpression) ] [**is** [Expression](#Expression)])|(**is** [Expression](#Expression))

Writable: [Identifier](#Identifier) [“**(**”[ExprList](#ExprList)“**)**”]{“**.**”[Identifier](#Identifier) “**(**”[ExprList](#ExprList)“**)**”}

ExprList: [Expression](#Expression) {“**,**” [Expression](#Expression)}

BooleanExpression: [Expression](#Expression)

Expression:  
 [IfExpression](#IfExpession) | [FeatureCallOrCreation](#FeatureCallOrCreation) | [Expression](#Expression) [Operator](#Operator) [Expression](#Expression)  
 | [Operator](#Operator) [Expression](#Expression)  
 | [Constant](#Constant) | [TypeOfExpression](#TypeOfExpression)  
 | [OldExpression](#OldExpression) | [RangeExpression](#RangeExpression) | [LambdaExpression](#LambdaExpression) | [TupleExpression](#TupleExpression) | “**(**”[Expression](#Expression)“**)**”

LambdaExpression:

(**routine** [RoutineName](#RoutineName) [[Signature](#Signature)])|[InlineLambdaExpression](#InlineLambdaExpression)

InlineLambdaExpression [: [](#TupleExpression)**[pure](#TupleExpression)**[|](#TupleExpression)**[safe](#TupleExpression)**[]](#TupleExpression) **[routine](#TupleExpression)** [[](#TupleExpression)[[Arguments](#TupleExpression)](#Arguments)[] [“](#TupleExpression)**[:](#TupleExpression)**[”](#TupleExpression) [[Type](#TupleExpression)](#Type)[]](#TupleExpression)

[[](#TupleExpression)[[RequireBlock](#TupleExpression)](#RequireBlock)[]  
 (“](#TupleExpression)**[is](#TupleExpression)**[”](#TupleExpression) [[InternalRoutineBody](#TupleExpression)](#InternalRoutineBody)[|](#TupleExpression) **[external](#TupleExpression)**[)|(“](#TupleExpression)**[=>](#TupleExpression)**[”](#TupleExpression)[[Expression](#TupleExpression)](#Expression) [)  
 [](#TupleExpression)[[EnsureBlock](#TupleExpression)](#EnsureBlock)[]  
 [](#TupleExpression)**[end](#TupleExpression)**]

RangeExpression : [Expression](#Expression)“**..**”[Expression](#Expression)

OldExpression [:](#TupleExpression) **[old](#TupleExpression)** [[Expression](#TupleExpression)](#Expression)

TupleExpression: “**(**”[[TupleElement](#TupleElement) {“**,**”|”;” [TupleElement](#TupleElement)}]“**)**”

TupleElement: [Expression](#Expression)|[Argument](#Arguemnt)

TypeOfExpression: [Expression](#Expression) **is** [UnitType](#UnitType)

Operator: [OperatorName](#OperatorName)|**in**

Constant: [StringConstant](#StringConstant) |[CharacterConstant](#CharacterConstant) |[IntegerConstant](#IntegerConstant) |[RealConstant](#RealConstant) |[BooleanConstant](#BooleanConstant)

IfExpression  
 : **if** [Expression](#Expression) (**is** [IfBodyExpression](#IfBodyExpression))|(**then** [Expression](#Expression))  
 {**elsif** [Expression](#Expression) (**is** [IfBodyExpression](#IfBodyExpression))|(**then** [Expression](#Expression))}  
 **else** [Expression](#Expression)

IfBodyExpression : [ValueAlternative](#ValueAlternative)“:”[Expression](#Expression)

{[ValueAlternative](#ValueAlternative)“:”[Expression](#Expression)}

{**elsif** [Expression](#Expression) (**is** [IfBodyExpression](#IfBodyExpression))|(**then** [Expression](#Expression))}

FeatureCallOrCreation:

([Identifier](#Identifier)|([UnitType](#UnitType)“.”**super**)|**super**|**this**|**return** [“**(**”[[ExpressionList](#ExpressionList)]”**)**”] {[CallChain](#CallChain)})

| ([**new**] [UnitType](#UnitType)|[Identifier](#Identifier)|(“**(**”[Type](#Type) “**)**”) [“.”**init**] [“**(**”[[ExpressionList](#ExpressionList)]”**)**”])

CallChain: “**.**”[Identifier](#Identifier)[“**(**”[[ExpressionList](#ExpressionList)]”**)**”]

ExpressionList: [Expression](#Expression){“**,**” [Expression](#Expression)}

IfCase : **if** [Expression](#Expression)

(**is** [IfBody](#IfBody))|(**then** [[StatementsList](#StatementsList)])

[**else** [ [StatementsList](#StatementsList) ]]  
 **end**

IfBody : [ValueAlternative](#ValueAlternative)“:”[StatementsList](#StatementsList)

{[ValueAlternative](#ValueAlternative)“:”[StatementsList](#StatementsList)}

{**elsif** [Expression](#Expression) (**is** [IfBody](#IfBody))|(**then** [[StatementsList](#StatementsList)])}

ValueAlternative : [Expression](#Expression) [“**..**”[Expression](#Expression) ] {“,”[Expression](#Expression) [“**..**”[Expression](#Expression)]}

Loop : [**while** [BooleanExpression](#BooleanExpression)] **loop** [”**:**”[Label](#Label)]

[[StatementsList](#StatementsList)]

[**invariant** [PredicatesList](#PredicatesList)]  
 [**variant** [PredicatesList](#PredicatesList)]  
 [**while** [BooleanExpression](#BooleanExpression)]  
 **end**

Type : [UnitType](#UnitType)|[AnchorType](#AnchorType)|[MultiType](#MultiType)|”**?**” [Type](#Type) |[TupleType](#TupleType)|[RangeType](#RangeType)|[RoutineType](#RoutineType)

RoutineType: **routine** [Signature](#Signature)

Signature: [“**(**”[[Type](#Type) {“**;**” [Type](#Type)}]“**)**”]

RangeType:

(([Constant](#Constant)|[Idenitifer](#Identifier))“**..**”([Constant](#Constant)|[Idenitifer](#Identifier)))

|

([Constant](#Constant)|[Idenitifer](#Identifier)) {“**|**” ([Constant](#Constant)|[Idenitifer](#Identifier))})

AnchorType  
 : **as** (**this**|[Identifier](#Identifier))

MultiType  
 : [UnitType](#UnitType) {“**|**”[UnitType](#UnitType)}

TupleType  
 : “**(**”[[TupleField](#TupleField) {“**,**”|”**;**” [TupleField](#TupleField)}]“**)**”

TupleField  
 : [[Identifier](#Identifier) {“**,**” [Identifier](#Identifier)}“**:**”] [UnitType](#UnitType)

UnitTypeName: [Identifier](#Identifier) [[GenericInstantiation](#GenericInstantiation)]

UnitType: [**ref**|**val**|**concurrent**] [UnitTypeName](#UnitTypeName)

Последующие правила – это лексическая грамматика. Ее не обязательно описывать правилами. Можно, конечно, но в любом случае отдельно от синтаксиса языка.

Comment : ( “**//**” { [Character](#Character) } )  
 | ( ”**/\***” { [Character](#Character) } “**\*/**” )

DocumentingComment : “**///**” { [Character](#Character) }

Identifier  
 : [Letter](#Letter) { [Letter](#Letter) | [Digit](#Digit) | ’\_’ }

StringConstant  
 : “**”**” { [Character](#Character) } “**”**”

CharacterConstant  
 : “**’**” [Character](#Character) “**’**”

IntegerConstant  
 : [ “**+**”|”**-**“ ] [Digit](#Digit) { [Digit](#Digit) } [ “**x**” “**B**”|”**b**”|”**H**”|”**h**” ]

RealConstant  
 : [ “**+**”|”**-**“ ] [Digit](#Digit) { [Digit](#Digit) } “.”{ [Digit](#Digit) } [“**e**”|”**E**”] [“**+**”|”**-**“] [Digit](#Digit) { [Digit](#Digit) }

BooleanConstant  
 : **true** | **false**

Character  
 : [Letter](#Letter) | [Digit](#Digit) | [Symbol](#Symbol)

Letter : ‘**A**’ | .. ’**Z**’ | ’**a**’ | ..’**z**’

Digit : ’**0**’ | ..’**9**’ | ’**A**’..’**F**’

Symbol : ASCII symbol 0..255

1. **SLang semantics:**

SEM001\_Compilation\_Full\_Validity (CFV): <[Compilation](#Compilation)> is valid if and only if all <[CompilationUnit](#CompilationUnit)>s are valid

SEM002\_Compilation\_Partial\_Validity (CPV): If not all <[CompilationUnit](#CompilationUnit)>s are valid then <[Compilation](#Compilation)> is partially valid.

SEM003\_Unit\_Validity (UV): <[CompilationUnit](#CompilationUnit)> is valid if and only if it has all its <[UseDirective](#UseDirective)>s as valid if any and

<[AnonymousRoutine](#AnonymousRoutine)> or <[StandaloneRoutine](#StandaloneRoutine)> or <[UnitDeclaration](#UnitDeclaration)> is valid as well.

SEM004\_Statement\_List (SLV): <[StatementsList](#StatementsList)> is valid if and only if every <[Statement](#Statement)> in the list is valid

SEM005\_AnonymousRoutine (ARV): <[AnonymousRoutine](#AnonymousRoutine)> is valid if and only if its every <[Statement](#Statement)> is valid

SEM006\_Statement (STMTV): <[Statement](#Statement)> is valid if and only if it is one of the following <[Assignment](#Assignment)> or <[LocalAttributeDeclaration](#LocalAttributeDeclaration)> or <[IfCase](#IfCase)> or <[IfCase](#IfCase)> or <[Loop](#Loop)> or <[Break](#Break)> or <[FeatureCallOrCreation](#FeatureCallOrCreation)> or <[Detach](#Detach)> or <[Check](#Check)> or <[Return](#Return)> or <[Try](#Try)> or <[Raise](#Raise)>

SEM007\_Assignment (AV): <[Assignment](#Assignment)> is valid if and only if

SEM008\_LocalAttribute (LAV): <[LocalAttributeDeclaration](#LocalAttributeDeclaration)> is valid if and only if

SEM009\_If (IV): <[IfCase](#IfCase)> is valid if and only if

SEM010\_Loop (LV): <[Loop](#Loop)> is valid if and only if

SEM011\_Break (BV): <[Break](#Break)> is valid if and only if

SEM012\_FeatureCallOrCreation (FCCV): <[FeatureCallOrCreation](#FeatureCallOrCreation)> is valid if and only if

SEM013\_Detach (DV): <[Detach](#Detach)[> is valid if and only if Identifier](#Identifier)

SEM014\_Check (CHKV): <[Check](#Check)> is valid if and only if [PredicatesList](#PredicatesList)

SEM015\_Return (RETV): <[Return](#Return)> is valid if and only if Expression is valid

SEM016\_Try (TRYV): <[Try](#Try)> is valid if and only if

SEM017\_Raise (RV): <[Raise](#Raise)> is valid if and only if [Expression](#Expression)

Я тут по воле судьбы стал читать спек ObjectiveC - там есть разного вида расширения классов. Вот при случае ты бы спросило Бертрана почему он отказался от расширений. Ведь по сути ответ Бертрана или ответ Эйфеля - если тебе понадобилось добавить свойство то заведи класс наследник и в него добавь. Но это не поменяет поведение тех мест где в явном виде создаются экземпляры класса родителя, а в случае расширений именно так и происходит. Вот пример

**unit** Parent

      factory: Parent **is**

**return** Parent   // Dynamic type of result is Parent with all extensions in ...

**end**

**end**

**extend** **unit** Parent

     foo **is**

**end**

**end**

**use** Parent **as** p

object := p.factory // At run-time object will have feature foo as we have unit extension as part of compilation ...

object.foo // This call is valid both statically and system-wide

Можно такое поддержать - да можно. Просто каждая инструкция создания объекта(экземпляра юнита) правильно апдейтиться умным линкером. Аналог ниже

**unit** Parent

      factory: Parent **is**

**return** **as this**   // Dynamic type of result will be liek this taking into account all possible extensions ...

**end**

**end**

**extend** **unit** Parent

     foo **is**

**end**

**end**

**extend** **unit** Parent

   goo: **as this**

**end**

**use** Parent **as** p

object := p.factory // At run-time object will have feature foo as we have unit extension as part of compilation ...

object.foo // This call is valid both statically and system-wide

object := object.goo

Т.е. все extensions для каждого юнита собирается при проверке семантики и подтверждается семантическая правильность юнита со всеми екстиеншенами. Т.е. все type safe !!!! И рабоатет для всех видов features

Другой вопрос может ли extension иметь invariant и свою init procedure - по идее - ДА! Так как если мы добавляем атрибут, то по сути возникает необходимость в invariant and init procedure update. С invariant все просто - они просто добавляются а вот с init procedure тонкость - сигнатуры ... Их объединить не получится автоматом ... А вот если сказать что расширять можно только цепочкой тогда ... Смотри пример

**unit** Parent

      factory: Parent **is**

**return** **as this**   // Dynamic type of result will be like this taking into account all possible extensions ...

**end**

**end**

**extend** **unit** Parent

**init is end**

**end**

**extend** **unit** Parent

     foo **is**

**end**

**init is end**// New version of init

**end**

**unit**B**extension**A

   goo: **as this**

**init is end**// New version of init

**end**

Т.е. в таком варианте комбинируется наследование и расширение - получаются новые юниты A и B и при этом сам юнит Child идентичен юниту B, что чертовски плохо!!! Тогда можно присвоить объект типа А сущности типа Child и обратитьтся к свойству введенному в B - т.е. не type safe, А следовательно A и B не являются юнитами!!! И типов они не пораждают!!! Короче надо думать еще ...

У тебя голова еще не съехала пока ты все это читал ? :-)

Теперь понимаешь почему Бертран не стал вводить class extensions :-) Но я еще подумаю и мы введем!!! И введем без потери перформанса как в ObjectiveC  - а кстати в SWIFT Apple оставил category & extension - так они называются в ObjectiveC?