**SLang reference manual. Version 0.93**

1. **SLang reserved word:**

|  |  |  |
| --- | --- | --- |
| ***#*** | ***Name*** | ***Kind(comment)*** |
| 1 | **alias** | Unit/Routine characteristic. The alternative name of the unit or routine. |
| 2 | **as** | Unit level/Type. Another name or anchor |
| 3 | **break** | Statement. Exit the current scope. |
| 4 | **concurrent** | Type. It can be a unit or an attribute. |
| 5 | **const** | Unit level. Start of constant objects declaration section and import of constant objects |
| 6 | **do** | Statement. Start of the block. |
| 7 | **else** | Statement. Start of else block. |
| 8 | **elsif** | Statement. Start of the else if block |
| 9 | **end** | End of block or construction |
| 10 | **ensure** | Predicate. Routine post-condition clause start. |
| 11 | **extend** | Unit level. Used to support inheritance and unit extensions. |
| 12 | **final** | Routine characteristic. Routine can not be overridden down in the inheritance hierarchy |
| 13 | **foreign** | Routine characteristic. Body of the routine is coded in 3rd party language |
| 14 | **if** | Statement. Conditional statement start. |
| 15 | **in** | Operator. Belongs to the range of values. |
| 16 | **init** | Routine characteristic. Start of the initialization procedure. |
| 17 | **is** | Definition of the initial value of an attribute. Check for a value of some expression |
| 18 | **new** | Statement/Expression. Creation of an object. |
| 19 | **old** | Expression. Value of some attribute before the routine start. To be used in post-conditions only. For the routine body, it means a call to the previous version of the overridden routine – precursor call |
| 20 | **override** | Unit member characteristics. States that this attribute overrides all possible inherited versions. |
| 21 | **pure** | Routine characteristic. Routine is prohibited to write into unit attributes or read them. Must work only with its parameters. No side effects. |
| 22 | **raise** | Statement. Raise an exception potentially with some object as an argument. With no parameters – rerasie exception occurred. |
| 23 | **ref** | Type. States that an object will be of the reference nature |
| 24 | **rename** | Unit level: give new name to the inherited member |
| 25 | **require** | Predicate. Start of precondition clause of routine, unit or loop invariant |
| 26 | **return** | Statement/ Expression. Stopes the execution of the routine and returns result in case of function |
| 27 | **rigid** | Attribute prefix. A deep version of attribute immutability. |
| 28 | **rtn** | Type. Has 2 meanings – denotes the routine type after a colon (‘:’) or creates a routine object from some routine in expressions |
| 29 | **safe** | Routine characteristic. Routine is prohibited to write into unit attributes. |
| 30 | **select** | Unit level: select one version among several versions to resolve ambiguity with polymorphic assignment |
| 31 | **this** | Expression. Reference to the current object. |
| 32 | **unit** | Unit level. Start of the unit description. |
| 33 | **use** | Compilation/Unit/Routine level. States that some unit will be used as a module (singleton) at a unit or routine level. Allows renaming units as well. |
| 34 | **val** | Type. States that an object will be of value nature. Object itself not a reference to it. |
| 35 | **var** | Attribute/parameter prefix. States that attribute can be assigned many times. It is a variable attribute of any type including routine one. If it is routine parameters then routines with side–effects can be called upon this parameter, as well as assignment to it. |
| 36 | **virtual** | Unit/Routine characteristic. Bodyless unit member or objectless unit |
| 37 | **when** | Statement. Exception handling condition clause. |
| 38 | **while** | Statement. Loop condition clause. |

Names below are names of operations and to be treated by the compiler as identifiers but not keywords.

|  |  |
| --- | --- |
| ***and*** | *Boolean operation* |
| ***xor*** | *Boolean operation* |
| ***or*** | *Boolean operation* |
| ***not*** | *Boolean operation* |
| ***implies*** | *Boolean operation* |
| ***and then*** | *Boolean operation* |
| ***or else*** | *Boolean operation* |

1. **SLang syntax:**

([val](#VAL001_Compilation_Full_Validity), sem-) Compilation : {[CompilationUnit](#CompilationUnit)}

([val](#VAL002_Compilation_Partial_Validity), sem-) CompilationUnit: {[RenameDirective](#RenameDirective)}

[AnonymousRoutine](#AnonymousRoutine)|[StandaloneRoutine](#StandaloneRoutine)|[UnitDeclaration](#UnitDeclaration))

(val-, sem-) RenameDirective: **rename** [RenameElement](#RenameElement) {“**,**” [RenameElement](#RenameElement)}

(val-, sem-) RenameElement: [Type](#Type) **as** [Identifier](#Identifier)

(val-, sem-) UseDirective: [**use** [[UseElement](#UseElement) {“**,**”[UseElement](#UseElement)}] [**const** [FullUnitName](#FullUnitName) {“**,**” [FullUnitName](#FullUnitName)}]]

(val-, sem-) UseElement: [Type](#Type) [**as** [Identifier](#Identifier)]

(val-, sem-) FullUnitName: [Identifier](#Identifier) [“**[“** [FactualGenericType](#FactualGenericType){“**,**” [FactualGenericType](#FactualGenericType)}“**]**”]

(val-, sem-) FactualGenericType: [UnitType](#UnitType) | [Constant](#Constant) | [RoutineType](#RoutineType)

([val](#VAL005_AnonymousRoutine), sem-) AnonymousRoutine : [StatementsList](#StatementsList)

([val](#VAL004_Statement_List), sem-) StatementsList: { [Statement](#Statement)[“**;**”]}

([val](#VAL004_Statement_List), sem-) WhenClause: **when** [[Identifier](#Identifier)**:**][UnitType](#UnitType) **do** [StatementsList](#StatementsList)

(val-, sem-) InnerBlock: [[Label](#Label)”**:**”] **do** [“(”[Identifier](#Identifier) {“,” [Identifier](#Identifier)} “)”] /\* Do not check

invariants for these objects within the block \*/

[StatementsList](#StatementsList)

[ [WhenClause](#WhenClause) {[WhenClause](#WhenClause)} [**else** [[StatementsList](#StatementsList)]] ]

(val-, sem-) StandaloneRoutine:

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

([[RequireBlock](#RequireBlock)] [InnerBlock](#InnerBlock)|**foreign** [[EnsureBlock](#EnsureBlock)] [**end**] ) | (“**=>**”[Expression](#Expression) )

(val-, sem-) Parameters : “**(**”[Parameter](#Parameter){”**;**” [Parameter](#Parameter)}“**)**”

(val-, sem-) Parameter : ([[**var**] [Identifier](#Identifier){“**,**” [**var**] [Identifier](#Identifier)} “**:**”] [Type](#Type))|([Identifier](#Identifier) “**is**” [Expression](#Expression))

(val-, sem-) RequireBlock : **require** [PredicatesList](#PredicatesList)

(val-, sem-) EnsureBlock : **ensure** [PredicatesList](#PredicatesList)

(val-, sem-) InvariantBlock: **require** [PredicatesList](#PredicatesList)

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

(val-, sem-) Predicate : [BooleanExpression](#BooleanExpression) [[DocumentingComment](#DocumentingComment)]

(val-, sem-) UnitDeclaration:

[**final**] [**ref**|**val**|**concurrent**|**virtual**|**extend**]

**unit** [Identifier](#Identifier) [[AliasName](#AliasName)] [[FormalGenerics](#FormalGenerics)] [[InheritDirective](#InheritDirective)] [[UseDirective](#UseDirective)]  
{

[MemberSelection](#FeatureSelection)|

[InheritedMemberOverriding](#InheritedFeatureOverriding)|

([InheritedMemberRename](#InheritedMemberRename) |[RenameDirective](#RenameDirective))|

[InitProcedureInheritance](#InitProcedureInheritance) |

[ConstObjectsDeclaration](#ConstObjectsDeclaration)|

([ [MemberVisibility](#MemberVisibility) “:”] [MemberDeclaration](#FeatureDeclaration))

}

[[InvariantBlock](#InvariantBlock)]

**end**

(val-, sem-) InheritDirective: **extend** [Parent](#Parent) {“,” [Parent](#Parent)}

(val-, sem-) Parent : [“**~**”] [UnitTypeName](#UnitTypeName)

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

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

(val-, sem-) FormalGeneric: [Identifier](#Identifier) ([“**extend**” [Type](#Type) ] [“**init**” [Identifier](#Identifier) [[Signature](#Signature)]])| [“**:**” [UnitType](#UnitType) | [RoutineType](#RoutineType)]

(val-, sem-) MemberSelection: **select** [UnitTypeName](#UnitTypeName)”**.**”[Identifier](#Identifier)[[Signature](#Signature)] {[“**,**”] [UnitTypeName](#UnitTypeName)**.**”[Identifier](#Identifier)[[Signature](#Signature)]}

(val-, sem-) InitProcedureInheritance: **init** [InitFromParent](#InitFromParent) {[“**,**”] [InitFromParent](#InitFromParent)}

(val-, sem-) InitFromParent: [UnitTypeName](#UnitTypeName)”**.**” [RoutineName](#RoutineName) [[Signature](#Signature)]

(val-, sem-) InheritedMemberRename: **rename** [RenamePair](#RenamePair) {“**,**” [RenamePair](#RenamePair) }

(val-, sem-) RenamePair: [UnitTypeName](#UnitTypeName)”**.**”[Identifier](#Identifier)[[Signature](#Signature)] **as** [Identifier](#Identifier)

(val-, sem-) InheritedMemberOverriding: **override** [UnitTypeName](#UnitTypeName)”**.**”[Identifier](#Identifier)[[Signature](#Signature)]

(val-, sem-) MemberVisibility: “**{**” [**this**| [UnitTypeName](#UnitTypeName) {“**,**” [UnitTypeName](#UnitTypeName)} ] “**}**”

(val-, sem-) MemberDeclaration:

[[MemberVisibility](#MemberVisibility)]

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

(val-, sem-) UnitRoutineDeclaration:  
 ([**pure**|**safe**] [RoutineName](#RoutineName) [**final** [Identifier](#Identifier)])|**init** [[Parameters](#Parameters)] [“**:**” [Type](#Type)] [[UseDirective](#UseDirective)]

([[RequireBlock](#RequireBlock)] [InnerBlock](#InnerBlock)|**virtual**|**foreign** [[EnsureBlock](#EnsureBlock)]

[**end**]) | (“**=>**”[Expression](#Expression) )

/\* **final** [Identifier](#Identifier) it allows to call this version from any descendant\*/

(val-, sem-) AliasName : **alias** [Identifier](#Identifier)

(val-, sem-) RoutineName : ( [Identifier](#Identifier) [[Identifier](#Identifier)] )|“**()**”|“:=”|([OperatorName](#OperatorName) [[AliasName](#AliasName)]) (val-, sem-) OperatorName : [OperatorSign](#OperatorSign) [[OperatorSign](#OperatorSign)]

(val-, sem-) OperatorSign : “**=**” | “**/**” | ”**<**” | ”**>**” | “**+**” | “**-**“ | “**\***” | “**\**” | “**^**” | ”**&**” | ”|”

------------- Old variant

/\*

(val-, sem-) RoutineName : [Identifier](#Identifier)|“**()**”|“:=”|([OperatorName](#OperatorName) [[AliasName](#AliasName)])

(val-, sem-) OperatorName : “**=**”|“**/=**”|”**<=**”|”**>=**”|“**+**”|“**-**“|“**\***”|“**/**”|“**\**”|“**\*\***”|“**^**”|”**&**”|”|”|

**and**|**or**|**not**|**xor**|**implies**|“->”|**and then**|**or else** |“+**=**”|“-**=**”|“\***=**”|“/**=**”|“++” |“--”

\*/

(val-, sem-) ConstObjectsDeclaration :

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

(val-, sem-) ConstObject :

[Constant](#Constant) | (“{” [RegularExpression](#RegularExpression) “}” [IntegerConstant](#IntegerConstant) [“+”])

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

(val-, sem-) RegularExpression:

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

([val](#VAL006_Statement), sem-) Statement:

[Assignment](#Assignment)

| [LocalAttributeCreation](#LocalAttributeCreation)

| UnitAttributeCreation

| ProcedureCall

// | [MemberCallOrCreation](#FeatureCallOrCreation)

| [IfCase](#IfCase)

| [Loop](#Loop)

| [Break](#Break)

| **?** [Identifier (val-, sem-)](#Identifier)   
 |[Return](#Return)

|[HyperBlock](#HyperBlock)

| **raise** [[Expression](#Expression)] (val-, sem-)

([val](#VAL015_Return), sem-) Return: **return** [[Expression](#Expression)]

([val](#VAL011_Break), sem-) Break: **break** [[Label](#Label)“:”]

(val-, sem-) Label : [Identifier](#Identifier)

(val-, sem-) HyperBlock :

[[RequireBlock](#RequireBlock)]

[InnerBlock](#InnerBlock)

[[EnsureBlock](#EnsureBlock)]

**end**

([val](#VAL007_Assignment), sem-) Assignment:

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

(val-, sem-) AttributeNamesList:

[**var** | **rigid**] [Identifier](#Identifier) {“**,**”[**var** | **rigid**] [Identifier](#Identifier)}

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

([val](#VAL008_LocalAttribute), sem-) LocalAttributeCreation:

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

|

( “**(**”[AttributeNamesList](#AttributeNamesList) “**)**” **is** [Expression](#Expression) )

(val-, sem-) UnitAttributeDeclaration:

[AttributeNamesList](#AttributeNamesList) “**:**” [Type](#Type) [ **rtn** “**:=**”[[ [[[Parameters](#Parameters)] [HyperBlock](#HyperBlock)](#TupleExpression)](#OldExpression) ] ]

(val-, sem-) Writable: [Identifier](#Identifier) [“**(**”[ExpressionList](#ExpressionList)“**)**”]{“**.**”[Identifier](#Identifier) “**(**”[ExpressionList](#ExpressionList)“**)**”}

| “**(**”[Identifier](#Identifier) [“**,**”[Identifier](#Identifier)]“**)**”

Дорабатывать надо writable …

(a,b,c) := (E1, E2, E3)

(a.x,b.y.z,c) := (E1, E2, E3)

A := expr

a.b.c := expr

Что-то я не уверен про “**(**”[ExpressionList](#ExpressionList)“**)**”

x is foo(…)

x.y := expr

foo(…).y := expr

(val-, sem-) BooleanExpression: [Expression](#Expression)

(val-, sem-) Expression:

[ “{”[UnitType](#UnitType) “}”]

[IfExpression  
 | [FunctionCall](#FeatureCallOrCreation)](#IfExpession)

| [LocalObjectCreation](#FeatureCallOrCreation)

// | [MemberCallOrCreation](#FeatureCallOrCreation) | [Expression](#Expression) [Operator](#Operator) [Expression](#Expression)  
 | [Operator](#Operator) [Expression](#Expression)  
 | [Constant](#Constant) | [TypeOfExpression](#TypeOfExpression)  
 | [OldExpression](#OldExpression) | [RangeExpression](#RangeExpression) | [LambdaExpression](#LambdaExpression) | [TupleExpression](#TupleExpression) | [RefExpression](#RefExpression) | “**(**”[Expression](#Expression)“**)**”{[CallChain](#CallChain)}

(val-, sem-) [RefExpression:](#TupleExpression) **ref** [Expression](#Expression)

(val-, sem-) LambdaExpression:

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

(val-, sem-) InlineLambdaExpression [: [](#EnsureBlock)**[pure](#EnsureBlock)**[|](#EnsureBlock)**[safe](#EnsureBlock)**[]](#EnsureBlock) **[rtn](#EnsureBlock)** [[](#EnsureBlock)[[Parameters](#EnsureBlock)](#Parameters)[] [“](#EnsureBlock)**[:](#EnsureBlock)**[”](#EnsureBlock) [[Type](#EnsureBlock)](#Type)[]](#EnsureBlock)

[( [](#EnsureBlock)[[RequireBlock](#EnsureBlock)](#RequireBlock)[]](#EnsureBlock) [[InnerBlock](#EnsureBlock)](#InnerBlock) [|](#EnsureBlock) **[foreign](#EnsureBlock)** [[[EnsureBlock](#EnsureBlock)] [](#EnsureBlock)**[end](#EnsureBlock)**[] )|(“](#EnsureBlock)**[=>](#EnsureBlock)**[”](#EnsureBlock)[[Expression](#EnsureBlock)](#Expression) [)](#EnsureBlock)

(val-, sem-) RangeExpression : [Expression](#Expression)“**..**”[Expression](#Expression)

(val-, sem-) OldExpression [:](#TupleExpression) **[old](#TupleExpression)** [[Expression](#TupleExpression)](#Expression)(val-, sem-) TupleExpression: “**(**”[[TupleElement](#TupleElement) {“**,**”|”;” [TupleElement](#TupleElement)}]“**)**”

(val-, sem-) TupleElement: [Expression](#Expression)| [Parameter](#Parameter)

(val-, sem-) TypeOfExpression: [Expression](#Expression) **is** [UnitType](#UnitType)

(val-, sem-) Operator: [OperatorName](#OperatorName)|**in**

(val-, sem-) Constant: [[UnitType](#UnitType) “.”] [StringConstant](#StringConstant) |[CharacterConstant](#CharacterConstant) |[IntegerConstant](#IntegerConstant) |[RealConstant](#RealConstant) |[BooleanConstant](#BooleanConstant)

|[Identifier](#Identifier)

(val-, sem-) IfExpression  
 : **if** [Expression](#Expression) (**is** [IfBodyExpression](#IfBodyExpression))|(**do** [Expression](#Expression))  
 {**elsif** [Expression](#Expression) (**is** [IfBodyExpression](#IfBodyExpression))|(**do** [Expression](#Expression))}  
 **else** [Expression](#Expression)

(val-, sem-) IfBodyExpression : [ValueAlternative](#ValueAlternative)“:”[Expression](#Expression)

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

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

(val-, sem-) MemberCallOrCreation:

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

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

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

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

(val-, sem-) CallChain: “**.**”[Identifier](#Identifier)[“**(**”[[ExpressionList](#ExpressionList)]”**)**”]

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

([val](#VAL009_If), sem-) IfCase : **if** [Expression](#Expression)

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

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

(val-, sem-) IfBody : ( [ValueAlternative](#ValueAlternative)“**:**”[StatementsList](#StatementsList)

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

( “**(**” [MemberDesciption](#memberDescription) {“,”} [MemberDesciption](#memberDescription) “**)**” )

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

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

(val-, sem-) MemberDescription : ( [**rtn**] [RoutineName](#RoutineName) [[Signature](#Signature)] )|( [Idenitifer](#Identifier) “**:**”[UnitType](#UnitType) )

([val](#VAL010_Loop), sem-) Loop :

[**while** [BooleanExpression](#BooleanExpression)]

[[RequireBlock](#RequireBlock)]

[InnerBlock](#InnerBlock)

[**while** [BooleanExpression](#BooleanExpression)]

[[EnsureBlock](#EnsureBlock)]

**end**

(val-, sem-) ConstantExpression : [Expression](#Expression)

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

(val-, sem-) RoutineType: **rtn** [[Signature](#Signature)]

(val-, sem-) Signature: “**(**”[[Type](#Type) {“**,**”|“**;**” [Type](#Type)}]“**)**”[“**:**” [Type](#Type)]

(val-, sem-) ConstantExpression : [Expression](#Expression)

(val-, sem-) RangeType:

([ConstantExpression](#ConstantExpression) “**..**”[ConstantExpression](#ConstantExpression))

|

([ConstantExpression](#ConstantExpression) {“**|**” [ConstantExpression](#ConstantExpression)})

(val-, sem-) AnchorType  
 : **as** (**this**|[Identifier](#Identifier) [[Signature](#Signature)])

(val-, sem-) MultiType  
 : [UnitType](#UnitType) {“**|**”[UnitType](#UnitType)}

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

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

(val-, sem-) UnitTypeName: [Identifier](#Identifier) [[GenericInstantiation](#GenericInstantiation)]

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

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

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) } “**\*/**” )

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

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

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

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

(“0” (“**x**”|“**X**”|“**b**”|“**B**”|“**o**”|“**O**”) [Digit](#Digit) { [Digit](#Digit) })

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 validity:**

VAL001\_Compilation\_Full\_Validity (CFV): <[Compilation](#Compilation)> is valid if and only if all <[CompilationUnit](#CompilationUnit)>s are valid

VAL002\_Compilation\_Partial\_Validity (CPV): If not all <[CompilationUnit](#CompilationUnit)>s are valid then <[Compilation](#Compilation)> is partially valid.

VAL003\_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.

VAL004\_Statement\_List (SLV): <[StatementsList](#StatementsList)> is valid if and only if every <[Statement](#Statement)> in the list is valid

VAL005\_AnonymousRoutine (ARV): <[AnonymousRoutine](#AnonymousRoutine)> is valid if and only if its every <[Statement](#Statement)> is valid

VAL006\_Statement (STMTV): <[Statement](#Statement)> is valid if and only if valid of one of the following <[Assignment](#Assignment)> or <[LocalAttributeCreation](#LocalAttributeCreation)> or <[IfCase](#IfCase)> or <[IfCase](#IfCase)> or <[Loop](#Loop)> or <[Break](#Break)> or <[MemberCallOrCreation](#FeatureCallOrCreation)> or <[Detach](#Detach)> or <[Check](#Check)> or <[Return](#Return)> or <[Try](#Try)> or <[Raise](#Raise)>

VAL007\_Assignment (AV): <[Assignment](#Assignment)> is valid if and only if <[Writable](#Writable)> and <[Expression](#Expression)> are both valid and type of <[Expression](#Expression)> conforms or converts into type of <[Writable](#Writable)>

VAL008\_LocalAttribute (LAV): <[LocalAttributeCreation](#LocalAttributeCreation)> is valid if and only if

VAL009\_If (IV): <[IfCase](#IfCase)> is valid if and only if

VAL010\_Loop (LV): <[Loop](#Loop)> is valid if and only if it has no while or only one while clause and …

VAL011\_Break (BV): <[Break](#Break)> is valid if and only if

VAL012\_MemberCallOrCreation (FCCV): <[MemberCallOrCreation](#FeatureCallOrCreation)> is valid if and only if

VAL013\_Detach (DV): <[Detach](#Detach)[> is valid if and only if Identifier](#Identifier)

VAL014\_Check (CHKV): <[Check](#Check)> is valid if and only if [PredicatesList](#PredicatesList)

VAL015\_Return (RETV): <[Return](#Return)> is valid if and only if Expression is valid and <[Return](#Return)> is in the body of the function and type of Expression conforms to the type of the function.

VAL016\_Raise (RV): <[Raise](#Raise)> is valid if and only if [Expression](#Expression)

1. **SLang semantics:**

SEM001\_: ..