**SLang reference manual. Version 0.73**

1. **SLang reserved word:**

|  |  |
| --- | --- |
| ***Name*** | ***Kind(comment)*** |
| **abstract** | Unit/Routine characteristic |
| **alias** | Unit/Routine characteristic |
| **as** | Unit level/Type |
| **break** | Statement |
| **check** | Statement |
| **concurrent** | Type |
| **const** | Attribute prefix |
| **deep** | Attribute prefix |
| **else** | Statement |
| **elsif** | Statement |
| **end** | End of block or construction |
| **ensure** | Predicates |
| **extend** | Unit level |
| **final** | Routine characteristic |
| **foreign** | Routine characteristic |
| **hidden** | Routine characteristic |
| **if** | Statement |
| **in** | Operator |
| **init** | Routine characteristic |
| **invariant** | Predicates |
| **is** | Definition |
| **loop** | Statement |
| **new** | Statement/Expression |
| **old** | Expression |
| **override** | Routine characteristic |
| **pure** | Routine characteristic |
| **ref** | Type |
| **require** | Predicate |
| **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** | Predicate |
| **while** | Statement |

Below keywords are names of operations from unit Boolean and processed by the compiler as Boolean operations but not keywords.

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

1. **SLang syntax:**

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

([val](#VAL002_Compilation_Partial_Validity), sem-) CompialtionUnit: {[UseDirective](#UseDirective)} ([AnonymousRoutine](#AnonymousRoutine)|[StandaloneRoutine](#StandaloneRoutine)|[UnitDeclaration](#UnitDeclaration))

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

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

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

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

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

(val-, sem-) StandaloneRoutine:

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

[[RequireBlock](#RequireBlock)]

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

(val-, sem-) Arguments : “**(**”[Argument](#Arguemnt){”**;**” [Argument](#Arguemnt)}“**)**”

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

[Expression](#Expression))

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

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

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

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

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

(val-, sem-) UnitDeclaration:

[**final**] [**ref**|**val**|**concurrent**|**abstract**|**extend**] **unit** [Identifier](#Identifier) [[FormalGenerics](#FormalGenerics)]

[[AliasName](#AliasName)]  
 [[InheritDirective](#InheritDirective)]  
 [**use** [[UseElement](#UseElement) {“**,**”[UseElement](#UseElement)}] [**const** [FullUnitName](#FullUnitName) {“**,**” [FullUnitName](#FullUnitName)}]]

[**is**]  
 {

[FeatureSelection](#FeatureSelection)|

[ConstObjectsDeclaration](#ConstObjectsDeclaration)|

[FeatureDeclaration](#FeatureDeclaration)|

[InheritedFeatureOverriding](#InheritedFeatureOverriding)|

[UnitDeclaration](#UnitDeclaration)

}

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

(val-, sem-) InheritDirective: [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) [“**->**” [Type](#Type) [“**init**” [[Signature](#Signature)]]]| [“**:**” [UnitType](#UnitType)]

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

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

(val-, sem-) FeatureDeclaration:

[**hidden**] [**override**]

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

(val-, sem-) InitDeclaration:

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

[[RequireBlock](#RequireBlock)]

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

**end**

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

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

[**end**]

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

(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 is** [ [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)

| [LocalAttributeDeclaration](#LocalAttributeDeclaration)

| [IfCase](#IfCase)

| [Loop](#Loop)

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

| [FeatureCallOrCreation](#FeatureCallOrCreation)

| **?** [Identifier](#Identifier) | **check** [PredicatesList](#PredicatesList) **end** ([val](#VAL014_Check), sem-)  
 |**return** [[Expression](#Expression)] ([val](#VAL015_Return), sem-)

|[Try](#Try)

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

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

([val](#VAL016_Try), sem-) Try :

**try**

[StatementsList](#StatementsList)

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

[[StatementsList](#StatementsList)]

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

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

**end**

**/// Alternative try – block with exception check**

**do**

[StatementsList](#StatementsList)

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

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

**end**

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

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

(val-, sem-) AttributeNamesList:

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

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

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

|

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

(val-, sem-) UnitAttributeDeclaration:

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

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

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

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

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

(val-, sem-) 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) | [RefExpression](#RefExpression) | “**(**”[Expression](#Expression)“**)**”{[CallChain](#CallChain)}

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

(val-, sem-) LambdaExpression:

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

(val-, sem-) 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) **[foreign](#TupleExpression)**[)|(“](#TupleExpression)**[=>](#TupleExpression)**[”](#TupleExpression)[[Expression](#TupleExpression)](#Expression) [)  
 [](#TupleExpression)[[EnsureBlock](#TupleExpression)](#EnsureBlock)[]  
 [](#TupleExpression)**[end](#TupleExpression)**]

(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)|[Argument](#Arguemnt)

(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)

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

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

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

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

(val-, sem-) FeatureCallOrCreation:

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

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

| ([**new**] [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))|(**then** [[StatementsList](#StatementsList)])

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

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

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

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

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

([val](#VAL010_Loop), sem-) Loop : [**while** [BooleanExpression](#BooleanExpression)] **loop** [”**:**”[Label](#Label)]

[StatementsList](#StatementsList)

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

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

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

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

(val-, sem-) RangeType:

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

|

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

(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)

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 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 <[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)>

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): <[LocalAttributeDeclaration](#LocalAttributeDeclaration)> 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\_FeatureCallOrCreation (FCCV): <[FeatureCallOrCreation](#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

VAL016\_Try (TRYV): <[Try](#Try)> is valid if and only if

VAL017\_Raise (RV): <[Raise](#Raise)> is valid if and only if [Expression](#Expression)

1. **SLang semantics:**

SEM001\_: ..

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

SEM\_\_\_: PredicatesList is evaluated using And and

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