**SLang reference manual. Version 0.80**

1. **SLang reserved word:**

|  |  |  |
| --- | --- | --- |
| ***#*** | ***Name*** | ***Kind(comment)*** |
| 1 | **abstract** | Unit/Routine characteristic. Bodyless unit member or objectless unit |
| 2 | **alias** | Unit/Routine characteristic. Alternative name of the unit or routine. |
| 3 | **as** | Unit level/Type. Another name or anchor |
| 4 | **break** | Statement. Exit the current scope. |
| 5 | **concurrent** | Type. It can be unit or attribute. |
| 6 | **const** | Attribute prefix. States for immutability of the attribute. |
| 7 | **deep** | Attribute prefix. Deep version of attribute immutability. |
| 8 | **do** | Statement. Start of the block. |
| 9 | **else** | Statement. Start of else block. |
| 10 | **elsif** | Statement. Start of the else if block |
| 11 | **end** | End of block or construction |
| 12 | **ensure** | Predicate. Routine post-condition clause start. |
| 13 | **extend** | Unit level. Used to support inheritance and unit extentsions. |
| 14 | **final** | Routine characteristic. Routine can not be overriden down in the inheritance hierarchy |
| 15 | **foreign** | Routine characteristic. Body of the routine is coded in 3rd party language |
| 16 | **if** | Statement. Conditional statement start. |
| 17 | **in** | Operator. Belongs to the range of values. |
| 18 | **init** | Routine characteristic. Start of the initialization procedure. |
| 19 | **is** | Definition of the initial value of an attribute. Check for a value of some expression |
| 20 | **new** | Statement/Expression. Creation of an object. May be skipped. |
| 21 | **old** | Expression. Value of some attribute before the routine start. To be used in post-conditions only. |
| 22 | **override** | Unit member characteristic. States that this attribute overrides all possible inherited versions. |
| 23 | **pure** | Routine characteristic. Routine is prohibited to write into unit attributes or read them. Must work only with its arguments |
| 24 | **raise** | Statement. Raise an exception with some object as a parameter. |
| 25 | **ref** | Type. States that an object will be of the reference nature |
| 26 | **require** | Predicate. Start of precondition clause of routine, unit or loop invariant |
| 27 | **return** | Statement/ Expression. Stopes the execution of the routine and returns result in case of function |
| 28 | **routine** | Type. Has 2 meanings – denotes the routine type after : or creates routine object from some routine in expressions |
| 29 | **safe** | Routine characteristic. Routine is prohibited to write into unit attributes. |
| 30 | ***super*** | Statement. In fact, that is a precursor call. To be replaced with the proper keyword. |
| 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 a module (singleton) at a unit or routine level. Allows top rename units as well. |
| 34 | **val** | Type. States that an object will be of value nature. Object itself not a reference to it. |
| 35 | **when** | Statement. Exception handling condition clause. |
| 36 | **while** | Statement. Loop condition clause. |

Below keywords are names of operations and to be processed by the compiler as operations but not keywords.In fact, the compiler treats them as just ordinary identifiers and when we have a name of the operation as “and then” or “or else” that will be the token – multiword token

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

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)[“**;**”]|[UnitRoutineDeclaration](#UnitRoutineDeclaration)}

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

(val-, sem-) InnerBlock: **do** [”**:**”[Label](#Label)]

[StatementsList](#StatementsList)

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

(val-, sem-) StandaloneRoutine:

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

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

(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** [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**|**abstract**|**extend**]

**unit** [Identifier](#Identifier) [[AliasName](#AliasName)] [[FormalGenerics](#FormalGenerics)]

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

[**is**]  
 {

[MemberSelection](#FeatureSelection)|

[ConstObjectsDeclaration](#ConstObjectsDeclaration)|

[MemberDeclaration](#FeatureDeclaration)|

[InheritedMemberOverriding](#InheritedFeatureOverriding)|

[UnitDeclaration](#UnitDeclaration)

}

[[InvariantBlock](#InvariantBlock)]  
 **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) [“extend” [Type](#Type) [“**init**” [[Signature](#Signature)]]]| [“**:**” [UnitType](#UnitType) | [RoutineType](#RoutineType)]

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

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

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

(val-, sem-) MemberDeclaration:

[“**{**” [**this**| [UnitTypeName](#UnitTypeName) {“**,**” [UnitTypeName](#UnitTypeName)} ] “**}**”] [**override**]

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

(val-, sem-) InitDeclaration:

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

[HyperBlock](#HyperBlock)

**end**

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

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

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

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

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

------------- what is “[**final** [Identifier](#Identifier)]” ????

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

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

------------- May be any sequence of characters started with some special symbols???

(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)

| [LocalEntityCreation](#LocalAttributeDeclaration)

| [IfCase](#IfCase)

| [Loop](#Loop)

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

| [MemberCallOrCreation](#FeatureCallOrCreation)

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

|[HyperBlock](#HyperBlock) **end**

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

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

(val-, sem-) HyperBlock :

[[RequireBlock](#RequireBlock)]

[InnerBlock](#InnerBlock)

[[EnsureBlock](#EnsureBlock)]

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

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

(val-, sem-) AttributeNamesList:

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

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

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

|

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

(val-, sem-) UnitAttributeDeclaration:

[AttributeNamesList](#AttributeNamesList) “**:**” [Type](#Type) [“**:=**” [[[[Arguments](#Arguments)] [HyperBlock](#HyperBlock) **end**](#TupleExpression)](#OldExpression) ]

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

(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:  
 [IfExpression](#IfExpession) | [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:

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

(val-, sem-) InlineLambdaExpression [: [](#EnsureBlock)**[pure](#EnsureBlock)**[|](#EnsureBlock)**[safe](#EnsureBlock)**[]](#EnsureBlock) **[routine](#EnsureBlock)** [[](#EnsureBlock)[[Arguments](#EnsureBlock)](#Arguments)[] [“](#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)|[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))|(**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)“.”**super**)|**super**|**this**|**return** [“**(**”[[ExpressionList](#ExpressionList)]”**)**”] {[CallChain](#CallChain)})

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

| ([**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)}

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

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

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

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

[[RequireBlock](#RequireBlock)]

[InnerBlock](#InnerBlock)

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

[[EnsureBlock](#EnsureBlock)]

**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)

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) } [ “**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 <[LocalEntityCreation](#LocalAttributeDeclaration)> 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): <[LocalEntityCreation](#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\_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\_: ..