**SLang syntax-validty-semantics manual. Version 0.99.6**

1. **SLang keywords:**

|  |  |  |
| --- | --- | --- |
| ***#*** | ***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 | **concurrent** | Type. It can be a unit or an attribute. |
| 4 | **const** | Unit-routine level. Import of constant objects |
| 5 | **do** | Statement. Start of the block. |
| 6 | **else** | Statement. Start of else block. |
| 7 | **elsif** | Statement. Start of the else if block |
| 8 | **end** | End of block or construction |
| 9 | **enum** | Unit level. Start of constant objects declaration section |
| 10 | **ensure** | Predicate. Routine post-condition clause start. |
| 11 | **extend** | Unit level. Used to support inheritance and unit extensions. |
| 12 | **final** | Member characteristic. Member can not be overridden down in the inheritance hierarchy. Also final can be applied to attribute of the unit to state finalization action |
| 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 started. 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. If no arguments then reraise the last exception occurred. |
| 23 | **ref** | Type. States that an object will be of the reference nature |
| 24 | **require** | Predicate. Start of precondition clause of routine, unit, or loop invariant |
| 25 | **return** | Statement/ Expression. Stopes the execution of the routine and returns result in case of function |
| 26 | **rigid** | Attribute prefix. A deep version of attribute immutability. |
| 27 | **rtn** | Type. Has 2 meanings – denotes the routine type after a colon (‘:’) or creates a routine object from some routine in expressions |
| 28 | **safe** | Routine characteristic. Routine is prohibited to write into unit attributes. |
| 29 | **select** | Unit level: select one version among several versions to resolve ambiguity in case of polymorphic assignment |
| 30 | **this** | Expression. Reference to the current object. |
| 31 | **unit** | Unit level. Start of the unit description. |
| 32 | **use** | Unit/Routine level. It states that some unit will be used as a module (singleton) at a unit or routine level. It allows renaming units as well. Unit level: give a new name to the inherited member  System level - import constants for this source |
| 33 | **val** | Type. States that an object will be of value nature. Object itself not a reference to it. |
| 34 | **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 an assignment into it. |
| 35 | **virtual** | Unit/Routine characteristic. Bodyless unit routine or objectless unit |
| 36 | **when** | Statement. Exception handling condition clause. |
| 37 | **while** | Statement. Loop condition clause. |

1. **SLang syntax: list of all syntax rules**

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

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

(val-, sem-) Context: **context**

[ProjectDsc](#ProjectDsc) | [Clusters](#Clusters)

**end**

ProjectDsc:

**target** [OS] (**exe** [[ContextName](#ContextName)] [exe\_type] [**init** exe\_entry]) | (**lib** [**dll**] [[ContextName](#ContextName)] [**init** lib\_entry {lib\_entry}])

[Clusters](#Clusters)

[**foreign** {[ContextName](#ContextName) [“:” {foreign\_option} **end**]} ]

Clusters:

[**use** {[ContextName](#ContextName) [“:” {use\_option} **end**]} ]

OS: Win32|Win64|Lin32|Lin64|Android|iOS|All

exe\_type: console|GUI|All

exe\_entry: UnitName|ProcedureName [“(“Signature”)”]

lib\_entry: RoutineName [“(“Signature”)”]

ContextName: [Identifier](#Identifier)| [StringConstant](#StringConstant)

(val-) UseDirective:

**use** (**const** [UnitTypeName](#UnitTypeName) {“**,**” [UnitTypeName](#UnitTypeName)}) | ([AttachedType](#AttachedType) **as** [Identifier](#Identifier)) [NewLine]

(val-) EnclosedUseDirective: [**use** [[EnclosedUseEement](#EnclosedUseEement) {“**,**” [EnclosedUseEement](#EnclosedUseEement)}]

[**const** [UnitTypeName](#UnitTypeName) {“**,**” [UnitTypeName](#UnitTypeName)}]] [NewLine]

(val-) EnclosedUseEement: [UnitTypeName](#UnitTypeName) [**as** [Identifier](#Identifier)]]

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

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

(val-) StandaloneRoutine: [**pure**|**safe**] [Identifier](#Identifier) [[FormalGenerics](#FormalGenerics)] [[Parameters](#Parameters)]

[ReturnType [Type](#Type)] [[EnclosedUseDirective](#EnclosedUseDirective)] [[RequireBlock](#RequireBlock)]

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

(val-, [sem](#SEM001_InnerBlock)) InnerBlock:

BlockStart[GroupStart [Identifier](#Identifier) {“,” [Identifier](#Identifier)} GroupEnd]

[StatementsList](#StatementsList)

[ [WhenClause](#WhenClause) {[WhenClause](#WhenClause)}

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

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

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

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

(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-) UnitDeclaration: ([**final**] [**ref**|**val**|**concurrent**])|[**virtual**]|[**extend**] **unit** [Identifier](#Identifier) [[AliasName](#AliasName)] [[FormalGenerics](#FormalGenerics)] [[InheritDirective](#InheritDirective)] [[EnclosedUseDirective](#EnclosedUseDirective)]

[[MemberSelection]](#MemberSelection)

[[InheritedMemberOverriding]](#InheritedMemberOverriding)

[[InitProcedureInheritance](#InitProcedureInheritance)]

[[ConstObjectsDeclaration](#ConstObjectsDeclaration)]

{ ( [MemberVisibility](#MemberVisibility) “:” {[MemberDeclaration](#FeatureDeclaration)}) | [MemberDeclaration](#FeatureDeclaration) }

[[InvariantBlock](#InvariantBlock)]

BlockEnd

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

(val-) Parent: [UnitTypeName](#UnitTypeName) | (“**~**” [UnitTypeName](#UnitTypeName) [“(”[MemberName](#MemberName){“,”[MemberName](#MemberName)}“)”])

(val-) MemberName: [Identifier](#Identifier)|([RoutineName](#RoutineName) [[Signature](#Signature)])

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

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

(val-) MemberSelection: **select** [MemberName](#MemberName) {“**,**”[MemberName](#MemberName)}

(val-) InheritedMemberOverriding:

**override** [UnitTypeName](#UnitTypeName)”**.**”[MemberName](#MemberName){“,” [UnitTypeName](#UnitTypeName)”**.**”[MemberName](#MemberName)}

(val-) InitProcedureInheritance:

**init** [UnitTypeName](#UnitTypeName)[[Signature](#Signature)] {“**,**” [UnitTypeName](#UnitTypeName)[[Signature](#Signature)]}

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

(val-) MemberDeclaration: [[MemberVisibility](#MemberVisibility)]

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

(val-) InitDeclaration:

**init** [[“**:=**”][Parameters](#Parameters)] [[EnclosedUseDirective](#EnclosedUseDirective)] [[RequireBlock](#RequireBlock)]

([InnerBlock](#InnerBlock) [[EnsureBlock](#EnsureBlock)] BlockEnd)|(**foreign** [[EnsureBlock](#EnsureBlock) BlockEnd])

(val-) UnitRoutineDeclaration:

[**pure**|**safe**] [RoutineName](#RoutineName) [**final** [Identifier](#Identifier)] [[“**:=**”][Parameters](#Parameters)] [ReturnType [Type](#Type)] [[EnclosedUseDirective](#EnclosedUseDirective)] [[RequireBlock](#RequireBlock)]

(([InnerBlock](#InnerBlock)) [[EnsureBlock](#EnsureBlock)] BlockEnd)

|

((**virtual**|**foreign**|(“**=>**”[Expression](#Expression)))[[EnsureBlock](#EnsureBlock) BlockEnd])

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

(val-) AliasName : **alias** ([Identifier](#Identifier)|“*and then*”|“*or else*”)

(val-) OperatorName : [OperatorSign](#OperatorSign) [[OperatorSign](#OperatorSign)]

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

(val-, sem-) ConstObjectsDeclaration: **enum** [ [ConstObject](#ConstObject) { “**,**” [ConstObject](#ConstObject)} ] BlockEnd

(val-, sem-) ConstObject :

(

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

[ [“{”[OperatorName](#OperatorName) [ConstantExpression](#ConstantExpression) “}”] “**..**” ([Constant](#Constant) | ([Idenitifer](#Identifier) [“.”**init**] [ [Arguments](#Arguments) ])) ]

)

|

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

(val-, sem-) RegularExpression:

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

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

[Assignment](#Assignment)

| [LocalAttributeCreation](#LocalAttributeCreation)

| [MemberCall](#MemberCall)

| [NewStatement](#NewStatement)

| [IfCase](#IfCase)

| [Loop](#Loop)

| [Detach](#Detach)

|[Return](#Return)

|[HyperBlock](#HyperBlock)

| [Raise](#Raise)

(val-, sem-) Detach: **?** [Identifier](#Identifier) [NewLine]

(val-, sem-) Raise: **raise** [[Expression](#Expression)] [NewLine]

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

(val-, sem-) HyperBlock: [[RequireBlock](#RequireBlock)] [InnerBlock](#InnerBlock) [[EnsureBlock](#EnsureBlock)] BlockEnd

([val](#VAL007_Assignment), sem-) Assignment: [Writable](#Writable) “**:=**” [Expression](#Expression) [NewLine]

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

Examples: (a.x, b(x).y.z, c) := (E1, E2, E3) a := expr a.b.c := expr foo(…).y := expr

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

([LocalAttributeNamesList](#LocalAttributeNamesList) ([“**:**” [Type](#Type)] **is** [Expression](#Expression) [NewLine])|(“**:**” “**?**” [AttachedType](#AttachedType)))

|(“**(**“ [LocalAttributeNamesList](#LocalAttributeNamesList) “**)**” **is** [Expression](#Expression) [NewLine])

(val-, sem-) LocalAttributeNamesList: [**var**|**rigid**] [Identifier](#Identifier) {“**,**”[**var**|**rigid**] [Identifier](#Identifier)}

(val-, sem-) UnitAttributeDeclaration:

(

( [UnitAttributeNamesList](#UnitAttributeNamesList) “:” [Type](#Type))

|

( [**const**|**rigid**] [Identifier](#Identifier) [“:” [AttachedType](#AttachedType)] **is** [ConstantExpression](#ConstantExpression) [NewLine])

|

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

) [**final** (“**=>**”[Statement](#Statement)[NewLine] ) | ([InnerBlock](#InnerBlock) [[EnsureBlock](#EnsureBlock)] BlockEnd)]

(val-) UnitAttributeNamesList:

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

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

(val-, sem-) ConstantExpression:

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

(val-, sem-) Expression:

[IfExpression](#IfExpession) | [[MemberCall](#MemberCall)](#FeatureCallOrCreation) | [NewExpression](#NewExpression) | [Expression](#Expression) [Operator](#Operator) [Expression](#Expression)

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

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

(val-, sem-) LambdaExpression: (**rtn** [Identifier](#Identifier) [[Signature](#Signature)])|[InlineLambdaExpression](#InlineLambdaExpression)

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

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

(val-, sem-) RangeExpression:

[Expression](#Expression) [“{”[OperatorName](#OperatorName) [ConstantExpression](#ConstantExpression)“}”] “**..**”[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)| [AnonymousUnitType](#AnonymousUnitType))

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

(val-, sem-) Constant: [[UnitTypeName](#UnitTypeName) “.”]

( [StringConstant |](#StringConstant) [CharacterConstant |](#CharacterConstant) [IntegerConstant |](#IntegerConstant) [RealConstant |](#RealConstant) [BooleanConstant](#BooleanConstant) | [BitConstant](#BitConstant) | [Identifier](#Identifier) )

(val-, sem-) IfExpression:

**if** [Expression](#Expression) (**is** [IfBodyExpression](#IfBodyExpression))|( BlockStart[Expression](#Expression))  
{**elsif** [Expression](#Expression) (**is** [IfBodyExpression](#IfBodyExpression))|( BlockStart[Expression](#Expression))}  
**else** [Expression](#Expression) “}”**Cmod**

(val-, sem-) IfBodyExpression:

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

(val-, sem-) MemberCall: [WritableCall](#WritableCall)|(**init** [[Arguments](#Arguments)])

(val-, sem-) WritableCall:

(([Identifier](#Identifier)|**return**|**this**) [“.”([Identifier](#Identifier)|[OperatorName](#OperatorName))])

|**old** [GroupStart [UnitTypeName](#UnitTypeName) Groupend]

[[Arguments](#Arguments)] {[CallChain](#CallChain)}

(val-, sem-) NewStatement:

(**new** [GroupStart[UnitType](#UnitType) GroupEnd] **return**)

|

([**new**] [GroupStart [UnitType](#UnitType) GroupEnd] [Identifier](#Identifier))

[“.”**init**] [ [Arguments](#Arguments) ]

(val-, sem-) NewExpression: [**new**] [UnitType](#UnitType) [“.”**init**] [ [Arguments](#Arguments) ]

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

(val-, sem-) Arguments: “**(**” [[ExpressionList](#ExpressionList)] ”**)**”

(val-, sem-) ExpressionList:

[GroupStart [UnitType](#UnitType) GroupEnd] [Expression](#Expression)

{“**,**” [GroupStart [UnitType](#UnitType) GroupStart] [Expression](#Expression)}

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

**if** [Expression](#Expression) (**is** [IfBody](#IfBody))|(BlockStart[[StatementsList](#StatementsList)])

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

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

(val-, sem-) IfBody:

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

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

(val-, sem-) ValueAlternative:

[Expression](#Expression) ([[GroupStart [OperatorName](#OperatorName) [ConstantExpression](#ConstantExpression) GroupEnd] “**..**”[Expression](#Expression) ] | {“**|**”[Expression](#Expression)} )

(val-) MemberDescription:

([**rtn**] [RoutineName](#RoutineName)[[Signature](#Signature)])|([Idenitifer](#Identifier){“,”[Idenitifer](#Identifier)} ”**:**” [UnitType](#UnitType))

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

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

|

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

[[EnsureBlock](#EnsureBlock)] BlockEnd

(val-) Type: [”**?**”] [AttachedType](#AttachedType)

(val-) AttachedType: [UnitType](#UnitType)|[AnchorType](#AnchorType)|[MultiType](#MultiType)|[TupleType](#TupleType)|[RangeType](#RangeType)|[RoutineType](#RoutineType)|[AnonymousUnitType](#AnonymousUnitType)

(val-) AnonymousUnitType: **unit** [MemberDesciption](#memberDescription) {[“;”] [MemberDesciption](#memberDescription)} BlockEnd

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

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

(val-) RangeType:

([ConstantExpression](#ConstantExpression) [GroupStart[OperatorName](#OperatorName) [ConstantExpression](#ConstantExpression) GroupEnd] “**..**”[ConstantExpression](#ConstantExpression))

|

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

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

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

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

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

(val-) UnitTypeName:

[Identifier](#Identifier)

[GenericsStart([Type](#Type)|[ConstantExpression](#ConstantExpression)) {“**,**” ( [Type](#Type)|[ConstantExpression](#ConstantExpression))} GenericsEnd ]

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

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

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

ReturnType: “**:**”|”**->**”

GenericsStart: “**[**“|”**<**”**Cmod**

GenericsEnd: “**]**”|”**>**”**Cmod**

BlockStart: **do**|”**{**”**Cmod**

BlockEnd: **end**|”**}**”**Cmod**

GroupStart: “**{**”|”**[**“**Cmod**

GroupEnd: “**}**”|”**]**“**Cmod**

NewLine: “**;**”|*newLine*

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

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

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

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

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

BitConstant:

[ “**+**”|”**-**“ ] (“0”|”1“ {“0”|”1“} [“**B**”|”**b**”]) | (“0” ( “**b**”|“**B**”) “0”|”1“{“0”|”1“})

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

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

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

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

Symbol : ASCII symbol 0..255

UnicodeSymbol: … ‘\u’ ‘\U’

1. **SLang validity: list of all validity rules**

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)> are 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 the 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 provided and then valid and <[Return](#Return)> is in the body of the function and type of the Expression conforms to the type of the function.

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

VR\_001: Compilation: {[CompilationUnit](#CompilationUnit)} is valid if and only if every [CompilationUnit](#CompilationUnit) if provided is valid

CompilationUnit: {[UseDirective](#UseDirective)} ([AnonymousRoutine](#AnonymousRoutine)|[StandaloneRoutine](#StandaloneRoutine)|[UnitDeclaration](#UnitDeclaration)) is valid if and only if

UseDirective: **use** (**const** [UnitTypeName](#UnitTypeName) {“**,**” [UnitTypeName](#UnitTypeName)}) | ([AttachedType](#AttachedType) **as** [Identifier](#Identifier)) is valid if and only if

EnclosedUseDirective: [**use** [[EnclosedUseEement](#EnclosedUseEement) {“**,**” [EnclosedUseEement](#EnclosedUseEement)}]

[**const** [UnitTypeName](#UnitTypeName) {“**,**” [UnitTypeName](#UnitTypeName)}]] is valid if and only if

EnclosedUseEement: [UnitTypeName](#UnitTypeName) [**as** [Identifier](#Identifier)]] is valid if and only if

AnonymousRoutine : [StatementsList](#StatementsList) is valid if and only if

StatementsList: { [Statement](#Statement)[“**;**”]} is valid if and only if

StandaloneRoutine: [**pure**|**safe**] [Identifier](#Identifier) [[FormalGenerics](#FormalGenerics)] [[Parameters](#Parameters)]

[“**:**” [Type](#Type)] [[EnclosedUseDirective](#EnclosedUseDirective)] [[RequireBlock](#RequireBlock)]

([InnerBlock](#InnerBlock) [[EnsureBlock](#EnsureBlock)] **end**)|(((“**=>**”[Expression](#Expression))|**foreign**) [[EnsureBlock](#EnsureBlock) **end**]) is valid if and only if

InnerBlock: **do** [”**:**”[Label](#Label)][“{”[Identifier](#Identifier) {“,” [Identifier](#Identifier)} “}”]

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

[**else** [[StatementsList](#StatementsList)]]] is valid if and only if

WhenClause: **when** [[Identifier](#Identifier)**:**][UnitType](#UnitType) **do** [StatementsList](#StatementsList) is valid if and only if

Parameters: “**(**”[“**:=**”][Parameter](#Parameter){”**;**” [Parameter](#Parameter)}“**)**” is valid if and only if

Parameter: ([[**var**] [Identifier](#Identifier){“**,**” [**var**] [Identifier](#Identifier)} “**:**” [Type](#Type))|([Identifier](#Identifier) “**is**” [Expression](#Expression)|(“**as**” [Identifier](#Identifier))) is valid if and only if

RequireBlock : **require** [PredicatesList](#PredicatesList) is valid if and only if

EnsureBlock : **ensure** [PredicatesList](#PredicatesList) is valid if and only if

InvariantBlock : **require** [PredicatesList](#PredicatesList) is valid if and only if

PredicatesList : [[Predicate](#Predicate) {[”**;**”] [Predicate](#Predicate)}] is valid if and only if

Predicate : [BooleanExpression](#BooleanExpression) [[DocumentingComment](#DocumentingComment)] is valid if and only if

UnitDeclaration: ([**final**] [**ref**|**val**|**concurrent**])|[**virtual**]|[**extend**] **unit** [Identifier](#Identifier) [[AliasName](#AliasName)] [[FormalGenerics](#FormalGenerics)] [[InheritDirective](#InheritDirective)] [[EnclosedUseDirective](#EnclosedUseDirective)]

[[MemberSelection]](#MemberSelection)

[[InheritedMemberOverriding]](#InheritedMemberOverriding)

[[InitProcedureInheritance](#InitProcedureInheritance)]

[[ConstObjectsDeclaration](#ConstObjectsDeclaration)]

{ ( [MemberVisibility](#MemberVisibility) “:” {[MemberDeclaration](#FeatureDeclaration)}) | [MemberDeclaration](#FeatureDeclaration) }

[[InvariantBlock](#InvariantBlock)]

**end** is valid if and only if

InheritDirective: **extend** [Parent](#Parent) {“,” [Parent](#Parent)} is valid if and only if

Parent: [UnitTypeName](#UnitTypeName) | (“**~**” [UnitTypeName](#UnitTypeName) [“(”[MemberName](#MemberName){“,”[MemberName](#MemberName)}“)”]) is valid if and only if

MemberName: [Identifier](#Identifier)|([RoutineName](#RoutineName) [[Signature](#Signature)]) is valid if and only if

FormalGenerics: “**[**”[FormalGeneric](#FormalGeneric) {“**,**” [FormalGeneric](#FormalGeneric)}“**]**” is valid if and only if

FormalGeneric: [Identifier](#Identifier) ([“**extend**” [UnitTypeName](#UnitTypeName)] [“**init**” [[Signature](#Signature)]])| [“**:**” ([UnitType](#UnitType) | [RoutineType](#RoutineType)] is valid if and only if

MemberSelection: **select** [MemberName](#MemberName) {“**,**”[MemberName](#MemberName)} is valid if and only if

InheritedMemberOverriding: **override** [UnitTypeName](#UnitTypeName)”**.**”[MemberName](#MemberName){“,” [UnitTypeName](#UnitTypeName)”**.**”[MemberName](#MemberName)} is valid if and only if

InitProcedureInheritance:**init** [UnitTypeName](#UnitTypeName)[[Signature](#Signature)] {“**,**” [UnitTypeName](#UnitTypeName)[[Signature](#Signature)]} is valid if and only if

MemberVisibility: “**{**” [**this**| [UnitTypeName](#UnitTypeName) {“**,**” [UnitTypeName](#UnitTypeName)} ] “**}**”is valid if and only if

MemberDeclaration: [[MemberVisibility](#MemberVisibility)] ([**override**] [**final**] [UnitAttribiteDeclaration](#UnitAttributeDeclaration)|[UnitRoutineDeclaration](#UnitRoutineDeclaration)) | [InitDeclaration](#InitDeclaration) is valid if and only if

InitDeclaration: **init** [[“**:=**”][Parameters](#Parameters)] [[EnclosedUseDirective](#EnclosedUseDirective)] [[RequireBlock](#RequireBlock)]

([InnerBlock](#InnerBlock) [[EnsureBlock](#EnsureBlock)] **end**)|(**foreign** [[EnsureBlock](#EnsureBlock) **end**]) is valid if and only if

UnitRoutineDeclaration: [**pure**|**safe**] [RoutineName](#RoutineName) [**final** [Identifier](#Identifier)] [[“**:=**”][Parameters](#Parameters)] [“**:**” [Type](#Type)] [[EnclosedUseDirective](#EnclosedUseDirective)] [[RequireBlock](#RequireBlock)] (([InnerBlock](#InnerBlock)) [[EnsureBlock](#EnsureBlock)] **end**)|((**virtual**|**foreign**|(“**=>**”[Expression](#Expression))) [[EnsureBlock](#EnsureBlock) **end**]) is valid if and only if

RoutineName : [Identifier](#Identifier) |“**()**”|“**:=**”|([OperatorName](#OperatorName) [[AliasName](#AliasName)]) is valid if and only if

AliasName : **alias** ([Identifier](#Identifier)|“*and then*”|“*or else*”) is valid if and only if

OperatorName : [OperatorSign](#OperatorSign) [[OperatorSign](#OperatorSign)] is valid if and only if

OperatorSign : “**^**” | “**\***” | “**/**” | “**\**” | “**=**” | “**+**” | “**-**“ |”**<**” | ”**>**” | ”**&**” | “**|**”

ConstObjectsDeclaration: **enum** [ [ConstObject](#ConstObject) { “**,**” [ConstObject](#ConstObject)} ] **end** is valid if and only if

ConstObject :

(

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

[ [“{”[OperatorName](#OperatorName) [ConstantExpression](#ConstantExpression) “}”] “**..**” ([Constant](#Constant) | ([Idenitifer](#Identifier) [“.”**init**] [ [Arguments](#Arguments) ])) ]

)

|

(“{” [RegularExpression](#RegularExpression) “}” [IntegerConstant](#IntegerConstant) [“+”]) | is valid if and only if

RegularExpression: [Constant](#Constant) ({“**|**”[Constant](#Constant)}) | (“**|**””**..**” [Constant](#Constant)) is valid if and only if

Statement:[Assignment](#Assignment)| [LocalAttributeCreation](#LocalAttributeCreation)| [MemberCall](#MemberCall) | [NewStatement](#NewStatement)| [IfCase](#IfCase)| [Loop](#Loop)| [Break](#Break)

| [Detach](#Detach)|[Return](#Return)|[HyperBlock](#HyperBlock)| [Raise](#Raise) is valid if and only if

Detach: **?** [Identifier](#Identifier) is valid if and only if

Raise: **raise** [[Expression](#Expression)] is valid if and only if

Return: **return** [[Expression](#Expression)] is valid if and only if

Break: **break** [“:”[Label](#Label)] is valid if and only if

Label : [Identifier](#Identifier) is valid if and only if

HyperBlock: [[RequireBlock](#RequireBlock)] [InnerBlock](#InnerBlock) [[EnsureBlock](#EnsureBlock)] **end** is valid if and only if

Assignment: [Writable](#Writable) “**:=**” [Expression](#Expression) is valid if and only if

Writable: [WritableCall](#WritableCall) | (“**(**”[WritableCall](#WritableCall) {“**,**” [WritableCall](#WritableCall) } “**)**”) is valid if and only if

Examples: (a.x, b(x).y.z, c) := (E1, E2, E3) a := expr a.b.c := expr foo(…).y := expr

LocalAttributeCreation:

([LocalAttributeNamesList](#LocalAttributeNamesList) ([“**:**” [Type](#Type)] **is** [Expression](#Expression) )|(“**:**” “**?**” [AttachedType](#AttachedType)))

|(“**(**“ [LocalAttributeNamesList](#LocalAttributeNamesList) “**)**” **is** [Expression](#Expression) ) is valid if and only if

LocalAttributeNamesList: [**var**|**rigid**] [Identifier](#Identifier) {“**,**”[**var**|**rigid**] [Identifier](#Identifier)} is valid if and only if

UnitAttributeDeclaration:

(

( [UnitAttributeNamesList](#UnitAttributeNamesList) “:” [Type](#Type))

|

( [**const**|**rigid**] [Identifier](#Identifier) [“:” [AttachedType](#AttachedType)] **is** [ConstantExpression](#ConstantExpression) [“**;**”|*newLine*])

|

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

) [**final** (“**=>**”[Statement](#Statement) ) | ([InnerBlock](#InnerBlock) [[EnsureBlock](#EnsureBlock)] **end**)] is valid if and only if

UnitAttributeNamesList:[**const** | **rigid**] [Identifier](#Identifier) {“**,**”[**const** | **rigid**] [Identifier](#Identifier)} is valid if and only if

BooleanExpression: [Expression](#Expression) is valid if and only if

ConstantExpression:([Identifier](#Identifier) {“**.**” [Identifier](#Identifier)}) | [Constant](#Constant) [[Operator](#Operator) [ConstantExpression](#ConstantExpression)] is valid if and only if

Expression:

[IfExpression](#IfExpession) | [[MemberCall](#MemberCall)](#FeatureCallOrCreation) | [NewExpression](#NewExpression) | [Expression](#Expression) [Operator](#Operator) [Expression](#Expression)

| [Operator](#Operator) [Expression](#Expression) | [Constant | [TypeOfExpression](#TypeOfExpression) | [OldExpression](#OldExpression)](#Constant)| [RangeExpression |](#RangeExpression) [LambdaExpression](#LambdaExpression) | [TupleExpression |](#TupleExpression) [RefExpression](#RefExpression)| “**(**”[Expression](#Expression)“**)**”{[CallChain](#CallChain)} is valid if and only if

[RefExpression:](#TupleExpression) **ref** [Expression](#Expression) is valid if and only if

LambdaExpression: (**rtn** [Identifier](#Identifier) [[Signature](#Signature)])|[InlineLambdaExpression](#InlineLambdaExpression) is valid if and only if

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)**[end](#EnsureBlock)**[)|(](#EnsureBlock)**[foreign](#EnsureBlock)** [[[EnsureBlock](#EnsureBlock)]](#EnsureBlock) **[end](#EnsureBlock)**[])|(“](#EnsureBlock)**[=>](#EnsureBlock)**[”](#EnsureBlock)[[Expression](#EnsureBlock)](#Expression)[)](#EnsureBlock) is valid if and only if

RangeExpression:

[Expression](#Expression) [“{”[OperatorName](#OperatorName) [ConstantExpression](#ConstantExpression)“}”] “**..**”[Expression](#Expression) is valid if and only if

OldExpression [: **old** [Expression](#Expression) is valid if and only if](#TupleExpression)TupleExpression: “**(**”[[TupleElement](#TupleElement) {“**,**” [TupleElement](#TupleElement)}]“**)**” is valid if and only if

TupleElement: [Expression](#Expression)| [Parameter](#Parameter) is valid if and only if

TypeOfExpression: [Expression](#Expression) **is** (**“?”**| [UnitType](#UnitType)| [AnonymousUnitType](#AnonymousUnitType)) is valid if and only if

Operator: [OperatorName](#OperatorName)|**in** is valid if and only if

Constant: [[UnitTypeName](#UnitTypeName) “.”]

( [StringConstant |](#StringConstant) [CharacterConstant |](#CharacterConstant) [IntegerConstant |](#IntegerConstant) [RealConstant |](#RealConstant) [BooleanConstant](#BooleanConstant) | [BitConstant](#BitConstant) | [Identifier](#Identifier) ) is valid if and only if

IfExpression:

**if** [Expression](#Expression) (**is** [IfBodyExpression](#IfBodyExpression))|(**do** [Expression](#Expression))  
{**elsif** [Expression](#Expression) (**is** [IfBodyExpression](#IfBodyExpression))|(**do** [Expression](#Expression))}  
**else** [Expression](#Expression) is valid if and only if

IfBodyExpression:

“:”[ValueAlternative](#ValueAlternative)“:”[Expression](#Expression) {“:”[ValueAlternative](#ValueAlternative)“:”[Expression](#Expression)} is valid if and only if

MemberCall: [WritableCall](#WritableCall)|(**init** [[Arguments](#Arguments)]) is valid if and only if

WritableCall:

((([Identifier](#Identifier)|**return**|**this**) [“.”([Identifier](#Identifier)|[OperatorName](#OperatorName))])|**old** [“**{**”[UnitTypeName](#UnitTypeName)”**}**”])

[[Arguments](#Arguments)] {[CallChain](#CallChain)} is valid if and only if

NewStatement:

(**new** [“**{**” UnitType “**}**”] **return** ) | (**[new]** [“**{**” UnitType “**}**”] [Identifier](#Identifier) )

([“.”**init**] [ [Arguments](#Arguments) ]) is valid if and only if

NewExpression: [**new]** UnitType [“.”**init**] [ [Arguments](#Arguments) ] is valid if and only if

CallChain: “**.**”([Identifier](#Identifier)|[OperatorName](#OperatorName)) [ [Arguments](#Arguments) ] is valid if and only if

Arguments: “**(**” [[ExpressionList](#ExpressionList)] ”**)**” is valid if and only if

ExpressionList: [ “{”[UnitType](#UnitType) “}”] [Expression](#Expression){“**,**” [ “{”[UnitType](#UnitType) “}”] [Expression](#Expression)} is valid if and only if

IfCase:

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

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

[**else** [ [StatementsList](#StatementsList) ]]  
**end** is valid if and only if

IfBody:

“**:**”[ValueAlternative](#ValueAlternative)“**:**”[StatementsList](#StatementsList) {“**:**”[ValueAlternative](#ValueAlternative)“**:**”[StatementsList](#StatementsList)} is valid if and only if

ValueAlternative:

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

{“,”

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

} is valid if and only if

MemberDescription:([**rtn**] [RoutineName](#RoutineName)[[Signature](#Signature)])|([Idenitifer](#Identifier){“,”[Idenitifer](#Identifier)} ”**:**” [UnitType](#UnitType)) is valid if and only if

Loop:

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

|

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

[[EnsureBlock](#EnsureBlock)] **end** is valid if and only if

Type: [”**?**”] [AttachedType](#AttachedType) is valid if and only if

AttachedType: [UnitType](#UnitType)|[AnchorType](#AnchorType)|[MultiType](#MultiType)|[TupleType](#TupleType)|[RangeType](#RangeType)|[RoutineType](#RoutineType)|[AnonymousUnitType](#AnonymousUnitType) is valid if and only if

AnonymousUnitType: “**unit**” [MemberDesciption](#memberDescription) {[“;”] [MemberDesciption](#memberDescription)} “**end**” is valid if and only if

RoutineType: **rtn** [[Signature](#Signature)] is valid if and only if

Signature: (“**(**”[[Type](#Type) {“**,**” [Type](#Type)}]“**)**”[“**:**” [Type](#Type)])| (“**:**” [Type](#Type)) is valid if and only if

RangeType:

([ConstantExpression](#ConstantExpression) [“{”[OperatorName](#OperatorName) [ConstantExpression](#ConstantExpression)“}”] “**..**”[ConstantExpression](#ConstantExpression))

|

([ConstantExpression](#ConstantExpression) {“**|**” [ConstantExpression](#ConstantExpression)}) is valid if and only if

AnchorType: **as** (**this**|([Identifier](#Identifier) [[Signature](#Signature)])) is valid if and only if

MultiType: [UnitType](#UnitType) {“**|**”[UnitType](#UnitType)} is valid if and only if

TupleType: “**(**”[[TupleField](#TupleField) {“**,**”|”**;**” [TupleField](#TupleField)}]“**)**” is valid if and only if

TupleField: [[Identifier](#Identifier) {“**,**” [Identifier](#Identifier)}“**:**”] [UnitType](#UnitType) is valid if and only if

UnitTypeName:

[Identifier](#Identifier) [“**[**“([Type](#Type)|[ConstantExpression](#ConstantExpression)) {“**,**” ( [Type](#Type)|[ConstantExpression](#ConstantExpression))}“**]**” ] is valid if and only if

UnitType: [**ref**|**val**|**concurrent**] [UnitTypeName](#UnitTypeName) is valid if and only if

1. **SLang semantics: list of all behavioral patterns**

**Remove!** SEM001\_InnerBlock: **do** [“{”[Identifier](#Identifier) {“,” [Identifier](#Identifier)} “}”] - do not check invariants for these objects within the block

SEM002\_UnitRoutineDeclaration: **final** [Identifier](#Identifier) allows calling this version from any descendant unit

SEM003\_AnonymousRoutine: Identical to [SEM004\_StatementsList](#SEM004_StatementsList)

SEM004\_StatementsList: { [Statement](#Statement)[“**;**”]} All statements of the list are being executed by the processing element one by one according to [SEM006\_Statement](#SEM006_Statement) unless some may lead to an exception or leave the sequence (return)

SEM005\_InnerBlock: if the list of identifiers “{”[Identifier](#Identifier) {“,” [Identifier](#Identifier)} “}” is provided then for these identifiers calls invariants are not checked within the block. StatemntList is executed according to [SEM004\_StatementsList](#SEM004_StatementsList), if when clauses are provided and execution of StatemntList leads to some exception the check if this exception can be handled by one of when clauses is performed if such intercepting clause is found then when clause body is executed according to [SEM005\_WhenClause](#SEM005_WhenClause) otherwise if the else part is in place it is executed according to [SEM004\_StatementsList](#SEM004_StatementsList) otherwise if no else part present then exception block execution failure exception is raised.

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

[StatementsList](#StatementsList)

[ [WhenClause](#WhenClause) {[WhenClause](#WhenClause)}

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

SEM005\_WhenClause: **when** [[Identifier](#Identifier)**:**][UnitType](#UnitType) **do** [StatementsList](#StatementsList) if the type of exception conforms to the type of the when clause ([UnitType](#UnitType)) then do part is being executed according to [SEM004\_StatementsList](#SEM004_StatementsList) and exception is treated as handled. If the identifier is provided then the current exception object is available in the body of when clause handler using the identifier name

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

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

SEM031\_RequireBlock: **require** [PredicatesList](#PredicatesList) this clause is evaluated before any routine call according to [SEM034\_PredicatesList](#SEM034_PredicatesList) and if some predicate is evaluated to false exception object will be of type precondition violation

SEM032\_EnsureBlock: **ensure** [PredicatesList](#PredicatesList) this clause is evaluated after any successful routine call according to [SEM034\_PredicatesList](#SEM034_PredicatesList) and if some predicate is evaluated to false exception object will be of type postcondition violation

SEM033\_InvariantBlock: **require** [PredicatesList](#PredicatesList) this clause is evaluated after any successful routine call and then after any successful execution of [SEM032\_EnsureBlock](#SEM032_EnsureBlock) if present according to [SEM034\_PredicatesList](#SEM034_PredicatesList) and if some predicate is evaluated to false exception object will be of type unit invariant violation

SEM034\_PredicatesList: [[Predicate](#Predicate) {[”**;**”] [Predicate](#Predicate)}] each predicate of the list will be evaluated according to [SEM035\_Predicate](#SEM035_Predicate) until the first one which raises an exception. If all predicates were evaluated as true then execution continues

SEM035\_Predicate: [BooleanExpression](#BooleanExpression) [[DocumentingComment](#DocumentingComment)] Boolean expression is evaluated according to and if it was evaluated to false then exception is generated in case of [DocumentingComment](#DocumentingComment) provided passing it as an argument for exception object creation

SEM006\_Statement: its execution leads to the execution of one of the particular statements below

[Assignment](#Assignment)| [LocalAttributeCreation](#LocalAttributeCreation)| [MemberCall](#MemberCall) | [NewStatement](#NewStatement)| [IfCase](#IfCase)| [Loop](#Loop)| [Break](#Break)

| [Detach](#Detach)|[Return](#Return)|[HyperBlock](#HyperBlock)| [Raise](#Raise)

SEM007\_Detach: **?** [Identifier](#Identifier) [“**;**”|*newLine*]

SEM008\_Raise: **raise** [[Expression](#Expression)] [“**;**”|*newLine*]

SEM009\_Return: **return** [[Expression](#Expression)] [“**;**”|*newLine*]

SEM010\_Break: **break** [“:”[Label](#Label)]

SEM011\_HyperBlock: [[RequireBlock](#RequireBlock)] [InnerBlock](#InnerBlock) [[EnsureBlock](#EnsureBlock)] **end**

SEM012\_Assignment: [Writable](#Writable) “**:=**” [Expression](#Expression) [“**;**”|*newLine*]

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

Examples: (a.x, b(x).y.z, c) := (E1, E2, E3) a := expr a.b.c := expr foo(…).y := expr

SEM013\_LocalAttributeCreation:

([LocalAttributeNamesList](#LocalAttributeNamesList) ([“**:**” [Type](#Type)] **is** [Expression](#Expression) [“**;**”|*newLine*])|(“**:**” “**?**” [AttachedType](#AttachedType)))

|(“**(**“ [LocalAttributeNamesList](#LocalAttributeNamesList) “**)**” **is** [Expression](#Expression) [“**;**”|*newLine*])

(val-, sem-) LocalAttributeNamesList: [**var**|**rigid**] [Identifier](#Identifier) {“**,**”[**var**|**rigid**] [Identifier](#Identifier)}

(val-, sem-) UnitAttributeDeclaration:

(

( [UnitAttributeNamesList](#UnitAttributeNamesList) “:” [Type](#Type))

|

( [**const**|**rigid**] [Identifier](#Identifier) [“:” [AttachedType](#AttachedType)] **is** [ConstantExpression](#ConstantExpression) [“**;**”|*newLine*])

|

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

) [**final** (“**=>**”[Statement](#Statement)[“**;**”|*newLine*] ) | ([InnerBlock](#InnerBlock) [[EnsureBlock](#EnsureBlock)] **end**)]

(val-, sem-) UnitAttributeNamesList:

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

SEM014\_BooleanExpression: [Expression](#Expression)

SEM015\_ConstantExpression:

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

SEM016\_Expression:

[IfExpression](#IfExpession) | [[MemberCall](#MemberCall)](#FeatureCallOrCreation) | [NewExpression](#NewExpression) | [Expression](#Expression) [Operator](#Operator) [Expression](#Expression)

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

SEM017\_[RefExpression:](#TupleExpression) **ref** [Expression](#Expression)

SEM018\_LambdaExpression: (**rtn** [Identifier](#Identifier) [[Signature](#Signature)])|[InlineLambdaExpression](#InlineLambdaExpression)

SEM019\_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)**[end](#EnsureBlock)**[)|(](#EnsureBlock)**[foreign](#EnsureBlock)** [[[EnsureBlock](#EnsureBlock)]](#EnsureBlock) **[end](#EnsureBlock)**[])|(“](#EnsureBlock)**[=>](#EnsureBlock)**[”](#EnsureBlock)[[Expression](#EnsureBlock)](#Expression)[)](#EnsureBlock)

SEM020\_RangeExpression:

[Expression](#Expression) [“{”[OperatorName](#OperatorName) [ConstantExpression](#ConstantExpression)“}”] “**..**”[Expression](#Expression)

SEM021\_OldExpression [: **old** [Expression](#Expression)](#TupleExpression)SEM022\_TupleExpression: “**(**”[[TupleElement](#TupleElement) {“**,**” [TupleElement](#TupleElement)}]“**)**”

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

SEM023\_TypeOfExpression: [Expression](#Expression) **is** (**“?”**| [UnitType](#UnitType)| [AnonymousUnitType](#AnonymousUnitType))

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

(val-, sem-) Constant: [[UnitTypeName](#UnitTypeName) “.”]

( [StringConstant |](#StringConstant) [CharacterConstant |](#CharacterConstant) [IntegerConstant |](#IntegerConstant) [RealConstant |](#RealConstant) [BooleanConstant](#BooleanConstant) | [BitConstant](#BitConstant) | [Identifier](#Identifier) )

SEM024\_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)}

SEM025\_MemberCall: [WritableCall](#WritableCall)|(**init** [[Arguments](#Arguments)])

SEM026\_WritableCall:

((([Identifier](#Identifier)|**return**|**this**) [“.”([Identifier](#Identifier)|[OperatorName](#OperatorName))])|**old** [“**{**”[UnitTypeName](#UnitTypeName)”**}**”])

[[Arguments](#Arguments)] {[CallChain](#CallChain)}

SEM027\_NewStatement:

(**new** [“**{**” UnitType “**}**”] **return** ) | (**[new]** [“**{**” UnitType “**}**”] [Identifier](#Identifier) )

([“.”**init**] [ [Arguments](#Arguments) ])

SEM028\_NewExpression: [**new]** UnitType [“.”**init**] [ [Arguments](#Arguments) ]

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

(val-, sem-) Arguments: “**(**” [[ExpressionList](#ExpressionList)] ”**)**”

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

SEM029\_IfCase:

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

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

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

(val-, sem-) IfBody:

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

(val-, sem-) ValueAlternative:

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

{“,”

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

}

(val-, sem-) MemberDescription:

([**rtn**] [RoutineName](#RoutineName)[[Signature](#Signature)])|([Idenitifer](#Identifier){“,”[Idenitifer](#Identifier)} ”**:**” [UnitType](#UnitType))

SEM030\_Loop:

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

|

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

[[EnsureBlock](#EnsureBlock)] **end**

(val-, sem-) Type: [”**?**”] [AttachedType](#AttachedType)

(val-, sem-) AttachedType: [UnitType](#UnitType)|[AnchorType](#AnchorType)|[MultiType](#MultiType)|[TupleType](#TupleType)|[RangeType](#RangeType)|[RoutineType](#RoutineType)|[AnonymousUnitType](#AnonymousUnitType)

(val-, sem-) AnonymousUnitType: “**unit**” [MemberDesciption](#memberDescription) {[“;”] [MemberDesciption](#memberDescription)} “**end**”

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

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

(val-, sem-) RangeType:

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

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