**SLang/E# syntax-validity-semantics manual. Version 0.99.18, March 5th 2023**

1. **SLang keywords:**

|  |  |  |
| --- | --- | --- |
| ***#*** | ***Name*** | ***Brief description*** |
| 1 | **abstract** | Unit/Routine characteristic: Bodyless (‘abstract’) unit routine or objectless (‘abstract’) unit. Potentially may be replaced by virtual keyword |
| 2 | **active** | Type/entity/statement: It can be a unit or data attribute or raise statement |
| 3 | **alias** | Unit/Routine characteristic: The alternative name of the unit or routine |
| 4 | **as** | Unit level/Type: Another name in use-as directive or anchor reference |
| 5 | **case** | Statement: start of alternative (may be removed if proper parsing done for ‘:’ after alternative tag ) |
| 6 | **const** | Unit level: Start of constant objects declaration section or constant attribute declaration  Unit-routine level: Import of constant objects of some unit |
| 7 | **do** | Statement: Start of the block |
| 8 | **else** | Statement: Start of else part |
| 9 | **elsif** | Statement: Start of the else if section |
| 10 | **end** | End of block or other syntax construction |
| 11 | **ensure** | Predicate: Routine post-condition clause start |
| 12 | **extend** | Unit level: Used to support inheritance and unit extensions. |
| 13 | **final** | Unit level: The unit cannot have descendants  Unit 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. And it is possible to give a final name to some routine to use it in descendants |
| 14 | **foreign** | Routine characteristic: The body of the routine is coded in 3rd party language |
| 16 | **if** | Statement: Conditional statement start |
| 17 | **in** | Operator: checks if some expression value belongs to the range of values |
| 18 | **is** | Statement: Definition of the initial value of an attribute. Checks for the value or type of expression |
| 19 | **new** | Statement/Expression: Creation of an object. Maybe skipped |
| 20 | **old** | Expression/Statement: 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 |
| 21 | **override** | Unit member characteristics: States that this attribute overrides all possible inherited versions. |
| 22 | **pure** | Routine characteristic: Routine is prohibited to write into unit attributes or read them. Must work only with its parameters. No side effects. Can be safely evaluated once. Can be overridden only by pure routines |
| 23 | **raise** | Statement: Raises a new exception with some object as an argument. If no argument is provided then it raises the last exception occurred |
| 24 | **ref** | Type: States that an object will be of the reference nature |
| 25 | **require** | Predicate: Start of precondition clause of the routine, unit, or loop invariant |
| 26 | **return** | Statement: Stops execution of the routine and returns result in case of function.  Expression/Postcondition: Refers to the function result |
| 27 | **rigid** | Attribute prefix: A deep version of attribute immutability. Deep constant |
| 28 | **rtn** | Type: Has 2 meanings – denotes the routine type after a colon (‘:’) or creates a routine object from some routine in expressions |
| 29 | **safe** | Routine characteristic: Routine is prohibited to write into unit attributes but it can read them |
| 30 | **select** | Unit level: select one version among several versions to resolve ambiguity to support polymorphic assignments |
| 31 | **this** | Expression: Reference to the current object |
| 32 | **unit** | Unit level: Start of the unit description  Expression: duck typing style type check |
| 33 | **use** | Unit/Routine level: It states that the unit mentioned in the use directive will be used as a module (singleton) at the current unit or routine level. It allows renaming units as well. Unit level: give a new name to the inherited member  System-level: import constants of some unit for the current source |
| 34 | **val** | Type: States that an object will be of value nature. The object itself but not a reference to it. |
| 35 | **var** | Attribute/parameter prefix: States that attribute can be assigned many times. It is a variable attribute of any type including routine one. If it is routine parameters then routines with side–effects can be called upon this parameter, as well as an assignment into it |
| 36 | **when** | Statement: Exception handling condition clause. Part of the block |
| 37 | **while** | Statement: Loop condition clause |

* All entity names (unit attributes constant and variable, routine local attributes constant and variable) and routine names (except constructors which are started with upper-case) are started with low-case character while all unit names are started with upper-case one.
* SLang supports 2 modes of syntax Pascal-like and C-like depending on the source file extension (.slang and .clang accordingly)
* Lexical elements are numbers (integer and real ones), characters (‘X’), and strings (“xxx”) literals
* Compiler is the tool which receives set of sources and processes them according to their content. Every source should contain [Compilation](#Compilation)

1. **SLang syntax: list of all syntax rules decorated with validity rules and runtime semantics**
2. Compilation: {[[Build](#Build)] [CompilationUnit](#CompilationUnit)} S() F(EOF)

**VAL-COMP**: is valid if and only if all [CompilationUnit](#CompilationUnit)s are valid in the context of the [Build](#Build) provided or if no [Build](#Build) then in the default or specified as the compiler argument context

1. Build:

**build** [FSname](#FileName) (**from** [FSname](#FileName){[“,”] [FSname](#FileName)})|(**entry** [Identifier](#Identifier))

// library: input paths for the build. E.g. Kernel: “.” “../some path”

// open topic: recursive paths like “../some path/\*\*” not supported currently

// or

// program: entry point identifier stands for unit or routine name

[**target** [Win32|Win64|Lin32|Lin64|Android|iOS|MSIL|JVM|C|ARK|All] // target code kind

[**cluster** {[FSname](#FileName) // Cluster name or path to look for units

[

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

/\* Exclude/hide unit clause. Do not consider some units for the current build \*/

[**use** [Identifier](#Identifier) “**as**” [Identifier](#Identifier) {[“,”][Identifier](#Identifier) “**as**” [Identifier](#Identifier)}]

/\* Rename unit clause to ensure this unit will be used in the current build under the new name\*/

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

/\* Select unit clause works to resolve the case when several clusters have units with the same name. Unit name is strictly attached to particular cluster for the current build\*/

[**end**] // if hide or use or select specified then we need end

]

}]

[**foreign** {[FSname](#FileName)}] // List of 3rd party modules to be linked in

**end** S() F()

**VAL-BLD**: is valid if and only if TBD

1. (val, sem) FSname: (PathOrFileName| [StringConstant](#StringConstant)) [“\*”] S() F()

**VAL-FSNM**: is valid if and only if TBD

1. ([val](#VAL002_Compilation_Partial_Validity), sem) CompilationUnit:

{ [GlobalAlias](#GlobalAlias)} ([AnonymousRoutine](#AnonymousRoutine)|[StandaloneRoutine](#StandaloneRoutine)|[UnitDeclaration](#UnitDeclaration)) S() F()

**VAL-CU**: is valid if and only if TBD

1. (val) UseDirective:

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

**VAL-USDIR**: is valid if and only if TBD

1. GlobalAlias: **alias** [GlobalAliasElement](#GlobalAliasElement) {“**,**” [GlobalAliasElement](#GlobalAliasElement)}
2. GlobalAliasElement: [AttachedType](#AttachedType) **as** UnitName
3. (val) EnclosedUseDirective:

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

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

**VAL-ENCUSDIR**: is valid if and only if TBD

1. (val) EnclosedUseEement: [UnitTypeName](#UnitTypeName) [**as** UnitName] S() F()

**VAL-ENCUE**: is valid if and only if TBD

1. ([val](#VAL005_AnonymousRoutine), sem) AnonymousRoutine: [[UseDirective](#UseDirective)] [StatementsList](#StatementsList) S() F()

**VAL-ARTN**: is valid if and only if is [StatementsList](#StatementsList) valid

1. ([val](#VAL004_Statement_List), sem) StatementsList: {[Statement](#Statement)[“**;**”]} S() F()

**VAL-STMLST**: is valid if and only if every [Statement](#Statement) is valid

1. (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]) S() F()

**VAL-STALNRTN**: is valid if and only if TBD

1. (val, [sem](#SEM001_InnerBlock)) InnerBlock:

“**{**”|**do**|**safe**|**pure** [GroupStart [Identifier](#Identifier) {“,” [Identifier](#Identifier)} GroupEnd]

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

**VAL-INNBLK**: is valid if and only if TBD

1. ([val](#VAL004_Statement_List), sem) WhenClause: **when** ([[Identifier](#Identifier)**:**][UnitType](#UnitType))| [Expression](#Expression) BlockStart [StatementsList](#StatementsList) S() F()

**VAL-WHNCLS**: is valid if and only if TBD

1. UnitRoutineParameters: “**(**”[[UnitRoutineParameter](#Parameter){”**;**”|”,” [UnitRoutineParameter](#Parameter)}]“**)**”S() F()

**VAL-UNTRTNPARS**: is valid if and only if

1. every [UnitRoutineParameter](#Parameter) is valid
2. all [UnitRoutineParameter](#Parameter) are unique
3. (val) StandaloneRoutineParameters:

“**(**”[[StandaloneRoutineParameter](#Parameter){”**;**”|”,” [StandaloneRoutineParameter](#Parameter)}]“**)**” S() F()

**VAL-STALNRTNPARS**: is valid if and only if

1. every [StandaloneRoutineParameter](#Parameter) is valid
2. all [StandaloneRoutineParameter](#Parameter) are unique
3. (val) UnitRoutineParameter: [StandaloneRoutineParameter](#StandaloneRoutineParameter)|(“**:=**” [[Identifier](#Identifier)])) S() F()

**VAL-UNTRTNPAR**: is valid if and only if TBD

1. (val) StandaloneRoutineParameter: ([[**rigid**] [Identifier](#Identifier){“**,**” [**rigid**] [Identifier](#Identifier)} “**:**” [Type](#Type)) | ([Identifier](#Identifier) “**is**” [Expression](#Expression)) S() F()

**VAL-STALNRTNPAR**: is valid if and only if TBD

1. (val, sem) RequireBlock : **require** [PredicatesList](#PredicatesList) S() F()

**VAL-PREBLK**: is valid if and only if [PredicatesList](#PredicatesList) is valid

1. (val, sem) EnsureBlock : **ensure** [PredicatesList](#PredicatesList) S() F()

**VAL-PSTBLK**: is valid if and only if [PredicatesList](#PredicatesList) is valid

1. (val, sem) InvariantBlock: **require** [PredicatesList](#PredicatesList) S() F()

**VAL-INVBLK**: is valid if and only if [PredicatesList](#PredicatesList) is valid

1. (val, sem) PredicatesList: [[Predicate](#Predicate){[”**;**”] [Predicate](#Predicate)}] S() F()

**VAL-PRDLST**: is valid if and only if

1. every [Predicate](#Predicate) is valid
2. all [Predicate](#Predicate) are unique
3. (val, sem) Predicate: [BooleanExpression](#BooleanExpression) [[DocumentingComment](#DocumentingComment)] S() F()

**VAL-PRD**: is valid if and only if TBD

1. (val) UnitDeclaration: ([**final**] [**ref**|**val**|**active**])|[**abstract**]|[**extend**] **unit** [UnitName](#UnitName) [**alias** [UnitName](#UnitName)] [[FormalGenerics](#FormalGenerics)] [[InheritDirective](#InheritDirective)] [[EnclosedUseDirective](#EnclosedUseDirective)] [[MemberSelection]](#MemberSelection) [[InheritedMemberOverriding]](#InheritedMemberOverriding) [[InitProcedureInheritance](#InitProcedureInheritance)] [[ConstObjectsDeclaration](#ConstObjectsDeclaration)] { ( [MemberVisibility](#MemberVisibility) “:” {[MemberDeclaration](#FeatureDeclaration)}) | [MemberDeclaration](#FeatureDeclaration) } [[InvariantBlock](#InvariantBlock)] BlockEnd S() F()

**VAL-UNTDCL**: is valid if and only if TBD

1. (val) InheritDirective: **extend** [Parent](#Parent) {“,” [Parent](#Parent)} S() F()

**VAL-INHDIR**: is valid if and only if

1. every [Parent](#Parent) is valid
2. all [Parent](#Parent)s are unique
3. (val) Parent: [UnitTypeName](#UnitTypeName) | (“**~**” [UnitTypeName](#UnitTypeName) [“(”[MemberName](#MemberName){“,”[MemberName](#MemberName)}“)”]) S() F()

**VAL-PRNT**: is valid if and only if TBD

1. (val) MemberName: [Identifier](#Identifier)|([RoutineName](#RoutineName) [[Signature](#Signature)]) S() F()

**VAL-MBRNM**: is valid if and only if TBD

1. (val) FormalGenerics: GenericsStart [FormalGeneric](#FormalGeneric) {“**,**” [FormalGeneric](#FormalGeneric)} GenericsEnd S() F()

**VAL-FGLST**: is valid if and only if

1. every [FormalGeneric](#FormalGeneric) is valid
2. all [FormalGeneric](#FormalGeneric)s are unique
3. (val) FormalGeneric: ([UnitName](#UnitName) [“**extend**” [UnitTypeName](#UnitTypeName)] [“**new**” [[Signature](#Signature)]])| ([Identifier](#Identifier) “**:**” [UnitType](#UnitType)|[RoutineType](#RoutineType)) S() F()

**VAL-FG**: is valid if and only if

1. TBD every is valid
2. (val) MemberSelection: **select** [MemberName](#MemberName) {“**,**”[MemberName](#MemberName)} S() F()

**VAL-MBR-SEL**: is valid if and only if

1. every [MemberName](#MemberName) is valid and unique
2. (val) InheritedMemberOverriding: **override** [UnitTypeName](#UnitTypeName)”**.**”[MemberName](#MemberName) {“,” [UnitTypeName](#UnitTypeName)”**.**”[MemberName](#MemberName)} S() F()

**VAL-INH-MBR-OVRD**: is valid if and only if

1. TBD every is valid
2. (val) InitProcedureInheritance: **new** [UnitTypeName](#UnitTypeName)[[Signature](#Signature)] {“**,**” [UnitTypeName](#UnitTypeName)[[Signature](#Signature)]} S() F()

**VAL-INT-PROC-INH**: is valid if and only if

1. TBD every is valid
2. (val) MemberVisibility: “**{**” [**this**| [UnitName](#UnitName) {“**,**” [UnitName](#UnitName)} ] “**}**” S() F()

**VAL-MBR-VSBL**: is valid if and only if

1. TBD every is valid
2. (val) MemberDeclaration: [[MemberVisibility](#MemberVisibility)] ([**override**] [**final**] [UnitAttribiteDeclaration](#UnitAttributeDeclaration)|[UnitRoutineDeclaration](#UnitRoutineDeclaration)) | [InitDeclaration](#InitDeclaration) S() F()

**VAL-MBR-DCL**: is valid if and only if

1. TBD every is valid
2. (val) InitDeclaration: [UnitName](#UnitName) [[UnitRoutineParameters](#UnitRoutineParameters)] [[EnclosedUseDirective](#EnclosedUseDirective)] [[RequireBlock](#RequireBlock)] ([InnerBlock](#InnerBlock) [[EnsureBlock](#EnsureBlock)] BlockEnd)|(**foreign**|**none** [[EnsureBlock](#EnsureBlock) BlockEnd]) S() F()

**VAL-INIT-DCL**: is valid if and only if

1. TBD every is valid
2. (val) UnitRoutineDeclaration: [**pure**|**safe**] [RoutineName](#RoutineName) [**final** [Identifier](#Identifier)] [[UnitRoutineParameters](#UnitRoutineParameters)] [ReturnType [Type](#Type)] [[EnclosedUseDirective](#EnclosedUseDirective)][[RequireBlock](#RequireBlock)](([InnerBlock](#InnerBlock)) [[EnsureBlock](#EnsureBlock)] BlockEnd) | ((**abstract**|**foreign**|**none**| (“**=>**”[Expression](#Expression)))[[EnsureBlock](#EnsureBlock) BlockEnd]) S() F()

**VAL-UNT-RTN-DCL**: is valid if and only if

1. TBD every is valid
2. (val) RoutineName: [Identifier](#Identifier) |“**()**”|“**:=**”|([OperatorName](#OperatorName) [[AliasName](#AliasName)]) S() F()

**VAL-RTN-NM**: is valid if and only if

1. TBD every is valid
2. (val) AliasName: **alias** ([Identifier](#Identifier)|“*and then*”|“*or else*” ) S() F()

**VAL-ALS-NM**: is valid if and only if

1. TBD every is valid
2. (val) OperatorName : [OperatorSign](#OperatorSign) [[OperatorSign](#OperatorSign)] S() F()

**VAL-OPR-NM**: is valid if and only if

1. TBD every is valid
2. (val) OperatorSign : “**^**” | “**\***” | “**/**” | “**\**” | “**=**” | “**+**” | “**-**“ |”**<**” | ”**>**” | ”**&**” | “**|**”|“#” | “%”| “@”| “!”| “$”| “~” S() F()

**VAL-OPR-SGN**: is valid if and only if

1. TBD every is valid
2. (val, sem) ConstObjectsDeclaration: **const** “:”[ [ConstObject](#ConstObject) { “**,**” [ConstObject](#ConstObject)} ] BlockEnd S() F()

**VAL-CNST-OBJS-DCL**: is valid if and only if

1. TBD every is valid
2. (val, sem) ConstObject:

(

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

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

)

|

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

S() F()

**VAL-CNST-OBJ-DCL**: is valid if and only if

1. TBD every is valid
2. (val, sem) RegularExpression: [Constant](#Constant) ({“**|**”[Constant](#Constant)}) | (“**|**””**..**” [Constant](#Constant)) S() F()

**VAL-RGL-EXPR**: is valid if and only if

1. TBD every is valid
2. ([val](#VAL006_Statement), sem) Statement: [Assignment](#Assignment) | [LocalAttributeDeclaration](#LocalAttributeCreation)| [WritableCall](#WritableCall) | [ObjectCreation](#ObjectCreation) | [Conditional](#Conditional)| [Loop](#Loop) | [Detach](#Detach)|[Return](#Return) |[HyperBlock](#HyperBlock)| [Raise](#Raise) | [UnpackTuple](#UnpackTuple) S() F()

**VAL-STMT**: is valid if and only if

1. TBD every is valid
2. (val, sem) Detach: **?** [Identifier](#Identifier) [NewLine] S() F()

**VAL-DTCH**: is valid if and only if

1. TBD every is valid
2. (val, sem) Raise:[**active**] **raise** [[Expression](#Expression)] [NewLine] S() F()

**VAL-RS**: is valid if and only if

1. TBD every is valid
2. ([val](#VAL015_Return), sem)  Return: **return** [[Expression](#Expression)] [NewLine] S() F()

**VAL-RTRN**: is valid if and only if

1. TBD every is valid
2. (val, sem) HyperBlock: [[RequireBlock](#RequireBlock)] [InnerBlock](#InnerBlock) [[EnsureBlock](#EnsureBlock)] BlockEnd S() F()

**VAL-HPR-BLK**: is valid if and only if

1. TBD every is valid
2. ([val](#VAL007_Assignment), sem) Assignment: [Writable](#Writable) “**:=**” [Expression](#Expression) [NewLine] S() F()

**VAL-ASGN**: is valid if and only if

1. TBD every is valid
2. (val, sem) Writable: [WritableCall](#WritableCall) | (“**(**”[WritableCall](#WritableCall) {“**,**” [WritableCall](#WritableCall) } “**)**”) S() F()

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

**VAL-WRTBL**: is valid if and only if

1. TBD every is valid
2. ([val](#VAL008_LocalAttribute), sem) UnpackTuple: “**(**“ [LocalAttributeNamesList](#LocalAttributeNamesList) “**)**” **is** [Expression](#Expression)

**VAL-UNPK-TPL**: is valid if and only if

1. TBD every is valid
2. ([val](#VAL008_LocalAttribute), sem) LocalAttributeDeclaration:

[LocalAttributeNamesList](#LocalAttributeNamesList) ([“**:**” [Type](#Type)] **is** [Expression](#Expression))|(“**:**” [Type](#Type)) [**final** “**=>**” [Statement](#Statement)]

[NewLine]

S() F()

**VAL-LCL-ATTR-DCL**: is valid if and only if

1. TBD every is valid
2. (val, sem) LocalAttributeNamesList: [**var**|**rigid**] [Identifier](#Identifier) {“**,**”[**var**|**rigid**] [Identifier](#Identifier)} S() F()

**VAL-LCL-ATTRS-LST**: is valid if and only if

1. TBD every is valid
2. (val, sem) UnitAttributeDeclaration: **ПЕРЕДЕЛЫВАТЬ !!!! ЗАЧЕМ ????**

(( [UnitAttributeNamesList](#UnitAttributeNamesList) “:” [Type](#Type)) | ( [Identifier](#Identifier) [“:” [AttachedType](#AttachedType)] **is** [ConstantExpression](#ConstantExpression) [NewLine]) | ( **const**|**rigid** [Identifier](#Identifier) [“:” [AttachedType](#AttachedType)] **is** [ConstantExpression](#ConstantExpression) [NewLine] {“**,**”[Identifier](#Identifier) [“:” [AttachedType](#AttachedType)] **is** [ConstantExpression](#ConstantExpression) [NewLine]} ) | ([Identifier](#Identifier) “:” [Type](#Type) **rtn** “:=” [[[[[UnitRoutineParameters](#UnitRoutineParameters)] [HyperBlock](#HyperBlock)](#TupleExpression)](#OldExpression)]) ) [**final =>** [Statement](#Statement)[NewLine]] S() F()

**VAL-UNT-ATTR-DCL**: is valid if and only if

1. TBD every is valid
2. (val) UnitAttributeNamesList: [**const** | **rigid**] [Identifier](#Identifier) {“**,**”[**const** | **rigid**] [Identifier](#Identifier)} S() F()

**VAL-UNT-ATTRS-LST**: is valid if and only if

1. TBD every is valid
2. (val, sem) BooleanExpression: [Expression](#Expression) S() F()

**VAL-BOOL-EXPR**: is valid if and only if

1. [Expression](#Expression) is valid
2. [Expression](#Expression) has type Boolean
3. (val, sem) ConstantExpression: ([Identifier](#Identifier) {“**.**” [Identifier](#Identifier)}) | [Constant](#Constant) [[Operator](#Operator) [ConstantExpression](#ConstantExpression)] S() F()

**VAL-CNST-EXPR**: is valid if and only if

1. TBD every is valid
2. (val, sem) Expression: [[ForcedType](#ForcedType)] [IfExpression](#IfExpession) |[WritableCall](#WritableCall)| [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)} S() F()

**VAL-EXPR**: is valid if and only if

1. TBD every is valid
2. (val, sem) [RefExpression:](#TupleExpression) **ref** [Expression](#Expression) S() F()

**VAL-REF-EXPR**: is valid if and only if

1. TBD every is valid
2. (val, sem) LambdaExpression: (**rtn** [Identifier](#Identifier) [[Signature](#Signature)])|[InlineLambdaExpression](#InlineLambdaExpression) S() F()

**VAL-RTN-EXPR**: is valid if and only if

1. TBD every is valid
2. (val, sem) InlineLambdaExpression: [[[**pure**|**safe**] **rtn** [[StandaloneRoutineParameters](#StandaloneRoutineParameters)] [ReturnType [Type](#Type)] ([[RequireBlock](#RequireBlock)] ([InnerBlock](#InnerBlock) BlockEnd)|(**foreign** [[EnsureBlock](#EnsureBlock)] BlockEnd])|(“**=>**”[Expression](#Expression))](#EnsureBlock)](#OldExpression) S() F()

**VAL-LMBD-EXPR**: is valid if and only if

1. TBD every is valid
2. (val, sem) RangeExpression: [Expression](#Expression) [“{” [OperatorName](#OperatorName) [ConstantExpression](#ConstantExpression)“}”] “**..**” [Expression](#Expression) S() F()

**VAL-RNG-EXPR**: is valid if and only if

1. TBD every is valid
2. (val, sem) OldExpression: [**old** [Expression](#Expression)](#TupleExpression) S() F()

**VAL-OLD-EXPR**: is valid if and only if

1. TBD every is valid
2. (val, sem) TupleExpression: “**(**”[[TupleElement](#TupleElement) {“**,**” [TupleElement](#TupleElement)}]“**)**” S() F()

**VAL-TPL-EXPR**: is valid if and only if

1. TBD every is valid
2. (val, sem) TupleElement: [Expression](#Expression)| [Parameter](#Parameter) S() F()

**VAL-TPL-ELMNT**: is valid if and only if

1. TBD every is valid
2. (val, sem) TypeOfExpression: [Expression](#Expression) **is** (**“?”**| [UnitType](#UnitType) | [AnonymousUnitType](#AnonymousUnitType)) S() F() /\* Duck typing may work only when no preconditions and postconditions are for all routines of the [AnonymousUnitType](#AnonymousUnitType)) \*/

**VAL-IS-EXPR**: is valid if and only if

1. TBD every is valid
2. (val, sem) Operator: [OperatorName](#OperatorName)|**in** S() F()

**VAL-OPRTR**: is valid if and only if

1. TBD every is valid
2. (val, sem) Constant: [[UnitTypeName](#UnitTypeName) “.”]( [StringConstant |](#StringConstant) [CharacterConstant |](#CharacterConstant) [IntegerConstant |](#IntegerConstant) [RealConstant |](#RealConstant) [BooleanConstant](#BooleanConstant) | [BitConstant](#BitConstant) | [Identifier](#Identifier) ) S() F()

**VAL-CNST**: is valid if and only if

1. TBD every is valid
2. (val, sem) IfExpression:

**if** [Expression](#Expression) (**case** [ExpressionAlternatives](#ExpressionAlternatives))|( BlockStart[Expression](#Expression))  
{**elsif** [Expression](#Expression) (**case** [ExpressionAlternatives](#ExpressionAlternatives))|( BlockStart[Expression](#Expression))}  
**else** [Expression](#Expression) “}”**Cmod**  
S() F()

**VAL-IF-EXPR**: is valid if and only if

1. TBD every is valid
2. (val, sem) ExpressionAlternatives: [AlternativeTags](#AlternativeTags) [Expression](#Expression) { **case** [AlternativeTags](#AlternativeTags) [Expression](#Expression)} S() F()

**VAL-IF-ALTS**: is valid if and only if

1. TBD every is valid
2. (val, sem) WritableCall:

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

|(**old** [[ForcedType](#ForcedType)]) //Precursor call – ForcedType resolves Parent to call precursor from

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

S() F()

**VAL-WRTBL-CALL**: is valid if and only if

1. TBD every is valid
2. (val, sem) ObjectCreation: **new** [[ForcedType](#ForcedType)] **return**|[Identifier](#Identifier) [[Arguments](#Arguments)] S() F()

**VAL-OBJ-CRTN**: is valid if and only if

1. TBD every is valid
2. (val, sem) NewExpression: [**new**] [UnitType](#UnitType) [[Arguments](#Arguments)] S() F()

**VAL-NEW-EXPR**: is valid if and only if

1. TBD every is valid
2. (val, sem) CallChain: “**.**”([Identifier](#Identifier)|[OperatorName](#OperatorName)) [[Arguments](#Arguments)] S() F()

**VAL-CALL-CHN**: is valid if and only if

1. TBD every is valid
2. (val, sem) Arguments: “**(**” [[ExpressionList](#ExpressionList)] ”**)**” S() F()

**VAL-ARGS**: is valid if and only if

1. TBD every is valid
2. (val) ForcedType: GroupStart [UnitType](#UnitType) GroupEnd

**VAL-FRCD-TYP**: is valid if and only if

1. TBD every is valid
2. (val, sem) ExpressionList: [Expression](#Expression) {“**,**” [Expression](#Expression)} S() F()
   1. ~~ExpressionList: [[ForcedType](#ForcedType)]~~ [~~Expression~~](#Expression) ~~{“~~**~~,~~**~~” [[ForcedType](#ForcedType)]~~ [~~Expression~~](#Expression)~~}~~

**VAL-EXPR-LST**: is valid if and only if

1. every [Expression](#Expression) is valid
2. ([val](#VAL009_If), sem) Conditional:

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

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

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

**VAL-IF**: is valid if and only if

1. TBD every is valid
2. (val, sem) Alternatives:

[AlternativeTags](#AlternativeTags) [StatementsList](#StatementsList) { **case** [AlternativeTags](#AlternativeTags) [StatementsList](#StatementsList)}

S() F()

**VAL-IF-ALTS**: is valid if and only if

1. TBD every is valid
2. (val, sem) AlternativeTags: [AlternativeTag](#AlternativeTag) {“**,**” [AlternativeTag](#AlternativeTag)} S() F()

**VAL-ALT-TGS**: is valid if and only if

1. TBD every is valid
2. (val, sem) AlternativeTag: [Expression](#Expression) [[GroupStart [OperatorName](#OperatorName) [ConstantExpression](#ConstantExpression) GroupEnd] “**..**”[Expression](#Expression)] S() F()

**VAL-ALT-TG**: is valid if and only if

1. TBD every is valid
2. (val) MemberDescription: ([**rtn**] [RoutineName](#RoutineName)[[Signature](#Signature)])| ([Idenitifer](#Identifier){“,”[Idenitifer](#Identifier)} ”**:**” [UnitType](#UnitType)) S() F()

**VAL-MBR-DSC**: is valid if and only if

1. TBD every is valid
2. ([val](#VAL010_Loop), sem) Loop:(**while** [BooleanExpression](#BooleanExpression){“,”[BooleanExpression](#BooleanExpression)} [[RequireBlock](#RequireBlock)] [InnerBlock](#InnerBlock)) | ([[RequireBlock](#RequireBlock)] [InnerBlock](#InnerBlock) **while** [BooleanExpression](#BooleanExpression){“,”[BooleanExpression](#BooleanExpression)}) [[EnsureBlock](#EnsureBlock)] BlockEnd S() F()

**VAL-LOOP**: is valid if and only if

1. TBD every is valid
2. (val) Type: [”**?**”] [AttachedType](#AttachedType) S() F()

**VAL-TYPE**: is valid if and only if

1. TBD every is valid
2. (val) AttachedType: [UnitType](#UnitType)|[AnchorType](#AnchorType)|[MultiType](#MultiType)|[TupleType](#TupleType)|[RangeType](#RangeType)|[RoutineType](#RoutineType)|[AnonymousUnitType](#AnonymousUnitType) S() F()

**VAL-ATCH-TYP**: is valid if and only if

1. TBD every is valid
2. (val) AnonymousUnitType: **unit** [MemberDesciption](#memberDescription) {[“;”] [MemberDesciption](#memberDescription)} BlockEnd S() F()

**VAL-ANM-TYP**: is valid if and only if

1. TBD every is valid
2. (val) RoutineType: **rtn** [[Signature](#Signature)]  S() F()

**VAL-RTN-TYP**: is valid if and only if

1. TBD every is valid
2. (val) Signature: (“**(**”[[Type](#Type) {“**,**” [Type](#Type)}]“**)**”[ ReturnType [Type](#Type)])| (ReturnType [Type](#Type)) S() F()

**VAL-SGN**: is valid if and only if

1. TBD every is valid
2. (val) RangeType: RangeTypeItem {“**,**” RangeTypeItem}

**VAL-RNG-TYP**: is valid if and only if

1. TBD every is valid
2. (val) RangeTypeItem: [ConstantExpression](#ConstantExpression) [[GroupStart[OperatorName](#OperatorName) [ConstantExpression](#ConstantExpression) GroupEnd] “**..**” [ConstantExpression](#ConstantExpression) ] S() F()

**VAL-RNG-TYPE-ITM**: is valid if and only if

1. TBD every is valid
2. (val) AnchorType: **as** (**this**|([Identifier](#Identifier) [[Signature](#Signature)])) S() F()

**VAL-ANCHR-TYP**: is valid if and only if

1. TBD every is valid
2. (val) MultiType: [UnitType](#UnitType) {“**|**”[UnitType](#UnitType)} S() F()

**VAL-MLT-TYP**: is valid if and only if

1. TBD every is valid
2. (val) TupleType: “**(**”[[TupleField](#TupleField) {“**,**”|”**;**” [TupleField](#TupleField)}]“**)**” S() F()

**VAL-TPL-TYP**: is valid if and only if

1. TBD every is valid
2. (val) TupleField: [[Identifier](#Identifier) {“**,**” [Identifier](#Identifier)}“**:**”] [UnitType](#UnitType) S() F()

**VAL-TPL-FLD**: is valid if and only if

1. TBD every is valid
2. (val) UnitTypeName: {[Identifier](#Identifier)“**.**”} [UnitName](#UnitName) [GenericsStart([Type](#Type)|[ConstantExpression](#ConstantExpression)) {“**,**” ([Type](#Type)|[ConstantExpression](#ConstantExpression))} GenericsEnd ] S() F()

**VAL-UNT-TYPE\_NM**: is valid if and only if

1. TBD every is valid
2. (val) UnitType: [**ref**|**val**|**active**] [UnitTypeName](#UnitTypeName) S(**ref, val, active, typeName**) F()

**VAL-UNT-TYP**: is valid if and only if

1. TBD every is valid
2. (val) UnitName: [UpperCaseLetter](#UpperCaseLetter) { [Letter](#Letter) | [Digit](#Digit) | ’\_’ }

**VAL-UNT-NM**: is valid if and only if

1. TBD every is valid
2. DocumentingComment: “**///**” { [Character](#Character) }

**VAL-DOC-CMNT**: is valid if and only if

1. TBD every is valid
2. Comment: (“**//**” {[Character](#Character)}) | (”**/\***” { [Character](#Character) } “**\*/**”)

**VAL-CMNT**: is valid if and only if

1. TBD every is valid
2. ReturnType: “**:**”|”**->**”
3. BlockEnd: **end**|”**}**”**Cmod**
4. BlockStart: **do**|”**{**”**Cmod**
5. GenericsStart: “**[**“|”**<**”**Cmod**
6. GenericsEnd: “**]**”|”**>**”**Cmod**
7. GroupStart: “**{**”|”**[**“**Cmod**
8. GroupEnd: “**}**”|”**]**“**Cmod**
9. NewLine: “**;**”|”\n”
10. Identifier: [LowerCaseLetter](#LowerCaseLetter) { [Letter](#Letter) | [Digit](#Digit) | ’\_’ }
11. StringConstant: “**”**” { [Character](#Character) } “**”**”
12. CharacterConstant: “**’**” [Character](#Character) “**’**”
13. IntegerConstant: [ “**+**”|”**-**“ ] ([Digit](#Digit) { [Digit](#Digit) } [”**H**”|”**h**”|“**O**”|“**o**”])|(“0” (“**x**”|“**X**”|“**o**”|“**O**”) [Digit](#Digit) { [Digit](#Digit) })
14. BitConstant: [ “**+**”|”**-**“ ] (“0”|”1“ {“0”|”1“} [“**B**”|”**b**”]) | (“0” ( “**b**”|“**B**”) “0”|”1“{“0”|”1“})
15. RealConstant: [ “**+**”|”**-**“ ] [Digit](#Digit) { [Digit](#Digit) } “.”{ [Digit](#Digit) } [“**e**”|”**E**”] [“**+**”|”**-**“] [Digit](#Digit) { [Digit](#Digit) }
16. Character: [Letter](#Letter) | [Digit](#Digit) | [Symbol](#Symbol) | UnicodeSymbol | ControlCharacter
17. Letter : ‘**A**’ | .. ’**Z**’ | ’**a**’ | ..’**z**’
18. UpperCaseLetter: ‘**A**’ | .. ’**Z**’
19. LowerCaseLetter: ’**a**’ | ..’**z**’
20. Digit: ’**0**’ | ..’**9**’ | ’**A**’..’**F**’
21. Symbol: ‘\ASCII symbol code 0..255’
22. UnicodeSymbol: ‘\u’ | ‘\U’ …
23. ControlCharacter: ‘\n’ | ‘\t’
24. **SLang validity: list of all validity rules**

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 <[LocalAttributeDeclaration](#LocalAttributeDeclaration)> or <[Conditional](#Conditional)> or <[Conditional](#Conditional)> 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): <[LocalAttributeDeclaration](#LocalAttributeDeclaration)> is valid if and only if …

VAL009\_If (IV): <[Conditional](#Conditional)> 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

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 and if it was evaluated to false then an exception is generated. If [DocumentingComment](#DocumentingComment) is provided then it is passed as an argument for exception object creation

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

[Assignment](#Assignment)| [LocalAttributeDeclaration](#LocalAttributeDeclaration)| [WritableCall](#WritableCall)| [ObjectCreation](#ObjectCreation)| [Conditional](#Conditional)| [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*]

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\_LocalAttributeDeclaration:

([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) | [[WritableCall](#WritableCall)](#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)|(**new** [[Arguments](#Arguments)])

SEM026\_WritableCall:

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

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

SEM027\_ObjectCreation:

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

[[Arguments](#Arguments)]

SEM028\_NewExpression: [**new]** UnitType [[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\_ Conditional:

**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**|**active**] [UnitTypeName](#UnitTypeName)