**SLang/E# syntax-validity-semantics manual. Version 0.99.19, December 9th 2023**

1. **SLang keywords:**

|  |  |  |
| --- | --- | --- |
| ***#*** | ***Name*** | ***Brief description*** |
| 1 | **abstract** | Unit/Routine characteristic: Bodyless (‘abstract’) unit routine or objectless (‘abstract’) unit. |
| 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  Unit: initialization procedure declaration |
| 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 member 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  Expression: produces the reference version of the expression result |
| 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  Visibility: private |
| 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 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 literals:
  + numbers (integer (decimal, hex, binary) and real ones)
  + characters (use single quotes ‘’)
  + strings (use double quotes “”)
* Compiler is the tool which receives set of sources and processes them according to their content. Every source should be a valid [Compilation](#Compilation).
* Notation:
  + [] – optional
  + {} – repetition zero or more times
  + () – grouping
  + | - or
  + [CompilationUnit](#CompilationUnit) - non\_terminal
  + **build** – keyword
  + “,” - symbol, separator
  + Cmod – syntax mode, default is ‘Pascal-like’

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

S(S([Build](#Build)), S([CompilationUnit](#CompilationUnit)))

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 is used for validation.

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(build)

F(S([CompilationUnit](#CompilationUnit)))

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

1. TBD
2. FSname: (PathOrFileName| [StringConstant](#StringConstant)) [“\*”] // star means recursive

**VAL-FSNM**: is valid if and only if FSname refers to existing folder

1. CompilationUnit: { [GlobalAlias](#GlobalAlias) | [UseDirective](#UseDirective) } ([AnonymousRoutine](#AnonymousRoutine)|[StandaloneRoutine](#StandaloneRoutine)|[UnitDeclaration](#UnitDeclaration))

S(alias, use, S([AnonymousRoutine](#AnonymousRoutine)), S([StandaloneRoutine](#StandaloneRoutine)), S([UnitDeclaration](#UnitDeclaration)))

F(build, S([CompilationUnit](#CompilationUnit)))

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

1. all its [UseDirective](#UseDirective)s are valid
2. all its GlobalAliases are valid
3. [AnonymousRoutine](#AnonymousRoutine) or [StandaloneRoutine](#StandaloneRoutine) or [UnitDeclaration](#UnitDeclaration) is valid

**SEM-CU**:

1. all GlobalAliases are accumulated in accordance with the textual order
2. only the latest [UseDirective](#UseDirective) is actual for the current compilation unit
3. UseDirective:

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

S(use)

F()

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

1. every [UnitTypeName](#UnitTypeName) is valid
2. all [UnitTypeName](#UnitTypeName)s are unique across [UseDirective](#UseDirective)

**SEM-USDIR**: all constant declared in the unit called [UnitTypeName](#UnitTypeName) can be used in the current compilation unit with no qualification.

1. GlobalAlias: **alias** [GlobalAliasElement](#GlobalAliasElement) {“**,**” [GlobalAliasElement](#GlobalAliasElement)}

S(alias)

F()

1. GlobalAliasElement: [AttachedType](#AttachedType) **as** [UnitName](#UnitName)

S(S([AttachedType](#AttachedType)))

F()

1. EnclosedUseDirective:

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

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

S(use)

F()

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

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

S(<ident>)

F()

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

1. AnonymousRoutine: [[UseDirective](#UseDirective)] [StatementsList](#StatementsList)

S(S([UseDirective](#UseDirective)), S([Statement](#Statement)))

F()

**VAL-ARTN**: is valid if and only if is [StatementsList](#StatementsList) valid and if [UseDirective](#UseDirective) is present then it is valid too.

1. StatementsList: {[Statement](#Statement)[“**;**”]}

S([Statement](#Statement))

F()

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

1. StandaloneRoutine: [**pure**|**safe**] [Identifier](#Identifier) [[FormalGenerics](#FormalGenerics)] [[Parameters](#Parameters)] [[ReturnType](#ReturnType) [Type](#Type)] [[EnclosedUseDirective](#EnclosedUseDirective)] [[RequireBlock](#RequireBlock)] ([InnerBlock](#InnerBlock) [[EnsureBlock](#EnsureBlock)] BlockEnd)|(((“**=>**”[Expression](#Expression))|**foreign**) [[EnsureBlock](#EnsureBlock) BlockEnd])

S(pure, safe, <ident>)

F()

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

1. InnerBlock:

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

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

S(“{”|do,safe,pure)

F()

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

**SEM-INNBLK**:

1. “{” [Identifier](#Identifier) {“,” [Identifier](#Identifier)} “}” - do not check invariants for these entities within the [InnerBlock](#InnerBlock)
2. [StatementsList](#StatementsList) is executed, if exception is raised then current exception is matched against all [WhenClause](#WhenClause)s in the order of declaration. If match is detected then exception is handled running the code of the matched [WhenClause](#WhenClause). If no match then [StatementsList](#StatementsList) is to be executed and exception is handled. If no [StatementsList](#StatementsList) provided then exception is propagated.
3. WhenClause: **when** ([[Identifier](#Identifier)**:**][UnitType](#UnitType))| [Expression](#Expression) BlockStart [StatementsList](#StatementsList)

S(when)

F()

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

1. [StatementsList](#StatementsList) should be valid
2. If [UnitType](#UnitType) is provided then it is valid and if [Identifier](#Identifier) is provided then it is unique within the surrounding routine context
3. Otherwise [Expression](#Expression) is valid

**SEM-WHNCLS**:

1. If [UnitType](#UnitType) is provided and the current exception type conforms to the [UnitType](#UnitType) or
2. the current exception value is equal to the [Expression](#Expression) value
3. then [StatementsList](#StatementsList) is executed and current exception is handled
4. 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 within the context of [UnitRoutineParameters](#UnitRoutineParameters)
3. 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 across [StandaloneRoutineParameters](#StandaloneRoutineParameters)
3. UnitRoutineParameter: [StandaloneRoutineParameter](#StandaloneRoutineParameter)|(“**:=**” [[Identifier](#Identifier)]))

S(rigid, <ident>, “:=”)

F()

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

1. [StandaloneRoutineParameter](#Parameter) is valid if provided
   1. All [Identifier](#Identifier)s used in [StandaloneRoutineParameter](#Parameter) are uniqiue across all members of the surrounding unit
2. [Identifier](#Identifier) is valid
3. [Identifier](#Identifier) is the name of the surrounding unit attribute
4. StandaloneRoutineParameter: ([[**rigid**] [Identifier](#Identifier){“**,**” [**rigid**] [Identifier](#Identifier)} “**:**” [Type](#Type)) | ([Identifier](#Identifier) “**is**” [Expression](#Expression))

S(rigid, <ident>)

F()

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

1. every [Identifier](#Identifier) is valid
2. every [Type](#Type) is valid if provided
3. [Expression](#Expression) is valid if provided
4. All [Identifier](#Identifier)s are unique across all current routine parameters
5. There are no parameters with default value after every one with the default value
6. TBD
7. RequireBlock : **require** [PredicatesList](#PredicatesList)

S(require)

F()

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

1. EnsureBlock : **ensure** [PredicatesList](#PredicatesList)

S(ensure)

F()

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

1. InvariantBlock: **require** [PredicatesList](#PredicatesList)

S(require)

F()

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

1. PredicatesList: [[Predicate](#Predicate){[”**;**”] [Predicate](#Predicate)}]

S([Expression](#Expression))

F()

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

1. every [Predicate](#Predicate) is valid
2. all [Predicate](#Predicate)s are unique
3. Predicate: [BooleanExpression](#BooleanExpression) [[DocumentingComment](#DocumentingComment)]

S([Expression](#Expression))

F()

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

1. [BooleanExpression](#BooleanExpression) is valid
2. [DocumentingComment](#DocumentingComment) is valid if provided
3. UnitDeclaration: ([**final**] [**ref**|**val**|**active**])|[**abstract**]|[**extend**] **unit** [UnitName](#UnitName) [**alias** [UnitName](#UnitName)] [[FormalGenerics](#FormalGenerics)] [[InheritDirective](#InheritDirective)] [[EnclosedUseDirective](#EnclosedUseDirective)] [[MemberSelection]](#MemberSelection) [[InheritedMemberOverriding]](#InheritedMemberOverriding) [“**{**”**Cmod**] [[ConstObjectsDeclaration](#ConstObjectsDeclaration)] { ( [MemberVisibility](#MemberVisibility) “:” {[MemberDeclaration](#MemberDeclaration)}) | [MemberDeclaration](#MemberDeclaration) } [[InvariantBlock](#InvariantBlock)] BlockEnd

S(final, ref, val, active, abstract, extend, unit)

F(S([Build](#Build)), S([CompilationUnit](#CompilationUnit)))

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

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

S(extend)

F()

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

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

S(<ident>, “**~**”)

F()

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

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

S(<ident>, S([RoutineName](#RoutineName)))

F()

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

1. 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. FormalGeneric: ([UnitName](#UnitName) [**extend** [UnitTypeName](#UnitTypeName)] [**new** [[Signature](#Signature)]])| ([Identifier](#Identifier) “**:**” [UnitType](#UnitType)|[RoutineType](#RoutineType))

S(S([UnitName](#UnitName)), <ident>)

F()

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

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

S(select)

F()

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

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

S(override)

F()

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

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

S(“{“)

F()

**VAL-MBR-VSBL**: is valid if and only if list of [UnitName](#UnitName)s is provided then all [UnitName](#UnitName)s are valid and unique within the list.

1. MemberDeclaration: [[MemberVisibility](#MemberVisibility)] ([**override**] [**final**] [UnitAttribiteDeclaration](#UnitAttributeDeclaration)|[UnitRoutineDeclaration](#UnitRoutineDeclaration)) | [InitDeclaration](#InitDeclaration)

S(S([MemberVisibility](#MemberVisibility)), override, final, S([UnitAttribiteDeclaration](#UnitAttributeDeclaration)), S([UnitRoutineDeclaration](#UnitRoutineDeclaration)), S([InitDeclaration](#InitDeclaration)))

F()

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

1. [MemberVisibility](#MemberVisibility) is valid if provided
2. [UnitAttribiteDeclaration](#UnitAttributeDeclaration) or [UnitRoutineDeclaration](#UnitRoutineDeclaration) or [InitDeclaration](#InitDeclaration) is valid
3. InitDeclaration: **new** [[UnitRoutineParameters](#UnitRoutineParameters)] [[EnclosedUseDirective](#EnclosedUseDirective)] [[RequireBlock](#RequireBlock)]

[“**:**”[UnitTypeName](#UnitTypeName) [[Arguments](#Arguments)] {“**,**” [UnitTypeName](#UnitTypeName) [[Arguments](#Arguments)] }]

([InnerBlock](#InnerBlock) [[EnsureBlock](#EnsureBlock)] BlockEnd)|(**foreign**|**none** [[EnsureBlock](#EnsureBlock) BlockEnd])

S(new)

F()

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

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

S(pure, safe, <ident>, “()”, “:=”)

F()

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

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

S(<ident>, “()”, “:=”, S([OperatorName](#OperatorName)))

F()

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

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

S(alias)

F()

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

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

S([OperatorSign](#OperatorSign))

F()

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

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

S(“**^**”, “**\***”, “**/**”, “**\**”, “**=**”, “**+**”, “**-**“, ”**<**”, ”**>**”, ”**&**”, “**|**”, “#”, “%”, “@”, “!”, “$”, “~”)

F()

**VAL-OPR-SGN**: is always valid

1. ConstObjectsDeclaration: **const** “:”[ [ConstObject](#ConstObject) { “**,**” [ConstObject](#ConstObject)} ] BlockEnd

S(const)

F()

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

1. every [ConstObject](#ConstObject) is valid
2. all [ConstObject](#ConstObject) are unique within the [ConstObjectsDeclaration](#ConstObjectsDeclaration) and surrounding UnitDeclaration
3. ConstObject:

(

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

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

)

|

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

S(S([Constant](#Constant)), <ident>, “{”)

F()

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

1. TBD every is valid
2. RegularExpression: [Constant](#Constant)1 ({“**|**”[Constant](#Constant)2}) | (“**|**””**..**” [Constant](#Constant)n)

S(S([Constant](#Constant)))

F()

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

1. every [Constant](#Constant) is valid
2. TBD
3. Statement: [Assignment](#Assignment) | [LocalAttributeDeclaration](#LocalAttributeCreation)| [WritableCall](#WritableCall) | [ObjectCreation](#ObjectCreation) | [Conditional](#Conditional)| [Loop](#Loop) | [Detach](#Detach)|[Return](#Return) |[HyperBlock](#HyperBlock)| [Raise](#Raise) | [UnpackTuple](#UnpackTuple)

S(S([Assignment](#Assignment)), S([LocalAttributeDeclaration](#LocalAttributeCreation)), S([WritableCall](#WritableCall)), S([ObjectCreation](#ObjectCreation)), S([Conditional](#Conditional)), S([Loop](#Loop)), S([Detach](#Detach)), S([Return](#Return)), S([HyperBlock](#HyperBlock)), S([Raise](#Raise)), S([UnpackTuple](#UnpackTuple)))

F()

**VAL-STMT**: is valid if and only if one of the following <[Assignment](#Assignment)> or <[LocalAttributeDeclaration](#LocalAttributeDeclaration)> or <[WritableCall](#WritableCall)> or <[Conditional](#Conditional)> or <[Loop](#Loop)> or <[Detach](#Detach)> or <[ObjectCreation](#ObjectCreation)> or <[Return](#Return)> or <[UnpackTuple](#UnpackTuple)> or <[HyperBlock](#HyperBlock)> or <[Raise](#Raise)> is valid

1. Detach: **?** [Identifier](#Identifier) [NewLine]

S(“?”)

F()

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

1. [Identifier](#Identifier) is valid
2. [Identifier](#Identifier) is declared, visible, and writable entity
3. Raise:[**active**] **raise** [[Expression](#Expression)] [NewLine]

S(active, raise)

F()

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

1. [Expression](#Expression) is valid if provided
2. If no [Expression](#Expression) provided then Raise textually belongs to whenClasues or Else section
3. Return: **return** [[Expression](#Expression)] [NewLine]

S(return)

F()

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

1. 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.
2. HyperBlock: [[RequireBlock](#RequireBlock)] [InnerBlock](#InnerBlock) [[EnsureBlock](#EnsureBlock)] BlockEnd

S(S([RequireBlock](#RequireBlock)), S([InnerBlock](#InnerBlock)))

F()

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

1. [RequireBlock](#RequireBlock) is valid if provided
2. [InnerBlock](#InnerBlock) is valid
3. [EnsureBlock](#EnsureBlock) is valid if provided
4. Assignment: [Writable](#Writable) “**:=**” [Expression](#Expression) [NewLine]

S(S([Writable](#Writable)))

F()

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

1. <[Writable](#Writable)> and <[Expression](#Expression)> are both valid
2. type of <[Expression](#Expression)> conforms to or converts into the type of <[Writable](#Writable)>
3. Writable: [WritableCall](#WritableCall) | (“**(**”[WritableCall](#WritableCall) {“**,**” [WritableCall](#WritableCall) } “**)**”)

S(S([WritableCall](#WritableCall)), “(”)

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. UnpackTuple: “**(**“ [LocalAttributeNamesList](#LocalAttributeNamesList) “**)**” **is** [Expression](#Expression)

S(“(”)

F()

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

1. [LocalAttributeNamesList](#LocalAttributeNamesList) is valid
2. [Expression](#Expression) is valid
3. LocalAttributeDeclaration:

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

[NewLine]

S(S([LocalAttributeNamesList](#LocalAttributeNamesList)))

F()

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

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

S(var, rigid, <ident>)

F()

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

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

S(const, rigid, <ident>)

F()

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

1. TBD every is valid
2. BooleanExpression: [Expression](#Expression)

S([Expression](#Expression))

F()

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

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

S(<ident>, S([Constant](#Constant)))

F()

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

1. TBD every is valid
2. 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. [RefExpression:](#TupleExpression) **ref** [Expression](#Expression)

S(ref)

F()

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

1. [Expression](#Expression) is valid
2. TBD
3. LambdaExpression: (**rtn** [Identifier](#Identifier) [[Signature](#Signature)])|[InlineLambdaExpression](#InlineLambdaExpression)

S(rtn, S([InlineLambdaExpression](#InlineLambdaExpression)))

F()

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

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

F()

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

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

S(S([Expression](#Expression)))

F()

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

1. TBD every is valid
2. OldExpression: [**old** [Expression](#Expression)](#TupleExpression)

S(old)

F()

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

1. [Expression](#Expression) is valid
2. TBD
3. TupleExpression: “**(**”[[TupleElement](#TupleElement) {“**,**” [TupleElement](#TupleElement)}]“**)**”

S(“(”)

F()

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

1. Every [TupleElement](#TupleElement) is valid
2. TupleElement: [Expression](#Expression)| [Parameter](#Parameter)

S(S([Expression](#Expression)), S([Parameter](#Parameter)))

F()

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

1. [Expression](#Expression) or [Parameter](#Parameter) is valid
2. TBD
3. TypeOfExpression: [Expression](#Expression) **is** (**“?”**| [UnitType](#UnitType) | [AnonymousUnitType](#AnonymousUnitType))

S(S([Expression](#Expression)))

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. Operator: [OperatorName](#OperatorName)|**in**

S(S([OperatorName](#OperatorName)), “in”)

F()

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

1. [OperatorName](#OperatorName) is valid if provided
2. Constant: [[UnitTypeName](#UnitTypeName) “.”]( [StringConstant |](#StringConstant) [CharacterConstant |](#CharacterConstant) [IntegerConstant |](#IntegerConstant) [RealConstant |](#RealConstant) [BooleanConstant](#BooleanConstant) | [BitConstant](#BitConstant) | [Identifier](#Identifier) )

S(“, ‘, <digit>, <ident>)

F()

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

1. TBD every is valid
2. IfExpression:

**if** [Expression](#Expression)1 (**case** [ExpressionAlternatives](#ExpressionAlternatives))|( BlockStart[Expression](#Expression)2)  
{**elsif** [Expression](#Expression)3 (**case** [ExpressionAlternatives](#ExpressionAlternatives))|( BlockStart[Expression](#Expression)4)}  
**else** [Expression](#Expression)5 “}”**Cmod**  
S(if)

F()

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

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

S(S([AlternativeTags](#AlternativeTags)))

F()

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

1. TBD every is valid
2. 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(<ident>, return, this, old)

F()

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

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

S(new)

F()

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

1. TBD every is valid
2. NewExpression: [**new**] [UnitType](#UnitType) [[Arguments](#Arguments)]

S(new, S([UnitType](#UnitType)))

F()

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

1. [UnitType](#UnitType) is valid
2. [Arguments](#Arguments) are valid if provided
3. If [UnitType](#UnitType) has no initialization procedure or has initialization procedure with no parameters then no or empty [Arguments](#Arguments) are provided
4. If [UnitType](#UnitType) has no initialization procedure with non-empty parameters then TBD
5. TBD
6. CallChain: “**.**”([Identifier](#Identifier)|[OperatorName](#OperatorName)) [[Arguments](#Arguments)]

S(“.”)

F()

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

1. TBD every is valid
2. Arguments: “**(**” [[ExpressionList](#ExpressionList)] ”**)**”

S(“(”)

F()

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

1. [ExpressionList](#ExpressionList) is valid if provided
2. TBD
3. ForcedType: GroupStart [UnitType](#UnitType) GroupEnd

S(“{”)

F()

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

1. [UnitType](#UnitType) is valid
2. ExpressionList: [Expression](#Expression) {“**,**” [Expression](#Expression)}

S(S([Expression](#Expression)))

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. Conditional:

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

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

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

S(if)

F()

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

1. [Expression](#Expression) (one or two) is valid and has type Boolean
2. All [StatementsList](#StatementsList)s are valid
3. TBD
4. Alternatives:

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

S(S([AlternativeTags](#AlternativeTags)))

F()

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

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

S(S([AlternativeTag](#AlternativeTag)))

F()

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

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

S(S([Expression](#Expression)))

F()

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

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

F()

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

1. TBD every is valid
2. Loop:

(**while** [BooleanExpression](#BooleanExpression){“,”[BooleanExpression](#BooleanExpression)} [[RequireBlock](#RequireBlock)] [InnerBlock](#InnerBlock)) | ([[RequireBlock](#RequireBlock)] [InnerBlock](#InnerBlock) **while** [BooleanExpression](#BooleanExpression){“,”[BooleanExpression](#BooleanExpression)})

[[EnsureBlock](#EnsureBlock)] BlockEnd

S(while, S([RequireBlock](#RequireBlock)), S([InnerBlock](#InnerBlock)))

F()

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

1. VAL010\_Loop (LV): <[Loop](#Loop)> is valid if and only if it has no while or only one while clause and …
2. Type: [”**?**”] [AttachedType](#AttachedType)

S(“?”, S([AttachedType](#AttachedType)))

F()

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

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

S(S([UnitType](#UnitType)), S([AnchorType](#AnchorType)), S([UnionType](#UnionType)), S([TupleType](#TupleType)), S([RangeType](#RangeType)), S([RoutineType](#RoutineType)), S([AnonymousUnitType](#AnonymousUnitType)))

F()

**VAL-ATCH-TYP**: is valid if and only if [UnitType](#UnitType)| or [AnchorType](#AnchorType) or [UnionType](#UnionType) or [TupleType](#TupleType) or [RangeType](#RangeType) or [RoutineType](#RoutineType) or [AnonymousUnitType](#AnonymousUnitType) is valid

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

S(unit)

F()

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

1. every [MemberDesciption](#memberDescription) is valid
2. all [MemberDesciption](#memberDescription)s are unique within the AnonymousUnitType
3. RoutineType: **rtn** [[Signature](#Signature)]

S(rtn)

F()

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

1. [Signature](#Signature) is valid if provided
2. Signature: (“**(**”[[Type](#Type) {“**,**” [Type](#Type)}]“**)**”[ [ReturnType](#ReturnType) [Type](#Type)])| ([ReturnType](#ReturnType) [Type](#Type))

S(“(”, S([ReturnType](#ReturnType)))

F()

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

1. every [Type](#Type) is valid
2. RangeType: [RangeTypeItem](#RangeTypeItem) {“**,**” [RangeTypeItem](#RangeTypeItem)}

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

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

S(S([ConstantExpression](#ConstantExpression)))

F()

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

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

S(as)

F()

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

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

S(S([UnitType](#UnitType)))

F()

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

1. every [UnitType](#UnitType) is valid
2. all [UnitType](#UnitType)s are unique within the [UnionType](#UnitType)
3. TBD
4. TupleType: “**(**”[[TupleField](#TupleField) {“**,**”|”**;**” [TupleField](#TupleField)}]“**)**”

S(“(”)

F()

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

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

S(<ident>, S([UnitType](#UnitType)))

F()

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

1. every [Identifier](#Identifier) is valid if provided
2. [UnitType](#UnitType) is valid
3. UnitTypeName: {[Identifier](#Identifier)“**.**”} [UnitName](#UnitName) [GenericsStart([Type](#Type)|[ConstantExpression](#ConstantExpression)) {“**,**” ([Type](#Type)|[ConstantExpression](#ConstantExpression))} GenericsEnd ]

S(<ident>)

F()

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

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

S(ref, val, active, <ident>)

F()

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

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

S(<ident>)

**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 semantics: list of all behavioral patterns**

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

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

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)