**ENUMs:**

**unit** Integer **is**

**enum** System.MinInteger **..** System.MaxInteger **end**

plus **alias** + (other: **like this**): **like this**

**init (**other: **like this)**

**private:**

data: Bit [System.IntegerBits]

**end** Integer

**unit** PositiveInteger **inherit** Integer **is**

**enum** 0 **..** System.MaxInteger **end**

**end**

**unit** Character **alias** Char **is**

**enum** System.MinCharacter **..** System.MaxCharacter **end**

**init (**other: **like this)**

**private:**

data: Bit [System.CharacterBits]

**end** Character

**unit** LatinLetter **inherit** Character **is**

**enum** ‘a’ **..** ‘z’, ‘A’ .. ‘Z’ **end**

**end**

**unit** WeekDay **is**

**enum** Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday **end**

**end** WeekDay

Enums are in fact constant object of the unit type. They have all features the unit has. We may declare several enums in one unit only if we name each enum. If enum is unnamed then there is only one enum in the unit.

We may extend enums …

**unit** RGB **is**

**enum** rgb **is** Red, Green, Blue **end**

**end** RGB

**unit** Color **inherit** RGB **is**

**enum extends rgd** Black, Yellow, Magenta **end**

**end** Color

**with** RGB, Color

**var** rgb **is** Red // deduced type is RGB

**var** color: Color **is** Black

// NO this does not work well !!!!!

rgb := color /\* Type of color conforms to type of rgb but assigning Black to rgb is incorrect so, types must be identical if left side … Bad … Need to think more …\*/

**SLang reference manual.**

Version 0.34

**SLang reserved word:**

|  |
| --- |
| **abstract** |
| **as** |
| **break** |
| **case** |
| **concurrent** |
| **const** |
| **else** |
| **elseif** |
| **end** |
| **ensure** |
| **external** |
| **final** |
| **if** |
| **inherit** |
| **init** |
| **invariant** |
| **is** |
| **lambda** |
| **like** |
| **loop** |
| **new** |
| **old** |
| **override** |
| **private** |
| **public** |
| **pure** |
| **ref** |
| **require** |
| **rescue** |
| **result** |
| **retry** |
| **safe** |
| **super** |
| **then** |
| **this** |
| **typeof** |
| **unit** |
| **use** |
| **val** |
| **var** |
| **variant** |
| **while** |

**SLang syntax:**

Compilation : [CompilationUnit](#CompilationUnit) { [CompilationUnit](#CompilationUnit) }

CompialtionUnit  
 : { [UseDirective](#UseDirective) } [AnonymousRoutine](#AnonymousRoutine)  
 | { [UseDirective](#UseDirective) } [StandaloneRoutine](#StandaloneRoutine)  
 | { [UseDirective](#UseDirective) } [UnitDeclaration](#UnitDeclaration)

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

UseElement  
 : [FullUnitName](#FullUnitName) [ **as** [Identifier](#Identifier) ]

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

AnonymousRoutine  
 : [StatementsList](#StatementsList)

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

StandaloneRoutine  
 : [Identifier](#Identifier) [ [GenericInstantiation](#GenericInstantiation) ] [ [Arguments](#Arguments) ] [ “**:**” [Type](#Type) ] “**is**”  
 [ [RequireBlock](#RequireBlock) ]  
 [InternalRoutineBody](#InternalRoutineBody) | [ExternalRoutineBody](#ExternalRoutineBody)  
 [ [EnsureBlock](#EnsureBlock) ] [ [RescueBlock](#RescueBlock) ]  
 **end** [ [Identifier](#Identifier) ]

Arguments : “**(**” [Argument](#Arguemnt) { ”**;**” [Argument](#Arguemnt) } “**)**”

Argument : [Identifier](#Identifier) { “**,**” [Identifier](#Identifier) } “**:**” [Type](#Type)

RescueBlock  
 : **rescue** [StatementsList](#StatementsList)

RequireBlock  
 : **require** [ **else** ] { [Predicate](#Predicate) }

EnsureBlock  
 : **ensure** [ **then** ] { [Predicate](#Predicate) }

Predicate  
 : [ [Label](#Label) “**:**” ] [BooleanExpression](#BooleanExpression)

InternalRoutineBody  
 : [StatementsList](#StatementsList)

ExternalRoutineBody  
 : **external** [StringConstant](#StringConstant) **…<To Be Defined>**

UnitDeclaration  
 : [ **ref** | **val** | **concurrent** | **abstract** ] **unit** [Identifier](#Identifier) [ [FormalGenerics](#FormalGenerics) ]  
 [ [InheritDirective](#InheritDirective) ]  
 [ [UseDirective](#UseDirective) ]  
 { [FeatureSelection](#FeatureSelection) }  
 [ **init** [InitProcedure](#InitProcedure) { “,” [InitProcedure](#InitProcedure) } ]  
 { [FeatureDeclaration](#FeatureDeclaration) | ( **public** | **private** ) “**:**” { [FeatureDeclaration](#FeatureDeclaration) } ) }  
 [ **invariant** { [Predicate](#Predicate) } ]  
 **end** [ [Identifier](#Identifier) ]

InheritDirective  
 : ( **inherit** | “**:**” ) [Parent](#Parent) { “,” [Parent](#Parent) }

Parent : [UnitType](#UnitType) [ “**’**” ]

GenericInstantiation  
 : “**[**” [TypeList](#TypeList) “**]**”

TypeList: [Type](#Type) { “**,**” [Type](#Type) }

FormalGenerics  
 : “**[**” [FormalGeneric](#FormalGeneric) { “**,**” [FormalGeneric](#FormalGeneric) } “**]**”

FormalGeneric  
 : [Identifier](#Identifier) [ “**->**” [UnitType](#UnitType) [ **init** [InitProcedure](#InitProcedure) ] ]

InitProcedure  
 : [ [Identifier](#Identifier) ] [ “**(**” [UnitType](#UnitType) { “**,**” [UnitType](#UnitType) } ”**)**” ]

FeatureSelection  
 : **override** [Parent](#Parent) ”**.**” | ”**->**” [Identifier](#Identifier) { [“**,**”] [Parent](#Parent) ”**.**” |”**->**” [Identifier](#Identifier) }

FeatureDeclaration  
 : [ **override** ] [ **public** | **private** ] [ **final** ]  
 [UnitAttribiteDeclaration](#UnitAttributeDeclaration) | [UnitRoutineDeclaration](#UnitRoutineDeclaration) | [InitDeclaration](#InitDeclaration)

InitDeclaration  
 : **init**” [ [Identifier](#Identifier) ] [ [Arguments](#Arguments) ] “**is**” [ [RequireBlock](#RequireBlock) ]  
 [InternalRoutineBody](#InternalRoutineBody)  
 [ [EnsureBlock](#EnsureBlock) ] [ [RescueBlock](#RescueBlock) ]  
 **end** [  [Identifier](#Identifier) ]

UnitRoutineDeclaration  
 : [ **pure** | **safe** ] [Identifier](#Identifier) [ [Arguments](#Arguments) ] [ “**:**” [Type](#Type) ] “**is**” [ [RequireBlock](#RequireBlock) ]  
 **abstract** | [InternalRoutineBody](#InternalRoutineBody) | [ExternalRoutineBody>](#ExternalRoutineBody)  
 [ [EnsureBlock](#EnsureBlock) ] [ [RescueBlock](#RescueBlock) ]  
 **end** [ [Identifier](#Identifier) ]  
 // Abstract routine must not have <RescueBlock>

Statement  
 : [Assignment](#Assignment)  
 | [LocalAttributeDeclaration](#LocalAttributeDeclaration)  
 | ( [ [Label](#Label) ”**:**” ] [If](#If) )  
 | ( [ [Label](#Label) ”**:**” ] [Case](#Case) )  
 | ( [ [Label](#Label) ”**:**” ] [Loop](#Loop) )  
 | **break** [ [Label](#Label) ]  
 | [FeatureCallOrCreation](#FeatureCallOrCreation) | **retry**  
 | **?** [Identifier](#Identifier) | [Assert](#Assert) | **…<To Be Defined>**

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

Assert : { [ [Label](#Label) ”**:**” ] [Predicate](#Predicate) }

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

LocalAttributeDeclaration  
 : [TypedLocalAttributeDeclaration](#TypedLocalAttributeDeclaration)  
 | ( [ **var** ] [Identifier](#Identifier) [AttributeInitializer](#AttributeInitializer) )

UnitAttributeDeclaration  
 : [TypedUnitAttributeDeclaration](#TypedUnitAttributeDeclaration) | ( [ **const** ] [Identifier](#Identifier) [AttributeInitializer](#AttributeInitializer) [ **invariant** { [Predicate](#Predicate) } ] )

AttributeInitializer  
 : [ “**:**” [Type](#Type) ] **is** [Expression](#Expression)

TypedLocalAttributeDeclaration  
 : [ **var** ] [Identifier](#Identifier) { “**,**” [ **var** ] [Identifier](#Identifier) } “**:**” [Type](#Type)

TypedUnitAttributeDeclaration  
 : [ **const** ] [Identifier](#Identifier) { “**,**” [ **const** ] [Identifier](#Identifier) } “**:**” [Type](#Type)

Writable: **result** | ( [Identifier](#Identifier) [ “**[**” [ExprList](#ExprList) “**]**” ] { “**.**” [Identifier](#Identifier) “**[**” [ExprList](#ExprList) “**]**” } )

ExprList: [Expression](#Expression) { “**,**” [Expression](#Expression) }

BooleanExpression  
 : [Expression](#Expression)

Expression  
 : [IfExpression](#IfExpession) | [CaseExpression](#CaseExpression) | [FeatureCallOrCreation](#FeatureCallOrCreation) | [Expression](#Expression) [Operator](#Operator) [Expression](#Expression)  
 | [Operator](#Operator) [Expression](#Expression)  
 | [Constant](#Constant) | [TypeOfExpression](#TypeOfExpression)  
 | [OldExpression](#OldExpression) | [LambdaExpression](#LambdaExpression) | [TupleExpression](#TupleExpression) | **…<To Be Defined>**

LambdaExpression [[: [](#OldExpression) **[pure](#OldExpression)** [|](#OldExpression) **[safe](#OldExpression)** []](#OldExpression) **[lambda](#OldExpression)**  [[](#OldExpression) [[Arguments](#OldExpression)](#Arguments) [] [ “](#OldExpression)**[:](#OldExpression)**[”](#OldExpression) [[Type](#OldExpression)](#Type) [] “](#OldExpression)**[is](#OldExpression)**[” [](#OldExpression) [[RequireBlock](#OldExpression)](#RequireBlock) []](#OldExpression) [[InternalRoutineBody](#OldExpression)](#InternalRoutineBody) [|](#OldExpression) [[ExternalRoutineBody>](#OldExpression)](#ExternalRoutineBody) [[](#OldExpression) [[EnsureBlock](#OldExpression)](#EnsureBlock) [] [](#OldExpression) [[RescueBlock](#OldExpression)](#RescueBlock) []](#OldExpression) **[end](#OldExpression)**](#TupleExpression)

OldExpression [:](#TupleExpression) **[old](#TupleExpression)** [[Expression](#TupleExpression)](#Expression)

TupleExpression  
 : “**(**” [ [Expression](#Expression) { “**,**” [Expression](#Expression) } ] “**)**”

TypeOfExpression  
 : [Expression](#Expression) **typeof** [UnitType](#UnitType)

Operator: “**+**” | ”**-**” | ”**\***” | ”**/**” | “**~**” | “**!**” | “**&**” | “**=**” | “**/=**” | “**^**” |”**and**” | ”**not**” | ”**or**” | “**xor**”| “**rem**” | “**mod**” **…<To Be Defined>**

Constant: [StringConstant](#StringConstant) | [CharacterConstant](#CharacterConstant) | [IntegerConstant](#IntegerConstant) | [RealConstant](#RealConstant) | [BooleanConstant](#BooleanConstant)

IfExpession  
 : **if** [BooleanExpression](#BooleanExpression) **then** [Statement](#Statement)  
 { **elsif** [BooleanExpression](#BooleanExpression) **then** [Statement](#Statement) }  
 [ **else** [Statement](#Statement) ]  
 **end**

CaseExpression  
 : **case** [ **typeof** ] [Expression](#Expression)  
 **when** [WhenExpression](#WhenExpression) { “**,**” [WhenExpression](#WhenExpression) } **then** [Statement](#Statement) { **when** [WhenExpression](#WhenExpression) { “**,**” [WhenExpression](#WhenExpression) } **then** [Statement](#Statement) }  
 [ **else** [Statement](#Statement) ]  
 **end**

FeatureCallOrCreation  
 : ( [ **new** ] [Identifier](#Identifier) [ [GenericInstantiation](#GenericInstantiation) ] )  
 | ( **this** | **super )** [ [CallChain](#CallChain) ]

CallChain  
 : [ “**.**” [Identifier](#Identifier) ] [ “**(**” [ [Parameters](#Parameters) ] ”**)**” ]

Parameters  
 : [Expression](#Expression) { “**,**” [Expression](#Expression) }

If : **if** [BooleanExpression](#BooleanExpression) **then** [StatementsList](#StatementsList)  
 { **elsif** [BooleanExpression](#BooleanExpression) **then** [StatementsList](#StatementsList) }  
 [ **else** [StatementsList](#StatementsList) ] **end**

Case : **case** [ **typeof** ] [Expression](#Expression)  
 **when** [WhenExpression](#WhenExpression) { “**,**” [WhenExpression](#WhenExpression) } **then** [StatementsList](#StatementsList)  
 { **when** [WhenExpression](#WhenExpression) { “**,**” [WhenExpression](#WhenExpression) } **then** [StatementsList](#StatementsList) }  
 [ **else** [StatementsList](#StatementsList) ]  
 **end**

WhenExpression  
 : [Expression](#Expression) [ “**..**” [Expression](#Expression) ]

Loop : [ **while** [BooleanExpression](#BooleanExpression) ] **loop** [ **invariant** { [Predicate](#Predicate) } ]  
 [StatementsList](#StatementsList)  
 [ **variant** { [Predicate](#Predicate) } ]  
 [ **while** [BooleanExpression](#BooleanExpression) ]  
 **end**

Type : [UnitType](#UnitType) | [AnchorType](#AnchorType) | [MultiType](#MultiType) | ”**?**” [Type](#Type) | [TupleType](#TupleType) | **…<To Be Defined>**

AnchorType  
 : **like** **this** | [Identifier](#Identifier)

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

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

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

UnitType: [ **ref** | **val** | **concurrent** ] [Identifier](#Identifier) [ [GenericInstantiation](#GenericInstantiation) ]

Последующие правила – это лексическая грамматика. Ее не обязательно описывать правилами. Можно, конечно, но в любом случае отдельно от синтаксиса языка.

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

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

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

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

IntegerConstant  
 : [ “**+**”|”**-**“ ] [Digit](#Digit) { [Digit](#Digit) } [ “**x**” “**B**”|”**b**”|”**H**”|”**h**” ]

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

BooleanConstant  
 : **true** | **false**

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

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

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

Symbol : ’**+**’ | **…<To Be Defined>**

**SLang semantics:**

SEM001: <Compilation> is valid is and only if all <CompilationUnit> are valid

SEM002: <CompilationUnit> is valid is and only if

….