# SysADL

## BNF Grammar

### Legend

General Structure: <rule_name> ::= <rule_definition>

Alternative: |

Optional: ?

Multiple: *

### Terminals

```
BOOLEAN_VALUE  ::= 'true' | 'false'

INTEGER_VALUE  ::=
        ('0' | '1'..'9' (('_')? '0'..'9')*) | //DECIMAL
        (('0b' | '0B') '0'..'1' (('_')? '0'..'1')*) | // BINARY
        (('0x'|'0X') ('0'..'9'|'a'..'f'|'A'..'F')  (('_')? ('0'..'9'|'a'..'f'|'A'..'F'))*) | // HEX
        ('0' ('_')? '0'..'7' (('_')? '0'..'7')*) // OCT

ID  ::= ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*  | ('\'' -> '\'')

STRING  ::= '"' ( '\\' ('b'|'t'|'n'|'f'|'r'|'"'|"'"|'\\') | !('\\'|'"') )* '"'

ML_COMMENT  ::= '/*' -> '*/'

SL_COMMENT  ::= '//' !('\\n'|'\\r')* ('\\r'? '\\n')?

CONST ::=
        BOOLEAN_VALUE | INTEGER_VALUE | STRING
```

## Entry Rule

Model ::=

Package

## Package Definition

Package ::=

'Package'

ID

('import' ID ( "," ID)* '' )?

(Property)*

(ElementDef)*

(Requirement)*

## Abstract Defs

ElementDef ::=

ValueTypeDef | DimensionDef | UnitDef | ComponnentDef_Impl | BoundaryComponentDef | ArchitectureDef | CompositePortDef | SimplePortDef | DataTypeDef | SimpleConnectorDef | CompositeConnectorDef | ActivityDef | ConstraintDef | UserDefinedAction | Protocol | Enumeration

TypeDef ::=

ValueTypeDef | DataTypeDef | Enumeration

StructuralDef ::=

ComponnentDef_Impl | BoundaryComponentDef | ArchitectureDef | CompositePortDef | SimplePortDef | SimpleConnectorDef | CompositeConnectorDef

DataDef ::=

ValueTypeDef | DimensionDef | UnitDef | DataTypeDef | Enumeration

BehavioralDef ::=

ActivityDef | UserDefinedAction | Protocol

AllocElement ::=

ComponnentDef_Impl | BoundaryComponentDef | ArchitectureDef | SimpleConnectorDef | CompositeConnectorDef

ProtElement ::=

CompositePortDef | SimplePortDef

## Structural Defs

PortDef ::=

CompositePortDef | SimplePortDef

ComponnentDef ::=

ComponnentDef_Impl | BoundaryComponentDef | ArchitectureDef

ConnectorDef ::=

SimpleConnectorDef | CompositeConnectorDef

ComponnentDef_Impl ::=

'Component' 'Def'

ID

'{'

(Property

| StructuralDef

| DataDef)*

('ports'

PortUse*)

(Configuration)?

'}'

BoundaryComponentDef ::=

'Boundary' 'Component' 'Def'

ID

'{'

(Property

```
        | StructuralDef
        | DataDef
        | PortUse)*
        '}'

ArchitectureDef  ::=
'Architecture'
ID
'{'
        (Property
        | StructuralDef
        | DataDef
        | PortUse)*
        Configuration
        '}'

CompositePortDef  ::=
'Composite' 'Port' 'Def'
ID
'{'
        (Property
        | StructuralDef
        | DataDef
        | PortUse)*
        '}'

SimplePortDef  ::=
'Port' 'Def'
ID
'{'
        'flow' FlowProperty TypeUse
        (Property
        | StructuralDef
```

```
        | DataDef)*
        '}'

SimpleConnectorDef  ::=
'Connector' 'Def'
ID
'{'
        (Property
        | StructuralDef
        | DataDef
        | 'participants' PortUse_Reverse*
        Flow)*

        ('ports' '{' PortUse ( "," PortUse)* '}' )?
        '}'

CompositeConnectorDef  ::=
'Composite' 'Connector' 'Def'
ID
'{'
        (Property
        | StructuralDef
        | DataDef
        | Flow
        )*

        (ConnectorUse*
        | Configuration)
        '}'

enum ConstraintKind ::=
        preCondition = 'pre' | postCondition = 'post' | invariant = 'invariant'
```

## Structural Uses

PortUse ::=

PortUse_Impl | PortProxy

## Behavioral Uses and Statements

ActionUse ::=

ActionUse_Impl | ActionSend | ActionReceive

ActionDef ::= UserDefinedAction

Expression ::= Expression_Impl | Invocation | ValueAccess | RTTI_Impl | Instanceof | HasType | NewInstance | ReferenceAccess | Computation_Impl | Binary | Unary | InjectionExpression | '(' Expression ')'

Statement ::= (ActionUse_Impl | ActionSend | ActionReceive | SimpleInvocation | Inline | FreeInstance | Assign | Break | For_Impl | Fork | Join | While | Return) ';'

InstanceAccess ::=

InstanceAccess_Impl

AccessPart ::=

AccessPart_Impl | IndexAccessPart

Property ::=

'Property'

ID

(':' ID)?

('=' Object)?

Object ::=

CONST

ValueTypeDef ::=

```
        'ValueType'
        ID ('extends' ID)?
        '{'
                ('unit' '=' ID)?
                ('dimension' '=' ID)?
                (Property)*
        '}'

DataTypeDef ::=
        {DataTypeDef}
        'DataType'
        ID
        '{'
                (Property)*
                (TypeUse)*
        '}'

Enumeration ::=
        'Enum'
        ID
        '{'
                (Property)*
                (EnumLiteralValue ( "," EnumLiteralValue)* )?
        '}'

UnitDef ::=
        'Unit'
        ID
        '{'
                ('dimension' '=' ID)?
                (Property)*
        '}'
```

```
DimensionDef ::=
        'Dimension'
        ID
        ('{'
                (Property)*
        '}')?

TypeUse  ::=
        'ID ':' ('flow')? ID
                ('(' Property* ')' )?


EnumLiteralValue  ::=
        ID '=' Int


ActivityDef  ::=
        'Activity' 'Def'
        ID '(' (TypeUse (',' TypeUse)*)? ')' (':' TypeUse)? ('to' ID)?
        '{'
                (Property
                | BehavioralDef
                | DataDef
                | ConstraintUse
                )*
                (Body)?
                PinBinding*
        '}'

ConstraintDef  ::=
        'Constraint' 'Def'
        ID ('(' TypeUse (',' TypeUse)* ')' (':' TypeUse))?
        '{'
```

```
                (Expression '')
                (Property)*
        '}'

UserDefinedAction  ::=
        'Action' 'Def'
        ID
        '(' (TypeUse (',' TypeUse)*)? ')' (':' TypeUse)? '{'
                (Property
                | BehavioralDef
                | DataDef
                | ConstraintUse)*
                (ActionBody)?
        '}'

ActionBody ::=
        Statement
        FlowDelegation


Protocol  ::=
        'Protocol'
        ID ('for' PortUse)?
        '{'
                ProtocolBody
        '}'

ProtocolBody  ::=
        ((PredefinedActions
        | FlowDelegation) '')*


PredefinedActions  ::=
```

```
        ActionSend | ActionReceive


Flow  ::=
        'flow' ID 'from' ID 'to' ID

PortUse_Reverse ::=
        ID ':' '~' ID ('[' Int ',' Int ']')?
                        ('{' Property* '}')?

PortUse_Impl  ::=
        ID ':' ID ('[' Int ',' Int ']')?
                ('{' Property* '}')?

ConnectorBinding  ::=
        ID 'to' ID

PortProxy  ::=
        'Proxy'  ID ':' ID ('[' Int ',' Int ']')?
                ('{' Property* '}')?


Configuration  ::=
        'Configuration'
        '{'
                ('components' ComponentUse*
                | 'connectors' Attachment*
                | 'delegations' Delegation*)
        '}'

ComponentUse  ::=
        ID ':' ID ('[' Int ',' Int ']')?
        ('{'
```

```
                Property*
        '}')?

Delegation  ::=
        ID 'as' ID

Attachment  ::=
        ConnectorUse 'binding' ConnectorBinding ("," ConnectorBinding)*

ConnectorUse  ::=
        ID ':' ID ('[' Int ',' Int ']')?
        ('{'
                Property*
        '}')?

enum FlowProperty  ::=
        in = 'in' | out = 'out'

Body  ::=
                ((Statement
                | InternalFlow
                | FlowDelegation) '')*

ConstraintUse  ::=
        ConstraintKind ID

PinBinding  ::=
        ('bind' ID 'to' ID
        | 'bind' ID 'to' ID)


FlowDelegation  ::=
        'delegate' ID 'as' ID ''
```

```
ActionUse_Impl  ::=
        (ID ':')? ID '(' (Statement (',' Statement)? ')'
        ('{'
                Property*
        '}')? ''

ActionSend  ::=
        'via' ID 'send' ID

ActionReceive  ::=
        'via' ID 'receive' ID


Break  ::=
        'break'

For_Impl  ::=
        'for' ForControl
        '{'
                Statement*
        '}'

ForControl  ::=
        loopVariableDefinition+=LoopVariableDefinition


LoopVariableDefinition  ::=
        LoopTypeUseNoType 'in' Expression ('..' Expression)?
        | LoopTypeUse ':' Expression

LoopTypeUse  ::=
```

```
        ID ID

While  ::=
        'While' '(' Expression ')'
        '{'
                Statement*
        '}'

Return  ::=
        'Return' Expression ''

Assign ::=
        InstanceAccess '=' Expression

InstanceAccess_Impl  ::=
        ID | ID '.' ID

Invocation  ::=
        ID '(' Expression ( "," Expression)* ')'
                ('.' ValueAccess)?

ValueAccess  ::=
        AccessPart ( "." AccessPart)*
        ''

Instanceof  ::=
        ValueAccess '.' 'instanceOf' '(' ID ')'

HasType  ::=
        ValueAccess '.' 'hasType' '(' ID ')'

NewInstance  ::=
        'new' ID
```

```
ReferenceAccess  ::=
        ('refRef')?
        AccessPart ( '->' AccessPart)*

ExpressionFinal  ::=
        InstanceAccess | Unary | '(' ExpressionFinal ')'

Binary  ::=
        ExpressionFinal (('->'| '<->'| '+' | '-'| '*' | '//' | '!=' | '==' | '<' | '>' | '||' | '\&\&')
Expression)*

Unary ::=
        ('-'| '!') ExpressionFinal

ConditionalBlock ::=
        'If' Expression 'then'
        ('{'
                (Statement)*
        '}'
        (ElseBlock)?
        )?

ElseBlock ::=
        'Else'
        '{'
                Statement*
        '}'
```

## Requirements

```
Requirement ::=
        'Requirement' ID '{' STRING '}'
```