SAE International AS5506A (AADL V2) Syntax Cheat Sheet (September 19, 2011)

4.1 AADL Specifications

AADL_specification ::=
  (package_spec | property_set)+

4.2 Packages

package_spec ::=
  package *defining*_package_name
  (**public** package_declarations
    [**private** package_declarations]
    | **private** package_declarations)
  [**properties** ({ property_association }+ |
    none_statement)]
end *defining*_package_name **;**

package_declarations ::=
  { name_visibility_declaration }* { AADL_declaration }*

package_name ::=
  { *package*_identifier **::** }* *package*_identifier

none_statement ::= **none ;**

AADL_declaration ::=
  classifier_declaration | annex_library

classifier_declaration ::=
  component_classifier_declaration
  | feature_group_classifier_declaration

component_classifier_declaration ::=
  component_type | component_type_extension |
  component_implementation
  | component_implementation_extension

feature_group_classifier_declaration ::=
  feature_group_type | feature_group_type_extension

name_visibility_declaration ::=
  import_declaration | alias_declaration

import_declaration ::=
  with (package_name | property_set_identifier)
    { **,** (package_name | *property_set*_identifier) }* **;**

alias_declaration ::=
  (*defining*_identifier **renames package** package_name **;**) |
  ([*defining*_identifier] **renames**
    (component_category unique_component_type_reference |
    feature group unique_feature_group_type_reference) **;**) |
  (**renames** package_name **:: all ;**)

4.3 Component Types

component_type ::=
  component_category *defining_component_type*_identifier
  [**prototypes** ({ prototype }+ | none_statement)]
  [**features** ({ feature }+ | none_statement)]
  [**flows** ({ flow_spec }+ | none_statement)]
  [modes_subclause | requires_modes_subclause]
  [**properties** (
    { *component_type*_property_association
    | contained_property_association }+
    | none_statement)]
  { annex_subclause }*
  end *defining_component_type*_identifier **;**

component_type_extension ::=
  component_category *defining_component_type*_identifier
    extends unique_component_type_reference
    [prototype_bindings]
  [**prototypes** ({ prototype }+ | none_statement)]
  [**features** ({ feature }+ | none_statement)]
  [**flows** ({ flow_spec }+ | none_statement)]
  [modes_subclause | requires_modes_subclause]
  [**properties** (
    { *component_type*_property_association
    | contained_property_association }+
    | none_statement)]
  { annex_subclause }*
  end *defining_component_type*_identifier **;**

component_category ::=
  abstract_component_category
  | software_category
  | execution_platform_category
  | composite_category

abstract_component_category ::= **abstract**

software_category ::=
  data | **subprogram** | **subprogram group** |
  thread | **thread group** | **process**

execution_platform_category ::=
  memory | **processor** | **bus** | **device** |
  virtual processor | **virtual bus**

composite_category ::= **system**

unique_component_type_reference ::=
  [package_name **::**] *component_type*_identifier

4.4 Component Implementations

component_implementation ::=
  component_category **implementation**
  *defining*_component_implementation_name
    [prototype_bindings]
  [**prototypes** ({ prototype }+ | none_statement)]
  [**subcomponents** ({subcomponent}+ | none_statement)]
  [**calls** ({ subprogram_call_sequence }+ | none_statement)]
  [**connections** ({ connection }+ | none_statement)]
  [**flows** ({ flow_implementation |
    end_to_end_flow_spec }+ | none_statement)]
  [modes_subclause]
  [**properties** ({ property_association
    | contained_property_association }+ | none_statement)]
  { annex_subclause }*
  end *defining*_component_implementation_name **;**

component_implementation_name ::=
  *component_type*_identifier **.** *component_implementation*_identifier

component_implementation_extension ::=
  component_category **implementation**
  *defining*_component_implementation_name
  extends unique_component_implementation_reference
    [prototype_bindings]
  [**prototypes** ({ prototype }+ | none_statement)]
  [**subcomponents** ({subcomponent}+ | none_statement)]
  [**calls** ({ subprogram_call_sequence }+ | none_statement)]
  [**connections** ({ connection }+ | none_statement)]
  [**flows** ({ flow_implementation |
    end_to_end_flow_spec }+ | none_statement)]
  [modes_subclause]
  [**properties** ({ property_association
    | contained_property_association }+ | none_statement)]
  { annex_subclause }*
  end *defining*_component_implementation_name **;**

unique_component_implementation_reference ::=
  [package_name **::**] component_implementation_name

4.5 Subcomponents

subcomponent ::=
  *defining_subcomponent*_identifier **:** component_category
  [unique_component_classifier_reference [prototype_bindings]
    | *prototype*_identifier]
  [array_dimensions [array_element_implementation_list]]
  [{ { *subcomponent*_property_association
    | contained_property_association }+ }]
  [component_in_modes] **;**

subcomponent_refinement ::=
  *defining_subcomponent*_identifier **: refined to**
    component_category
  [unique_component_classifier_reference [prototype_bindings]
    | *prototype*_identifier]
  [array_dimensions [array_element_implementation_list]]
  [{ { *subcomponent*_property_association
    | contained_property_association }+ }]
  [component_in_modes] **;**

unique_component_classifier_reference ::=

```
  ( unique_component_type_reference
  | unique_component_implementation_reference )

array_dimensions ::= { array_dimension }+

array_dimension ::= [ [ array_dimension_size ] ]

array_dimension_size ::=
  numeral | unique_property_constant_identifier |
  unique_property_identifier

array_element_implementation_list ::=
  ( unique_component_implementation_reference
      [ prototype_bindings ]
    { , unique_component_implementation_reference
      [ prototype_bindings ] }* )

array_selection_name ::= identifier array_selection

array_selection ::= { [ selection_range ] }+

selection_range ::= numeral [ .. numeral ]
```

4.7 Prototypes

```
prototype ::=
  defining_prototype_identifier :
    ( component_prototype
    | feature_group_type_prototype
    | feature_prototype )
  [ { { prototype_property_association }+ } ] ;

component_prototype ::=
  component_category [ unique_component_classifier_reference ]
  [ [ ] ]

feature_group_type_prototype ::=
  feature group [ unique_feature_group_type_reference ]

feature_prototype ::=
  [ in | out ] feature [unique_component_classifier_reference ]

prototype_refinement ::=
  defining_prototype_identifier : refined to
  ( component_prototype | feature_group_type_prototype
    | feature_prototype )
  [ { { prototype_property_association }+ } ] ;

prototype_bindings ::=
  ( prototype_binding { , prototype_ binding }* )

prototype_binding ::=
  prototype_identifier ⇒
    ( component_prototype_actual
    | component_prototype_actual_list
    | feature_group_type_prototype_actual
    | feature_prototype_actual )

component_prototype_actual ::=
  component_category
    ( unique_component_classifier_reference
      [ prototype_bindings ]
    | prototype_identifier )

component_prototype_actual_list ::=
  ( component_prototype_actual
    { , component_prototype_actual }* )

feature_group_type_prototype_actual ::=
  ( feature group unique_feature_group_type_reference
    [ prototype_bindings ] )
  | ( feature group feature_group_type_prototype_identifier )

feature_prototype_actual ::=
  ( ( in | out | in out )
    ( event | data | event data ) port ) |
  ( ( requires | provides )
    ( bus | data | subprogram group | subprogram )
      access )
    [ unique_component_classifier_reference ] )
  | ( [ in | out ] feature feature_prototype_identifier )
```

4.8 Annex Subclauses and Annex Libraries

```
annex_subclause ::=
  annex annex_identifier (
    ( {** annex_specific_language_constructs **} ) | none )
    [ in_modes ] ;

annex_library ::=
  annex annex_identifier
  ( ( {** annex_specific_reusable_constructs **} ) | none ) ;
```

5.2 Subprograms and Subprogram Calls

```
subprogram_call_sequence ::=
  defining_call_sequence_identifier :
  { { subprogram_call }+ }
    [ { { call_sequence_property_association }+ } ]
    [ in_modes ] ;

subprogram_call ::=
  defining_call_identifier : subprogram called_subprogram
    [ { { subcomponent_call_property_association }+ } ] ;

called_subprogram ::=
  subprogram_unique_component_classifier_reference
  | ( data_unique_component_type_reference
    . data_provides_subprogram_access_identifier )
  | ( subprogram_group_unique_component_type_reference
    . provides_subprogram_access_identifier )
  | ( abstract_unique_component_type_reference
    . provides_subprogram_access_identifier )
  | ( feature_group_identifier
    . requires_subprogram_access_identifier )
  | component_prototype_identifier
  | ( processor . provides_subprogram_access_identifier )
  | subprogram_subcomponent_identifier
  | ( subprogram_group_subcomponent_identifier
    . provides_subprogram_access_identifier )
  | requires_subprogram_access_identifier
  | ( requires_subprogram_group_access_identifier
    . provides_subprogram_access_identifier )
```

8 Features and Shared Access

```
feature ::=
  ( abstract_feature_spec |
  port_spec |
  feature_group_spec |
  subcomponent_access_spec |
  parameter_spec )
  [ array_dimension ]
  [ { { feature_contained_property_association }+ } ] ;

subcomponent_access_spec ::=
  subprogram_access_spec | subprogram_group_access_spec
  | data_access_spec | bus_access_spec

feature_refinement ::=
  abstract_feature_refinement
  port_refinement |
  feature_group_refinement |
  subcomponent_access_refinement |
  parameter_refinement
  [ array_dimension ]
  [ { { feature_contained_property_association }+ } ] ;

subcomponent_access_refinement ::=
  subprogram_access_refinement |
  subprogram_group_access_refinement
  | data_access_refinement | bus_access_refinement
```

8.1 Abstract Features

```
abstract_feature_spec ::=
  defining_abstract_feature_identifier :
  [ in | out ] feature [ feature_prototype_identifier ]

abstract_feature_refinement ::=
  ( defining_abstract_feature_identifier : refined to
    [ in | out ] feature [ feature_prototype_identifier ] )
  | port_refinement | feature_group_refinement
  | subcomponent_access_refinement | parameter_refinement
```

8.2 Feature Groups and Feature Group Types

```
feature_group_type ::=
  feature group defining_identifier
  [ prototypes ( { prototype }+ | none_statement ) ]
  [ features { feature }+ ]
  [ inverse of unique_feature_group_type_reference ]
  [ properties
    ( { feature_group_contained_property_association }+
```

```
    | none_statement ) ]
  { annex_subclause }*
  end defining_identifier ;

feature_group_type_extension ::=
  feature group defining_identifier
  extends unique_feature_group_type_reference
    [ prototype_bindings ]
  [ prototypes ( { prototype | prototype_refinement }+
    | none_statement ) ]
  [ features { feature | feature_refinement }+ ]
  [ inverse of unique_feature_group_type_reference ]
  [ properties
    ( { feature_group_contained_property_association }+
    | none_statement ) ]
  { annex_subclause }*
  end defining_identifier ;

feature_group_spec ::=
  defining_feature_group_identifier :
  [ in | out ] feature group
    [ [ inverse of ]
      ( unique_feature_group_type_reference
      | prototype_identifier ) ]

feature_group_refinement ::=
  defining_feature_group_identifier : refined to
    [ in | out ] feature group
    [ [ inverse of ]
      ( unique_feature_group_type_reference
      | prototype_identifier ) ]

unique_feature_group_type_reference ::=
  [ package_name :: ] feature_group_type_identifier
```

8.3 Ports

```
port_spec ::=
  defining_port_identifier : ( in | out | in out ) port_type

port_refinement ::=
  defining_port_identifier : refined to
    ( in | out | in out ) port_type

port_type ::=
  data port [ data_unique_component_classifier_reference
    | data_component_prototype_identifier ]
  | event data port
    [ data_unique_component_classifier_reference
    | data_component_prototype_identifier ]
  | event port
```

8.4 Subprogram and Subprogram Group Access

```
subprogram_access_spec ::=
  defining_subprogram_access_identifier :
    ( provides | requires ) subprogram access
    [ subprogram_unique_component_classifier_reference
    | prototype_identifier ]

subprogram_access_refinement ::=
  defining_subprogram_access_identifier : refined to
    ( provides | requires ) subprogram access
    [ subprogram_unique_component_classifier_reference
    | prototype_identifier ]

subprogram_group_access_spec ::=
  defining_subprogram_group_access_identifier :
    ( provides | requires ) subprogram group access
    [ subprogram_group_unique_component_classifier_reference
    | prototype_identifier ]

subprogram_group_access_refinement ::=
  defining_subprogram_group_access_identifier : refined to
    ( provides | requires ) subprogram group access
    [ subprogram_group_unique_component_classifier_reference
    | prototype_identifier ]
```

8.5 Subprogram Parameters

```
parameter_spec ::=
  defining_parameter_identifier :
    ( in | out | in out ) parameter
    [ data_unique_component_classifier_reference |
    prototype_identifier ]

parameter_refinement ::=
  defining_parameter_identifier : refined to
    ( in | out | in out ) parameter
    [ data_unique_component_classifier_reference |
    prototype_identifier ]
```

8.6 Data Component Access

```
data_access_spec ::=
  defining_data_component_access_identifier :
    ( provides | requires ) data access
    [ data_unique_component_classifier_reference
    | prototype_identifier ]

data_access_refinement ::=
  defining_data_component_access_identifier : refined to
    ( provides | requires ) data access
    [ data_unique_component_classifier_reference
    | prototype_identifier ]
```

8.7 Bus Component Access

```
bus_access_spec ::=
  defining_data_component_access_identifier :
    ( provides | requires ) bus access
    [ bus_unique_component_classifier_reference
    | prototype_identifier ]

bus_access_refinement ::=
  defining_data_component_access_identifier : refined to
    ( provides | requires ) bus access
    [ bus_unique_component_classifier_reference
    | prototype_identifier ]
```

9 Connections

```
connection ::=
  defining_connection_identifier :
  ( feature_connection
  | port_connection
  | parameter_connection
  | access_connection
  | feature_group_connection )
  [ { { property_association }+ } ]
  [ in_modes_and_transitions ] ;

connection_refinement ::=
  defining_connection_identifier : refined to
  ( feature_connection_refinement
  | port_connection_refinement
  | parameter_connection_refinement
  | access_connection_refinement
  | feature_group_connection_refinement )
  [ { { property_association }+ } ]
  [ in_modes_and_transitions ] ;
```

9.1 Feature Connections

```
feature_connection ::=
  feature source_feature_reference connection_symbol
  destination_feature_reference

connection_symbol ::=
  directional_connection_symbol
  | bidirectional_connection_symbol

directional_connection_symbol ::= →

bidirectional_connection_symbol ::= ↔

feature_reference ::=
  component_type_feature_identifier |
  component_type_feature_group_identifier . feature_identifier |
  subcomponent_identifier . feature_identifier |
  subprogram_call_identifier . feature_identifier

feature_connection_refinement ::= feature
```

9.2 Port Connections

```
port_connection ::=
  port
    source_port_connection_reference
      connection_symbol
    destination_port_connection_reference

port_connection_refinement ::= port

port_connection_reference ::=
  component_type_port_identifier
```

| *subcomponent*_identifier . *port*_identifier
| *component_type_feature_group*_identifier
. *element_port*_identifier
| *component_type_port*_identifier
. *data_subcomponent*_identifier
| *component_type_requires_data_access*_identifier
| *data_subcomponent*_identifier
| *subcomponent*_identifier
. *provides_data_access*_identifier
| *component_type_feature_group*_identifier
. *element_data_access*_identifier
| *data_subcomponent*_identifier
. *data_subcomponent*_identifier
| **processor** . *processor_port*_identifier
| **self** . *event_or_event_data_source*_identifier

9.3 Parameter Connections

parameter_connection ::=
parameter *source*_parameter_reference
directional_connection_symbol
*destination*_parameter_reference

parameter_connection_refinement ::= **parameter**

parameter_reference ::=
*component_type_parameter*_identifier
[. *data_subcomponent*_identifier]
| *subprogram_call*_identifier . *parameter*_identifier
| *component_type_port*_identifier
[. *data_subcomponent*_identifier]
| *data_subcomponent*_identifier
| *requires_data_access*_identifier
| *component_type_feature_group*_identifier
. *element_data_access*_identifier
| *component_type_feature_group*_identifier
. *element_port_or_parameter*_identifier

9.4 Access Connections

access_connection ::=
[**bus** | **subprogram** | **subprogram group** | **data**]
access
*source*_access_reference connection_symbol
*destination*_access_reference

access_connection_refinement ::=
[**bus** | **subprogram** | **subprogram group** | **data**]
access

access_reference ::=
*requires_access*_identifier | *provides_access*_identifier
| *feature_group*_identifier . *requires_access*_identifier
| *feature_group*_identifier . *provides_access*_identifier
| *subcomponent*_identifier . *provides_access*_identifier
| *subcomponent*_identifier . *requires_access*_identifier
| *subprogram_call*_identifier
. *requires_or_provides_access*_identifier
| *subcomponent*_identifier
| **processor** . *provides_subprogram_access*_identifier

9.5 Feature Group Connections

feature_group_connection ::=
feature group *source*_feature_group_reference
connection_symbol *destination*_feature_group_reference

feature_group_connection_refinement ::= **feature group**

feature_group_reference ::=
*component_type_feature_group*_identifier
| *subcomponent*_identifier . *feature_group*_identifier
| *component_type_feature_group*_identifier
. *element_feature_group*_identifier

10.1 Flow Specifications

flow_spec ::=
flow_source_spec | flow_sink_spec | flow_path_spec

flow_spec_refinement ::=
flow_source_spec_refinement | flow_sink_spec_refinement
| flow_path_spec_refinement

flow_source_spec ::=
*defining_flow*_identifier : **flow source** *out*_flow_feature_identifier
[{ { property_association }$^+$ }] [in_modes] ;

flow_sink_spec ::=
*defining_flow*_identifier : **flow sink** *in*_flow_feature_identifier
[{ { property_association }$^+$ }] [in_modes] ;

flow_path_spec ::=
*defining_flow*_identifier : **flow path**
*in*_flow_feature_identifier →
*out*_flow_feature_identifier
[{ { property_association }$^+$ }] [in_modes] ;

flow_source_spec_refinement ::=
*defining_flow*_identifier : **refined to flow source**
(({ { property_association }$^+$ }
[in_modes_and_transitions])
| in_modes_and_transitions) ;

flow_sink_spec_refinement ::=
*defining_flow*_identifier : **refined to flow sink**
(({ { property_association }$^+$ }
[in_modes_and_transitions])
| in_modes_and_transitions) ;

flow_path_spec_refinement ::=
*defining_flow*_identifier : **refined to flow path**
(({ { property_association }$^+$ }
[in_modes_and_transitions])
| in_modes_and_transitions) ;

flow_feature_identifier ::=
*feature*_identifier | *feature_group*_identifier
| *feature_group*_identifier . *feature*_identifier
| *feature_group*_identifier . *feature_group*_identifier

10.2 Flow Implementations

flow_implementation ::=
(flow_source_implementation
| flow_sink_implementation
| flow_path_implementation)
[{ { property_association }$^+$ }]
[in_modes_and_transitions] ;

flow_source_implementation ::=
*flow*_identifier : **flow source**
{ subcomponent_flow_identifier → *connection*_identifier → }*
*out*_flow_feature_identifier

flow_sink_implementation ::=
*flow*_identifier : **flow sink**
*in*_flow_feature_identifier
{ → *connection*_identifier → subcomponent_flow_identifier }*

flow_path_implementation ::=
*flow*_identifier : flow path
*in*_flow_feature_identifier
[{ → *connection*_identifier → subcomponent_flow_identifier }$^+$
→ *connection*_identifier]
→ *out*_flow_feature_identifier

subcomponent_flow_identifier ::=
(*subcomponent*_identifier [. *flow_spec*_identifier])
| data_component_reference

data_component_reference ::=
*data_subcomponent*_identifier | *requires_data_access*_identifier
| *provides_data_access*_identifier

10.3 End-To-EndFlows

end_to_end_flow_spec ::=
*defining_end_to_end_flow*_identifier : **end to end flow**
*start*_subcomponent_flow_or_etef_identifier
{ → connection_identifier
→ *flow_path*_subcomponent_flow_or_etef_identifier }$^+$
[{ { property_association }$^+$ }]
[in_modes_and_transitions] ;

end_to_end_flow_spec_refinement ::=
*defining_end_to_end*_identifier :
refined to end to end flow
({ { property_association }$^+$ }
[in_modes_and_transitions]
| in_modes_and_transitions) ;

subcomponent_flow_or_etef_identifier ::=
subcomponent_flow_identifier | *end_to_end_flow*_identifier

11.1 Property Sets

```
property_set ::=
  property set defining_property_set_identifier is
    { import_declaration }*
    { property_type_declaration
      | property_definition_declaration
      | property_constant }*
  end defining_property_set_identifier ;
```

11.1.1 Property Types

```
property_type_declaration ::=
  defining_property_type_identifier : type property_type ;

property_type ::=
  aadlboolean | aadlstring
  | enumeration_type | units_type
  | number_type | range_type
  | classifier_type
  | reference_type
  | record_type

enumeration_type ::=
  enumeration ( defining_enumeration_literal_identifier
    { , defining_enumeration_literal_identifier }* )

units_type ::= units units_list

units_list ::=
  ( defining_unit_identifier
    { , defining_unit_identifier ⇒
    unit_identifier * numeric_literal }* )

number_type ::=
  aadlreal [ real_range ] [ units units_designator ]
  | aadlinteger [ integer_range ] [ units units_designator ]

units_designator ::=
  units_unique_property_type_identifier | units_list

real_range ::=
  real_lower_bound .. real_upper_bound

real_lower_bound ::= signed_aadlreal_or_constant
real_upper_bound ::= signed_aadlreal_or_constant

integer_range ::=
  integer_lower_bound .. integer_upper_bound

integer_lower_bound ::= signed_aadlinteger_or_constant

integer_upper_bound ::= signed_aadlinteger_or_constant

signed_aadlreal_or_constant ::=
  ( signed_aadlreal | [ sign ] real_property_constant_term )

signed_aadlinteger_or_constant ::=
  ( signed_aadlinteger | [ sign ] integer_property_constant_term )

sign ::= + | −

signed_aadlinteger ::=
  [ sign ] integer_literal [ unit_identifier ]

signed_aadlreal ::=
  [ sign ] real_literal [ unit_identifier ]

range_type ::=
  range of number_type
  | range of number_unique_property_type_identifier

classifier_type ::=
  classifier
  [ ( classifier_category_reference
    { , classifier_category_reference }* ) ]

classifier_category_reference ::=
  classifier_qualified_meta_model_identifier

qualified_meta_model_identifier ::=
  [ { annex_identifier }** ] meta_model_class_identifier

meta_model_class_identifier ::= { identifier }+

reference_type ::=
  reference [ { ( reference_category
    { , reference_category }* ) ]

reference_category ::=
  named_element_qualified_meta_model_identifier

unique_property_type_identifier::=
  [ property_set_identifier :: ] property_type_identifier

record_type ::=
  record ( record_field { record_field }* )

record_field ::=
  defining_field_identifier : [ list of ] property_type_designator ;

property_type_designator ::=
  unique_property_type_identifier | property_type
```

11.1.2 Property Definitions

```
property_definition_declaration ::=
  defining_property_name_identifier :
    [ inherit ]
    ( single_valued_property | multi_valued_property )
    applies to ( property_owner { , property_owner }* ) ;

single_valued_property ::=
  property_type_designator [ ⇒ default_property_expression ]

multi_valued_property ::=
  { list of }+ property_type_designator
  [ ⇒ default_property_list_value ]

property_owner ::=
  named_element_qualified_meta_model_identifier |
  unique_classifier_reference

unique_classifier_reference ::=
  unqiue_component_classifier_reference
  | unqiue_feature_group_type_refererence
```

11.1.3 Property Constants

```
property_constant ::=
  single_valued_property_constant | multi_valued_property_constant

single_valued_property_constant ::=
  defining_property_constant_identifier : constant
    property_type_designator
  ⇒ constant_property_expression ;

multi_valued_property_constant ::=
  defining_property_constant_identifier : constant { list of }+
    property_type_designator ⇒ constant_property_list_value ;

unique_property_constant_identifier::=
  [ property_set_identifier :: ] property_constant_identifier
```

11.3 Property Associations

```
property_association ::=
  unique_property_identifier
  ( ⇒ | +⇒ )
    [ constant ] assignment
    [ in_binding ] ;

contained_property_association ::=
  unique_property_identifier
  ⇒ [ constant ] assignment
  applies to contained_model_element_path
    { , contained_model_element_path }*
  [ in_binding ] ;

unique_property_identifier ::= [ property_set_identifier :: ]
  property_name_identifier

 contained_model_element_path ::=
  ( contained_model_element { . contained_model_element }*
    [ annex_path ] )
  | annex_path

contained_model_element ::=
  named_element_identifier |
  named_element_array_selection_name

annex_path ::=
  annex annex_identifier {** annex_specific_path **}
```

annex_specific_path ::= *defined by annex*

assignment ::= property_value | modal_property_value

modal_property_value ::=
({ property_value in_modes , }* property_value in_modes)

property_value ::= single_property_value | property_list_value

single_property_value ::= property_expression

property_list_value ::=
([(property_list_value | property_expression)
{ , (property_list_value | property_expression) }*])

in_binding ::=
in binding (platform_classifier_reference
{ , platform_classifier_reference }*)

platform_classifier_reference ::=
*processor*_unique_component_classifier_reference
| *virtual_processor*_unique_component_classifier_reference
| *bus*_unique_component_classifier_reference
| *virtual_bus*_unique_component_classifier_reference
| *memory*_unique_component_classifier_reference

11.4 Property Expressions

property_expression ::=
boolean_term | real_term | integer_term | string_term
| enumeration_term | unit_term | real_range_term
| integer_range_term | property_term
| component_classifier_term | reference_term
| record_term | computed_term

boolean_term ::=
boolean_value | *boolean*_property_constant_term
| not boolean_term | boolean_term and boolean_term
| boolean_term or boolean_term | (boolean_term)

boolean_value ::= true | false

real_term ::= signed_aadlreal_or_constant

integer_term ::= signed_aadlinteger_or_constant

string_term ::= string_literal | *string*_property_constant_term

enumeration_term ::=
*enumeration*_identifier | *enumeration*_property_constant_term

unit_term ::=
*unit*_identifier | *unit*_property_constant_term

integer_range_term ::=
integer_term .. integer_term [delta integer_term]
| *integer_range*_property_constant_term

real_range_term ::=
real_term .. real_term [delta real_term]
| *real_range*_property_constant_term

property_term ::=
[*property_set*_identifier ::] *property_name*_identifier

property_constant_term ::=
[*property_set*_identifier ::] *property_constant*_identifier

component_classifier_term ::=
classifier (
(unique_component_type_reference |
unique_component_implementation_reference))

reference_term ::= reference (contained_model_element_path)

record_term ::=
(*record_field*_identifier ⇒ property_value ;
{ *record_field*_identifier ⇒ property_value ; }*)

computed_term ::= compute (*function*_identifier)

12 Modes and Mode Transitions

modes_subclause ::=
modes ({ mode | mode_transition }+ | none_statement)

requires_modes_subclause ::=
requires modes ({ mode }+ | none_statement)

mode ::=
*defining_mode*_identifier : [initial] mode
[{ { *mode*_property_association }+ }] ;

mode_transition ::=
[*defining_mode_transition*_identifier :]
*source_mode*_identifier
-[mode_transition_trigger
{ , mode_transition_trigger }*]→
*destination_mode*_identifier
[{ { *mode_transition*_property_association }+ }] ;

mode_transition_trigger ::=
unique_port_identifier | self . *event_source*_identifier
| processor . *port*_identifier

unique_port_identifier ::=
[*subcomponent_feature_group_or_subprogram_call*_identifier .]
*port*_identifier

in_modes ::=
in modes (*mode*_identifier { , *mode*_identifier }*)

component_in_modes ::=
in modes (mode_name { , mode_name }*)

mode_name ::= *local_mode*_identifier
[⇒ *subcomponent_mode*_identifier]

in_modes_and_transitions ::=
in modes ((mode_or_transition { , mode_or_transition }*)

mode_or_transition ::=
*mode*_identifier | *mode_transition*_identifier

15 Lexical Elements

character ::= graphic_character | format_effector
| other_control_character

graphic_character ::= identifier_letter | digit
| space_character | special_character

identifier ::= identifier_letter { [underline] letter_or_digit }*

letter_or_digit ::= identifier_letter | digit

numeric_literal ::= integer_literal | real_literal

integer_literal ::= decimal_integer_literal | based_integer_literal

real_literal ::= decimal_real_literal

decimal_integer_literal ::= numeral [positive_exponent]

decimal_real_literal ::= numeral . numeral [exponent]

numeral ::= digit { [underline] digit }*

exponent ::= E [+] numeral | E numeral

positive_exponent ::= E [+] numeral

based_integer_literal ::= base # based_numeral #
[positive_exponent]

base ::= digit [digit]

based_numeral ::= extended_digit { [underline] extended_digit }*

extended_digit ::= digit | A | B | C | D | E | F
| a | b | c | d | e | f

string_literal ::= " { string_element }* "

string_element ::= "" | non_quotation_mark_graphic_character

comment ::= -- { non_end_of_line_character }*