www.axini.com

AML Quick Reference

```
external channel_name [, type: :duplex]
internal channel_name
...
process(process_name) {
  timeout <decimal_value>
  channel(channel_name) {
    stimulus label_name ,
        param_name  => <type> ,
        ...
    response label_name, ...
  }

  var var_name, <type> [, <initial_value>]
  ...
  # behavior of process: AML statements
}

process(process_name) {
  ...
}
```

channel_name has to be declared globally

AML constructs are separated by newlines or ;

```
include 'model_part.aml'
```

can be used anywhere

```
<type>

:integer
:decimal
:boolean
:string
:date
:time
```

```
[ <type> ]                          list
{ <type> => <type> }                hash
{
  field_name => <type>,             struct
  ...
}

Set[<val>, ...]                     enumeration
```

all label options are optional

$$\frac{\texttt{send}}{\texttt{receive}}$$ *label_name*

```
,          on: channel_name
,   expedited: <boolean>
,      urgent: <boolean>
,       after: <time_expr>  |  <decimal_expr>
,      before: <time_expr>  |  <decimal_expr>
,  constraint: <bool_expr>
,      update: <assignment_expr>
,        note: <string>  |  [ <string>* ]
```

```
<statements> ::= <statement>
   | { <statement> ((<newline> | ;) <statement>)* }
```

```
choice {
  o   <statements>
  o   <statements>
  ...
}
```

```
repeat {
  o   <statements>
  o   <statements>
  ...
}
```

```
state state_name
goto state_name
```

```
stop_repetition
next_repetition
```

deterministic choice

```
_if <bool_expr>,
_then { ... } [,
_else { ... } ]
```

deterministic loop

```
_while(<bool_expr>) {
  ... # body
}
```

```
optionally { <statements> }
```

internal constraint

```
constraint <bool expr>
```

internal action

```
update <assignment_expr>
```

Ruby code

```
def method(params)
  ...
end
```

```
if <ruby_expr>
  # AML fragments
else
  # AML fragments
end
```

```
"...#{<ruby_expr>}..."
```

```
                         :terminating
behavior(behavior_name,  ─────────────────
                         :non_terminating
   [, [param_name => <type>, ...]] [, <return_type>] ) {
  # AML statements
}
```

terminating behaviors return value with `exit_with`

```
call terminating_name [, [<expr>, ...]] [, into:<var_name>]
behave_as non_terminating_name [, [<expr>, ...]]
```

```
function(func_name, [<type>, ...] => <type>) {|p, ...|
  # Ruby code - no side effects
}
```

functions can only be used within AML expressions: constraints or updates

`fixed font`	keywords
**_name*	string, e.g. “Notify”
<*_expr>	string, e.g. “x > 5”
<foo>	grammar placeholder
...	repetition / placeholder
[...]	optionally

version 0.2.1, 20-Apr-2022

Example of AML model

```
ABBR_LENGTH = 10

# Abbreviate a string.
function(‘abbreviate,
    [:string] => :string) {|msg|
  msg[0..ABBR_LENGTH]
}
```

functions.aml

```
channel('external') {
  stimulus 'SubscribeRequest',
      _SubscribeParams
  response 'SubscribeResponse',
      _SubscribeParams
  response 'Notify’,
      _NotifyParams
  stimulus 'Renew'
}
```

labels.aml

root model

```
include 'configuration.aml'
include 'macros.aml'
include 'functions.aml'

external 'external'
process('subscription') {
  timeout 10.0
  include 'labels.aml'
  include 'behaviors.aml'

  var 'last_sequence_nr', :integer, -1
  var 'abbreviated_message', :string
  var 'renew_value', :integer
  var 'deadline', :time

  call 'protocol negotiation',
                  [], into: 'renew_value'
  update 'deadline = clock + renew_value'

  repeat {
    o {
      send 'Notify',
          constraint: 'sequence_nr == last_sequence_nr + 1',
              update: 'last_sequence_nr = sequence_nr;
                       abbreviated_message = abbreviate(message)',
                note: '$abbreviated_message #blue'
    }

if config(:enable_renew)
    o {
      receive 'Renew', expedited: true, before: 'deadline',
          update: 'deadline = clock + renew_value'
    }
end
  }
}
```

Variables are usually stored in a separate file, as well.

Ruby as macro preprocessor to include/exclude parts.

Ruby macros

```
def config(key)
  @config.fetch(key)
end

def _SubscribeParams
  { 'address' => :string,
    'renew' => :integer }
end

def _NotifyParams
  { 'message' => :string,
    'sequence_nr' => :integer }
end

def _Subscribe(address, renew)
  "address == '#{address}' &&
   renew == #{renew}"
end
```

macros.aml

```
behavior('protocol negotiation', :terminating, [], :integer) {
  var 'renew_val', :integer
  choice {
    config(:renew_values).each do |value|
      o {
        receive 'SubscribeRequest',
            constraint: _Subscribe(ADDRESS, value)

        send 'SubscribeResponse',
            constraint: "address == '#{ADDRESS}'",
                update: 'renew_val = renew',
                  note: '$renew_val #red'
      }
    end
  }
  exit_with 'renew_val'
}
```

Using Ruby’s each to generate AML fragments.

String interpolation

State variable in a note.

behaviors.aml

```
ADDRESS = ‘www.axini.com’

@config = {
  renew_values: [10, 30],
  enable_renew: true
}
```

configuration.aml

version 0.1
20-Apr-2022