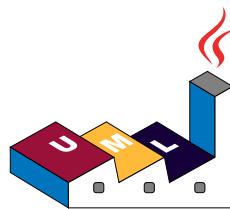


# Drawing UML with PlantUML



## PlantUML Language Reference Guide

(Version 1.2021.2)

**PlantUML** is a component that allows to quickly write :

- Sequence diagram
- Usecase diagram
- Class diagram
- Object diagram
- Activity diagram
- Component diagram
- Deployment diagram
- State diagram
- Timing diagram

The following non-UML diagrams are also supported:

- JSON Data
- YAML Data
- Network diagram (nwdiag)
- Wireframe graphical interface
- Archimate diagram
- Specification and Description Language (SDL)
- Ditaa diagram
- Gantt diagram
- MindMap diagram
- Work Breakdown Structure diagram
- Mathematic with AsciiMath or JLaTeXMath notation
- Entity Relationship diagram

Diagrams are defined using a simple and intuitive language.

# 1 Sequence Diagram

## 1.1 Basic examples

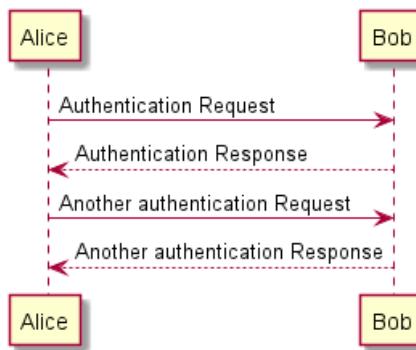
The sequence `->` is used to draw a message between two participants. Participants do not have to be explicitly declared.

To have a dotted arrow, you use `-->`

It is also possible to use `<-` and `<--`. That does not change the drawing, but may improve readability. Note that this is only true for sequence diagrams, rules are different for the other diagrams.

```
@startuml
Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response
```

```
Alice -> Bob: Another authentication Request
Alice <-- Bob: Another authentication Response
@enduml
```



## 1.2 Declaring participant

If the keyword `participant` is used to declare a participant, more control on that participant is possible.

The order of declaration will be the (default) **order of display**.

Using these other keywords to declare participants will **change the shape** of the participant representation:

- `actor`
- `boundary`
- `control`
- `entity`
- `database`
- `collections`
- `queue`

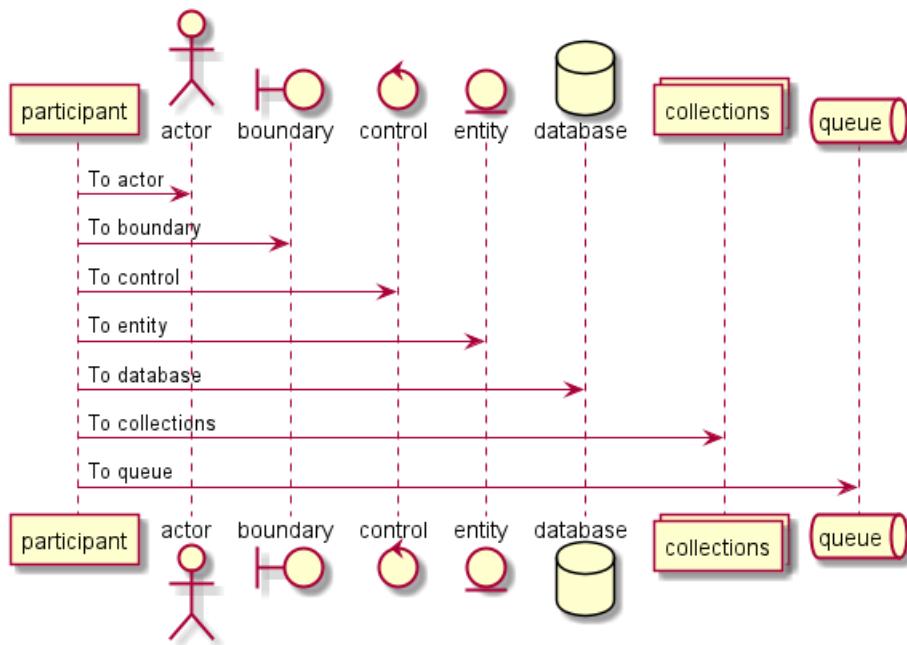
```
@startuml
participant participant as Foo
actor      actor      as Foo1
boundary   boundary   as Foo2
control    control    as Foo3
entity     entity     as Foo4
database   database   as Foo5
collections collections as Foo6
queue     queue     as Foo7
Foo -> Foo1 : To actor
```



```

Foo -> Foo2 : To boundary
Foo -> Foo3 : To control
Foo -> Foo4 : To entity
Foo -> Foo5 : To database
Foo -> Foo6 : To collections
Foo -> Foo7: To queue
@enduml

```



Rename a participant using the `as` keyword.

You can also change the background color of actor or participant.

```

@startuml
actor Bob #red
' The only difference between actor
'and participant is the drawing
participant Alice
participant "I have a really\nlong name" as L #99FF99
/' You can also declare:
    participant L as "I have a really\nlong name" #99FF99
    /

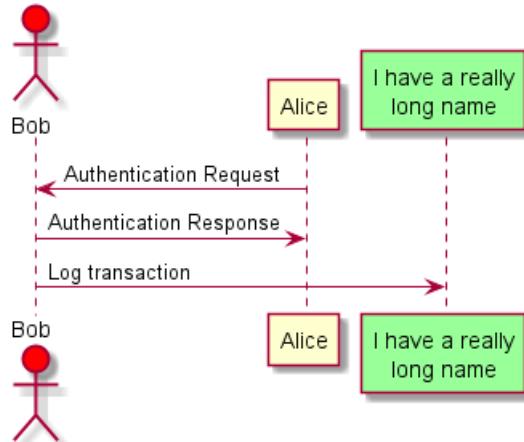
```

```

Alice->Bob: Authentication Request
Bob->Alice: Authentication Response
Bob->L: Log transaction
@enduml

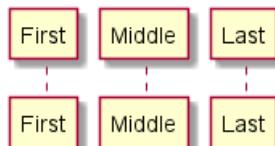
```





You can use the `order` keyword to customize the display order of participants.

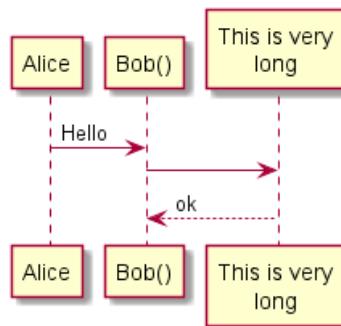
```
@startuml
participant Last order 30
participant Middle order 20
participant First order 10
@enduml
```



### 1.3 Use non-letters in participants

You can use quotes to define participants. And you can use the `as` keyword to give an alias to those participants.

```
@startuml
Alice -> "Bob()" : Hello
"Bob()" -> "This is very\lnlong" as Long
' You can also declare:
' "Bob()" -> Long as "This is very\lnlong"
Long --> "Bob()" : ok
@enduml
```



### 1.4 Message to Self

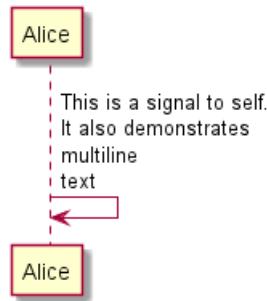
A participant can send a message to itself.

It is also possible to have multi-line using .

```
@startuml
```



```
Alice->Alice: This is a signal to self.\nIt also demonstrates\nmultiline \ntext
@enduml
```

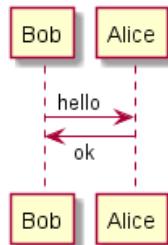


## 1.5 Text alignment

### 1.5.1 Text of response message below the arrow

You can put the text of the response message below the arrow, with the `skinparam responseMessageBelowArrow true` command.

```
@startuml
skinparam responseMessageBelowArrow true
Bob -> Alice : hello
Alice -> Bob : ok
@enduml
```



**TODO:** TODO Link to Text Alignment on skinparam page.

## 1.6 Change arrow style

You can change arrow style by several ways:

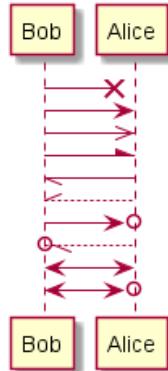
- add a final `x` to denote a lost message
- use `\` or `/` instead of `<` or `>` to have only the bottom or top part of the arrow
- repeat the arrow head (for example, `>>` or `//`) head to have a thin drawing
- use `--` instead of `-` to have a dotted arrow
- add a final `"o"` at arrow head
- use bidirectional arrow `<->`

```
@startuml
Bob ->x Alice
Bob -> Alice
Bob ->> Alice
Bob -\ Alice
Bob \\- Alice
Bob //-- Alice
```

```
Bob ->o Alice
Bob o\\-- Alice
```



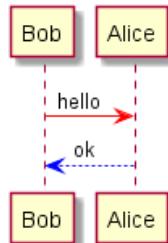
```
Bob <-> Alice
Bob <->o Alice
@enduml
```



## 1.7 Change arrow color

You can change the color of individual arrows using the following notation:

```
@startuml
Bob -[#red]> Alice : hello
Alice -[#0000FF]->Bob : ok
@enduml
```



## 1.8 Message sequence numbering

The keyword `autonumber` is used to automatically add number to messages.

```
@startuml
autonumber
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response
@enduml
```



You can specify a startnumber with `autonumber //start//`, and also an increment with `autonumber //start// //increment//`.

```
@startuml
autonumber
Bob -> Alice : Authentication Request
```



```

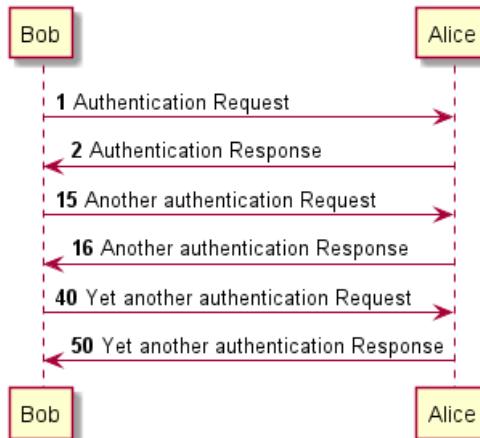
Bob <- Alice : Authentication Response

autonumber 15
Bob -> Alice : Another authentication Request
Bob <- Alice : Another authentication Response

autonumber 40 10
Bob -> Alice : Yet another authentication Request
Bob <- Alice : Yet another authentication Response

@enduml

```



You can specify a format for your number by using between double-quote.

The formatting is done with the Java class `DecimalFormat` (0 means digit, # means digit and zero if absent).

You can use some html tag in the format.

```

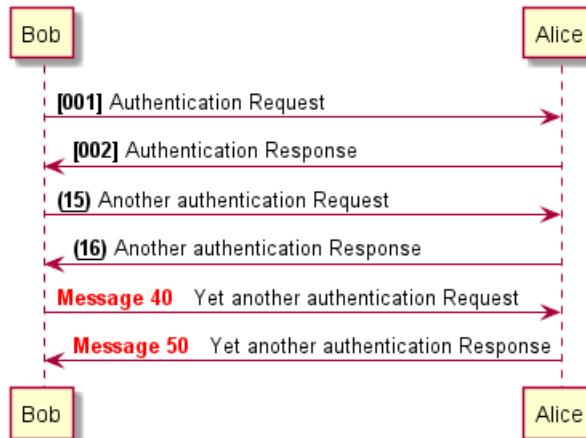
@startuml
autonumber "<b>[000]"
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response

autonumber 15 "<b>(<u>##</u>)"
Bob -> Alice : Another authentication Request
Bob <- Alice : Another authentication Response

autonumber 40 10 "<font color=red><b>Message 0  ">
Bob -> Alice : Yet another authentication Request
Bob <- Alice : Yet another authentication Response

@enduml

```



You can also use `autonumber stop` and `autonumber resume //increment// //format//` to respectively pause and resume automatic numbering.

```

@startuml
autonumber 10 10 "<b>[000]</b>"
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response

autonumber stop
Bob -> Alice : dummy

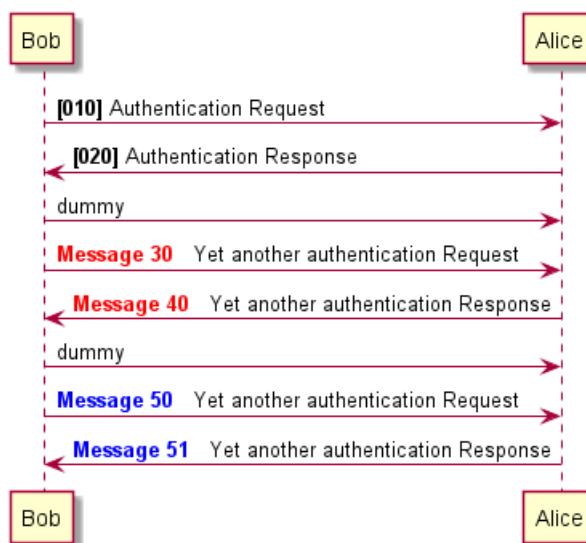
autonumber resume "<font color=red><b>Message 0 </b></font>"
Bob -> Alice : Yet another authentication Request
Bob <- Alice : Yet another authentication Response

autonumber stop
Bob -> Alice : dummy

autonumber resume 1 "<font color=blue><b>Message 0 </b></font>"
Bob -> Alice : Yet another authentication Request
Bob <- Alice : Yet another authentication Response
@enduml
  
```

This UML code defines a sequence diagram with the following steps:

- The sequence starts with Bob sending an 'Authentication Request' to Alice, followed by Alice's 'Authentication Response'.
- Bob sends a 'dummy' message to Alice.
- Alice sends a message labeled 'Message 0' to Bob.
- Bob sends another 'dummy' message to Alice.
- Alice sends a message labeled 'Message 0' to Bob.
- Bob sends another 'dummy' message to Alice.
- Alice sends a message labeled 'Message 0' to Bob.



## 1.9 Page Title, Header and Footer

The `title` keyword is used to add a title to the page.



Pages can display headers and footers using `header` and `footer`.

```
@startuml
```

```
header Page Header
footer Page %page% of %lastpage%

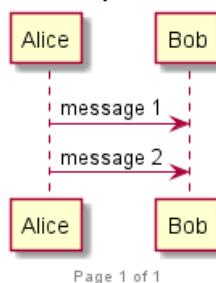
title Example Title

Alice -> Bob : message 1
Alice -> Bob : message 2
```

```
@enduml
```

Page Header

### Example Title



Page 1 of 1

## 1.10 Splitting diagrams

The `newpage` keyword is used to split a diagram into several images.

You can put a title for the new page just after the `newpage` keyword. This title overrides the previously specified title if any.

This is very handy with *Word* to print long diagram on several pages.

(Note: this really does work. Only the first page is shown below, but it is a display artifact.)

```
@startuml
```

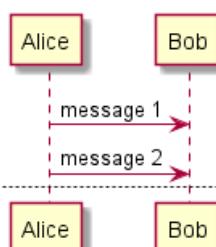
```
Alice -> Bob : message 1
Alice -> Bob : message 2
```

```
newpage
```

```
Alice -> Bob : message 3
Alice -> Bob : message 4
```

```
newpage A title for the\last page
```

```
Alice -> Bob : message 5
Alice -> Bob : message 6
@enduml
```



## 1.11 Grouping message

It is possible to group messages together using the following keywords:

- alt/else
- opt
- loop
- par
- break
- critical
- group, followed by a text to be displayed

It is possible to add a text that will be displayed into the header (for group, see next paragraph '*Secondary group label*').

The end keyword is used to close the group.

Note that it is possible to nest groups.

```
@startuml
Alice -> Bob: Authentication Request

alt successful case

    Bob -> Alice: Authentication Accepted

else some kind of failure

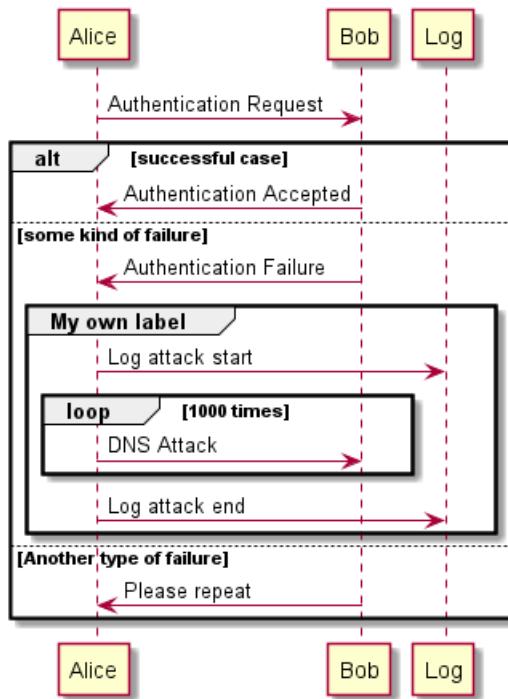
    Bob -> Alice: Authentication Failure
    group My own label
        Alice -> Log : Log attack start
        loop 1000 times
            Alice -> Bob: DNS Attack
        end
        Alice -> Log : Log attack end
    end

else Another type of failure

    Bob -> Alice: Please repeat

end
@enduml
```





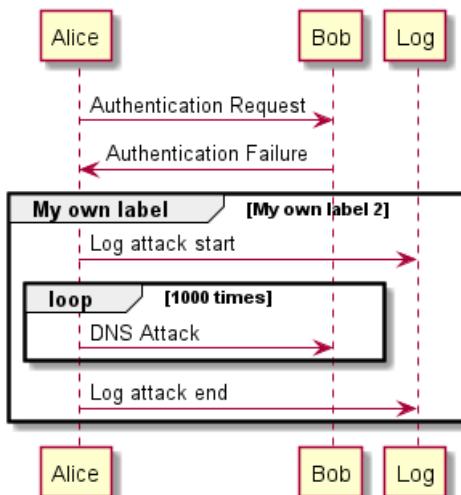
## 1.12 Secondary group label

For group, it is possible to add, between [ and ], a secondary text or label that will be displayed into the header.

```

@startuml
Alice -> Bob: Authentication Request
Bob -> Alice: Authentication Failure
group My own label [My own label 2]
    Alice -> Log : Log attack start
    loop 1000 times
        Alice -> Bob: DNS Attack
    end
    Alice -> Log : Log attack end
end
@enduml

```



[Ref. QA-2503]



### 1.13 Notes on messages

It is possible to put notes on message using the `note left` or `note right` keywords *just after the message*.

You can have a multi-line note using the `end note` keywords.

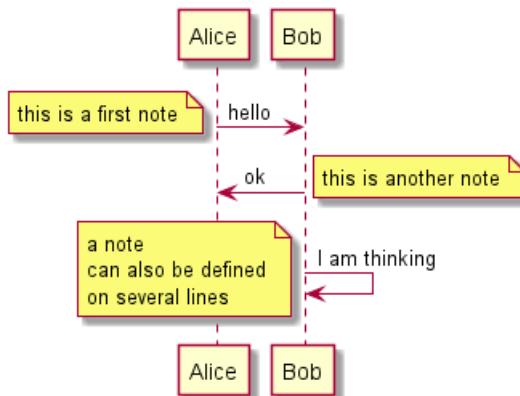
```
@startuml
Alice->Bob : hello
note left: this is a first note
```

```
Bob->Alice : ok
note right: this is another note
```

```
Bob->Bob : I am thinking
```

```
note left
a note
can also be defined
on several lines
end note
```

```
@enduml
```



### 1.14 Some other notes

It is also possible to place notes relative to participant with `note left of` , `note right of` or `note over` keywords.

It is possible to highlight a note by changing its background color.

You can also have a multi-line note using the `end note` keywords.

```
@startuml
participant Alice
participant Bob
note left of Alice #aqua
This is displayed
left of Alice.
end note
```

```
note right of Alice: This is displayed right of Alice.
```

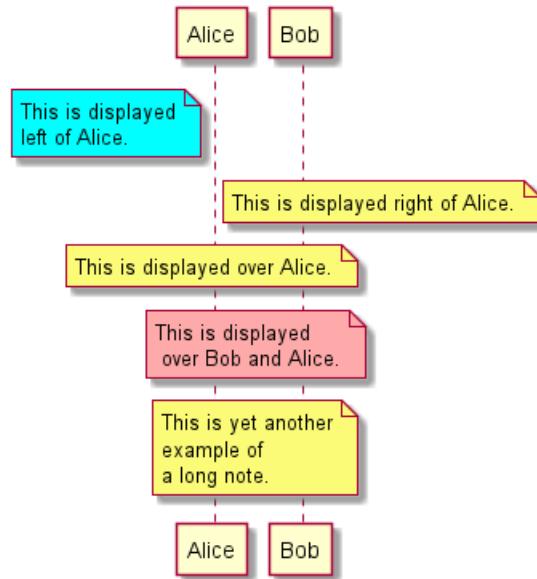
```
note over Alice: This is displayed over Alice.
```

```
note over Alice, Bob #FFAAAA: This is displayed\n over Bob and Alice.
```

```
note over Bob, Alice
This is yet another
example of
a long note.
```



```
end note
@enduml
```



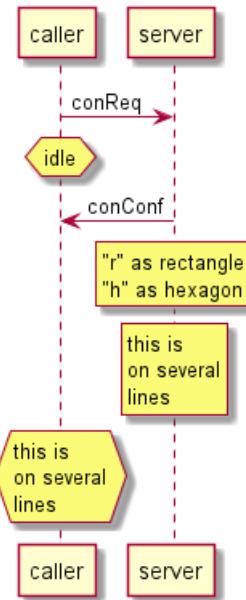
## 1.15 Changing notes shape [hnote, rnote]

You can use `hnote` and `rnote` keywords to change note shapes :

- `hnote` for hexagonal note;
- `rnote` for rectangle note.

```
@startuml
caller -> server : conReq
hnote over caller : idle
caller <- server : conConf
rnote over server
  "r" as rectangle
  "h" as hexagon
endrnote
rnote over server
  this is
  on several
  lines
endrnote
hnote over caller
  this is
  on several
  lines
endhnote
@enduml
```





[Ref. QA-1765]

## 1.16 Note over all participants [across]

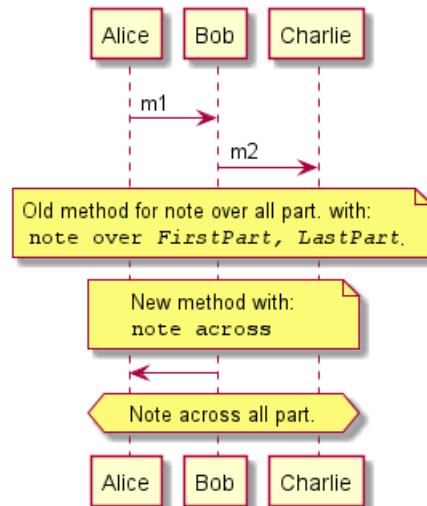
You can directly make a note over all participants, with the syntax:

- note across: note\_description

```

@startuml
Alice->Bob:m1
Bob->Charlie:m2
note over Alice, Charlie: Old method for note over all part. with:\n ""note over //FirstPart, LastPart"
note across: New method with:\n""note across""
Bob->Alice
hnote across:Note across all part.
@enduml

```



[Ref. QA-9738]

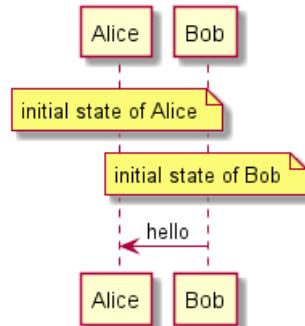
## 1.17 Several notes aligned at the same level [/]

You can make several notes aligned at the same level, with the syntax /:

- without / (by default, the notes are not aligned)

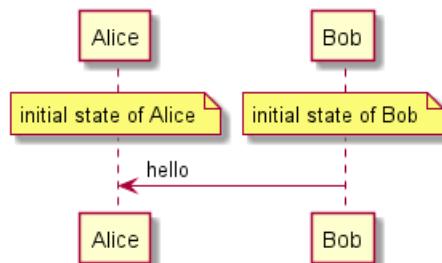


```
@startuml
note over Alice : initial state of Alice
note over Bob : initial state of Bob
Bob -> Alice : hello
@enduml
```



- with / (the notes are aligned)

```
@startuml
note over Alice : initial state of Alice
/ note over Bob : initial state of Bob
Bob -> Alice : hello
@enduml
```



[Ref. QA-354]

## 1.18 Creole and HTML

It is also possible to use creole formatting:

```
@startuml
participant Alice
participant "The **Famous** Bob" as Bob

Alice -> Bob : hello --there--
... Some ~~long delay~~ ...
Bob -> Alice : ok
note left
    This is **bold**
    This is //italics//
    This is ""monospaced"""
    This is --stroked--
    This is __underlined__
    This is ~~waved~~
end note

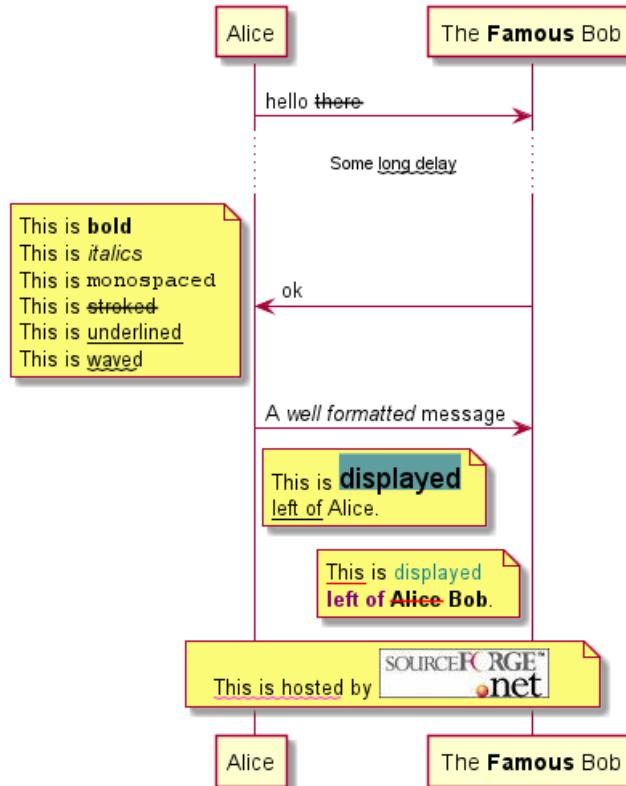
Alice -> Bob : A //well formatted// message
note right of Alice
    This is <back:cadetblue><size:18>displayed</size></back>
    __left of__ Alice.
```



```

end note
note left of Bob
<u:red>This</u> is <color #118888>displayed</color>
**<color purple>left of</color> <s:red>Alice</strike> Bob**.
end note
note over Alice, Bob
<w:#FF33FF>This is hosted</w> by <img sourceforge.jpg>
end note
@enduml

```



## 1.19 Divider or separator

If you want, you can split a diagram using == separator to divide your diagram into logical steps.

```
@startuml
```

```
== Initialization ==
```

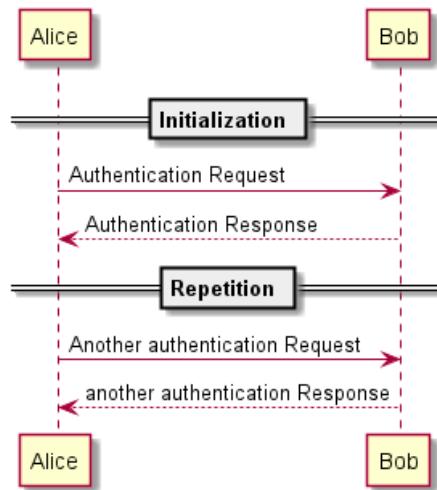
```
Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response
```

```
== Repetition ==
```

```
Alice -> Bob: Another authentication Request
Alice <-- Bob: another authentication Response
```

```
@enduml
```





## 1.20 Reference

You can use reference in a diagram, using the keyword `ref over`.

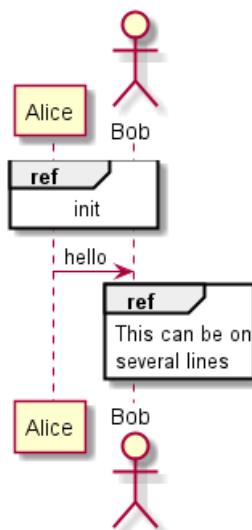
```

@startuml
participant Alice
actor Bob

ref over Alice, Bob : init

Alice -> Bob : hello

ref over Bob
  This can be on
  several lines
end ref
@enduml
  
```



## 1.21 Delay

You can use `...` to indicate a delay in the diagram. And it is also possible to put a message with this delay.

```
@startuml
```

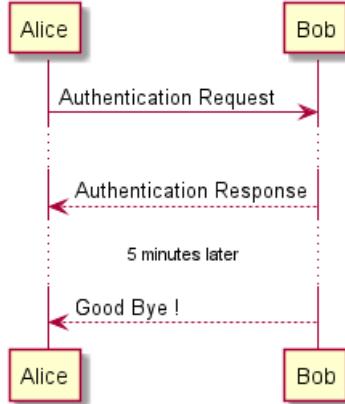


```

Alice -> Bob: Authentication Request
...
Bob --> Alice: Authentication Response
...5 minutes later...
Bob --> Alice: Good Bye !

```

@enduml



## 1.22 Text wrapping

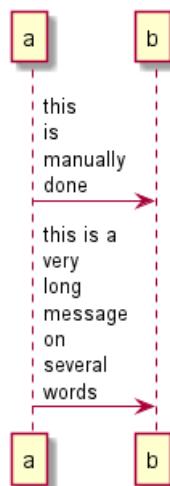
To break long messages, you can manually add `\n` in your text.

Another option is to use `maxMessageSize` setting:

```

@startuml
skinparam maxMessageSize 50
participant a
participant b
a -> b :this\nis\nmanually\ndone
a -> b :this is a very long message on several words
@enduml

```



## 1.23 Space

You can use `|||` to indicate some spacing in the diagram.

It is also possible to specify a number of pixel to be used.

```
@startuml
```



```

Alice -> Bob: message 1
Bob --> Alice: ok
|||
Alice -> Bob: message 2
Bob --> Alice: ok
||45||
Alice -> Bob: message 3
Bob --> Alice: ok

```

@enduml



## 1.24 Lifeline Activation and Destruction

The `activate` and `deactivate` are used to denote participant activation.

Once a participant is activated, its lifeline appears.

The `activate` and `deactivate` apply on the previous message.

The `destroy` denote the end of the lifeline of a participant.

@startuml  
participant User

```

User -> A: DoWork
activate A

A -> B: << createRequest >>
activate B

B -> C: DoWork
activate C
C --> B: WorkDone
destroy C

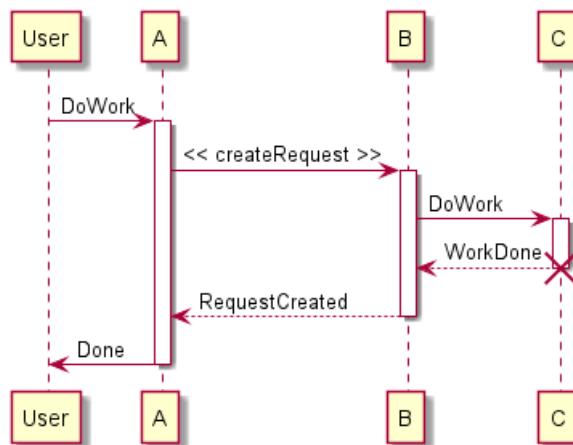
B --> A: RequestCreated
deactivate B

A -> User: Done
deactivate A

```

@enduml





Nested lifeline can be used, and it is possible to add a color on the lifeline.

```
@startuml
participant User
```

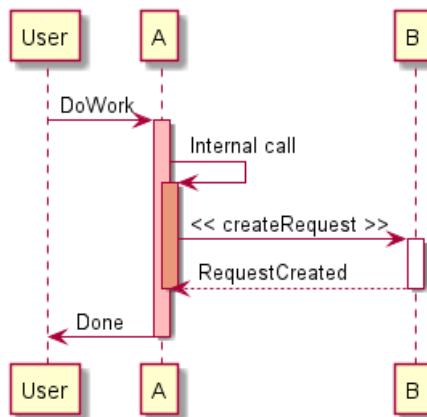
```
User -> A: DoWork
activate A #FFBBBB
```

```
A -> A: Internal call
activate A #DarkSalmon
```

```
A -> B: << createRequest >>
activate B
```

```
B --> A: RequestCreated
deactivate B
deactivate A
A -> User: Done
deactivate A
```

```
@enduml
```



Autoactivation is possible and works with the return keywords:

```
@startuml
autoactivate on
alice -> bob : hello
bob -> bob : self call
bill -> bob #005500 : hello from thread 2
bob -> george ** : create
return done in thread 2
```

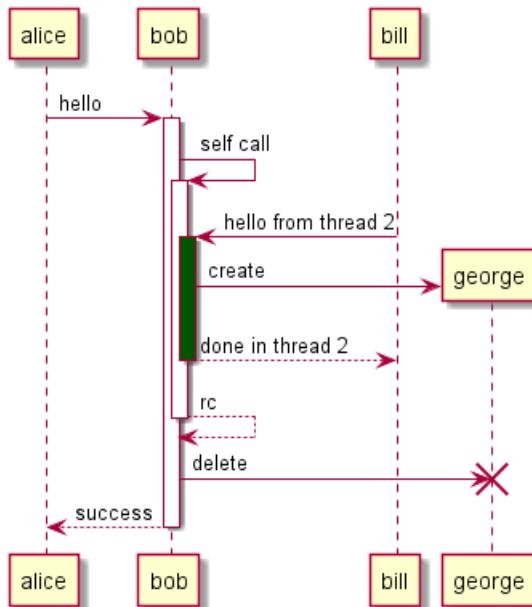


```

return rc
bob -> george !! : delete
return success

@enduml

```



## 1.25 Return

Command **return** generates a return message with optional text label.

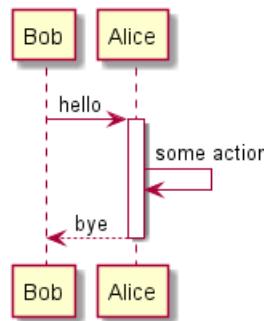
The return point is that which caused the most recent life-line activation.

The syntax is **return label** where **label** if provided is any string acceptable for conventional messages.

```

@startuml
Bob -> Alice : hello
activate Alice
Alice -> Alice : some action
return bye
@enduml

```



## 1.26 Participant creation

You can use the **create** keyword just before the first reception of a message to emphasize the fact that this message is actually *creating* this new object.

```

@startuml
Bob -> Alice : hello

```



```

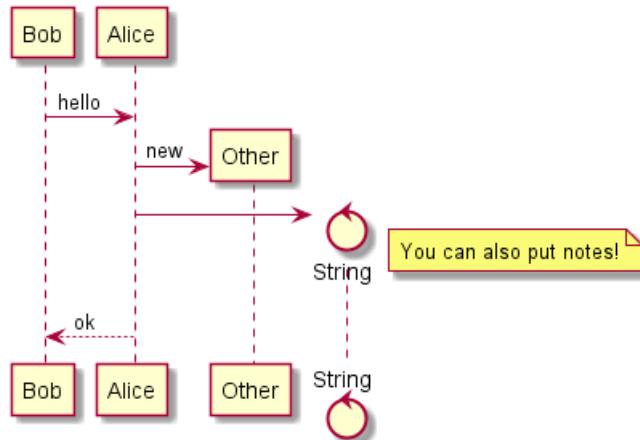
create Other
Alice -> Other : new

create control String
Alice -> String
note right : You can also put notes!

Alice --> Bob : ok

@enduml

```



## 1.27 Shortcut syntax for activation, deactivation, creation

Immediately after specifying the target participant, the following syntax can be used:

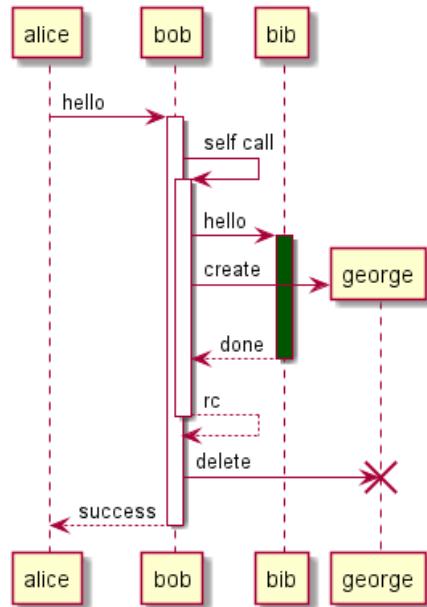
- **++** Activate the target (optionally a #color may follow this)
- **--** Deactivate the source
- **\*\*** Create an instance of the target
- **!!** Destroy an instance of the target

```

@startuml
alice -> bob ++ : hello
bob -> bob ++ : self call
bob -> bib ++ #005500 : hello
bob -> george ** : create
return done
return rc
bob -> george !! : delete
return success
@enduml

```

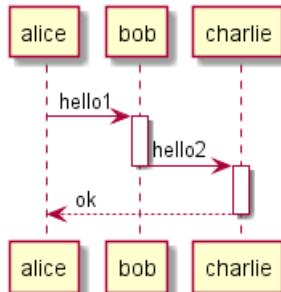




Then you can mix activation and deactivation, on same line:

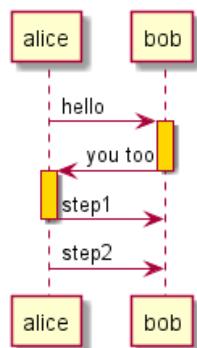
```

@startuml
alice -> bob ++ : hello1
bob -> charlie ---++ : hello2
charlie --> alice -- : ok
@enduml
  
```



```

@startuml
@startuml
alice -> bob ---++ #gold: hello
bob -> alice ---++ #gold: you too
alice -> bob --: step1
alice -> bob : step2
@enduml
@enduml
  
```



[Ref. QA-4834, QA-9573 and QA-13234]

## 1.28 Incoming and outgoing messages

You can use incoming or outgoing arrows if you want to focus on a part of the diagram.

Use square brackets to denote the left "[" or the right "]" side of the diagram.

```
@startuml
[-> A: DoWork
```

```
activate A
```

```
A -> A: Internal call
```

```
activate A
```

```
A ->] : << createRequest >>
```

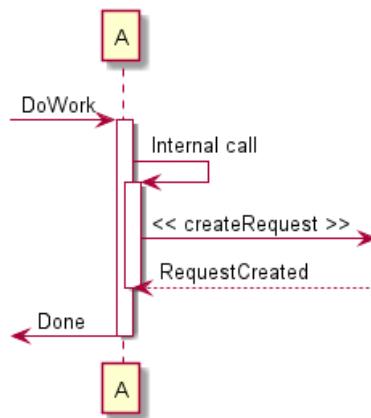
```
A<--] : RequestCreated
```

```
deactivate A
```

```
[<- A: Done
```

```
deactivate A
```

```
@enduml
```



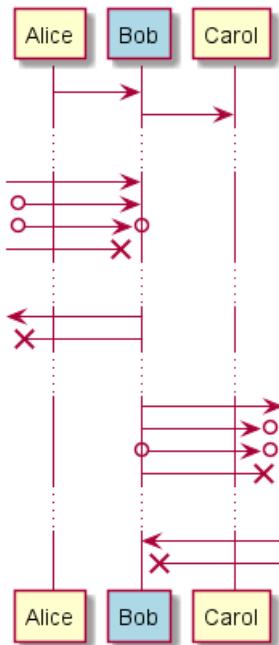
You can also have the following syntax:

```
@startuml
participant Alice
participant Bob #lightblue
Alice -> Bob
Bob -> Carol
...
[-> Bob
[o-> Bob
[o->o Bob
[x-> Bob
...
[<- Bob
[x<- Bob
...
Bob ->]
Bob ->o]
Bob o->o]
Bob ->x]
...
Bob <-]
```



```
Bob x<-]
```

```
@enduml
```

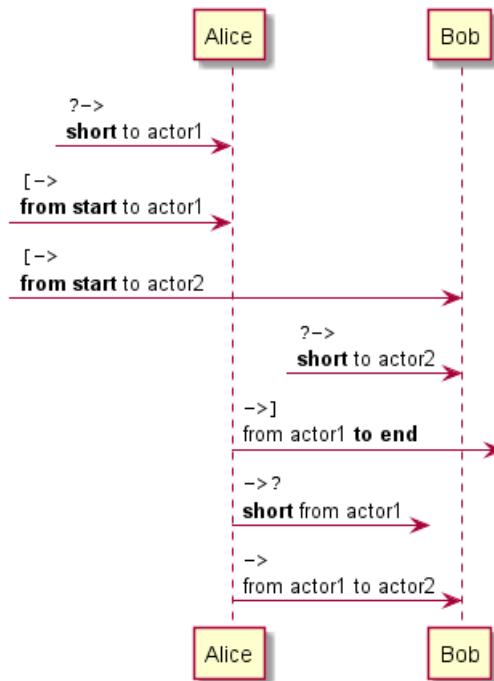


## 1.29 Short arrows for incoming and outgoing messages

You can have short arrows with using ?.

```
@startuml
?-> Alice : ""?->""\n**short** to actor1
[-> Alice : ""[->""\n**from start** to actor1
[-> Bob : ""[->""\n**from start** to actor2
?-> Bob : ""?->""\n**short** to actor2
Alice ->] : ""->]""\nfrom actor1 **to end**
Alice ->? : ""->?""\n**short** from actor1
Alice -> Bob : ""->"" \nfrom actor1 to actor2
@enduml
```





[Ref. QA-310]

## 1.30 Anchors and Duration

With `teoz` usage it is possible to add anchors to the diagram and use the anchors to specify duration time.

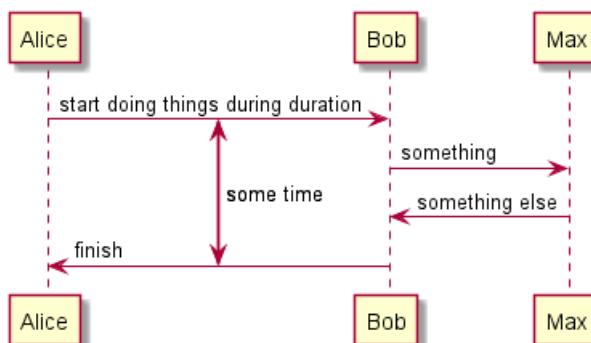
```

@startuml
!pragma teoz true

{start} Alice -> Bob : start doing things during duration
Bob -> Max : something
Max -> Bob : something else
{end} Bob -> Alice : finish

{start} <-> {end} : some time
  
```

@enduml



## 1.31 Stereotypes and Spots

It is possible to add stereotypes to participants using `<>` and `>>`.

In the stereotype, you can add a spotted character in a colored circle using the syntax `(X,color)`.

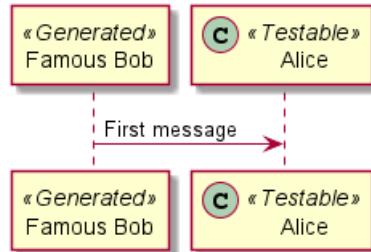
@startuml



```
participant "Famous Bob" as Bob << Generated >>
participant Alice << (C,#ADD1B2) Testable >>
```

Bob->Alice: First message

@enduml



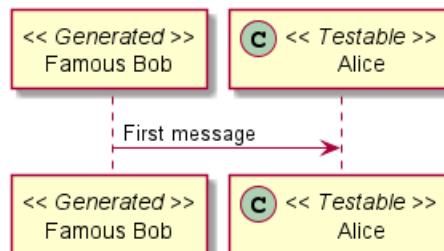
By default, the *guillemet* character is used to display the stereotype. You can change this behaviour using the `skinparam guillemet`:

@startuml

```
skinparam guillemet false
participant "Famous Bob" as Bob << Generated >>
participant Alice << (C,#ADD1B2) Testable >>
```

Bob->Alice: First message

@enduml

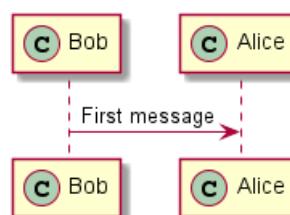


@startuml

```
participant Bob << (C,#ADD1B2) >>
participant Alice << (C,#ADD1B2) >>
```

Bob->Alice: First message

@enduml



## 1.32 More information on titles

You can use creole formatting in the title.



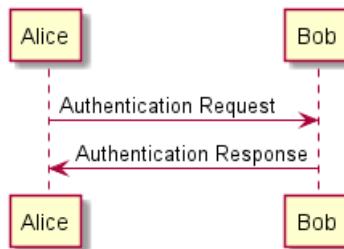
```
@startuml

title __Simple__ **communication** example

Alice -> Bob: Authentication Request
Bob -> Alice: Authentication Response

@enduml
```

### Simple communication example



You can add newline using `\n` in the title description.

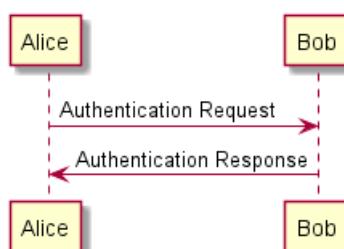
```
@startuml

title __Simple__ communication example\non several lines

Alice -> Bob: Authentication Request
Bob -> Alice: Authentication Response

@enduml
```

### Simple communication example on several lines



You can also define title on several lines using `title` and `end title` keywords.

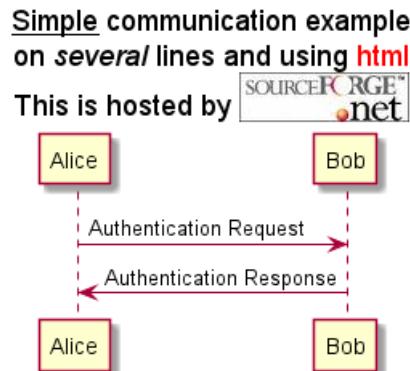
```
@startuml

title
<u>Simple</u> communication example
on <i>several</i> lines and using <font color=red>html</font>
This is hosted by <img:sourceforge.jpg>
end title

Alice -> Bob: Authentication Request
Bob -> Alice: Authentication Response

@enduml
```





### 1.33 Participants encompass

It is possible to draw a box around some participants, using `box` and `end box` commands.

You can add an optional title or a optional background color, after the `box` keyword.

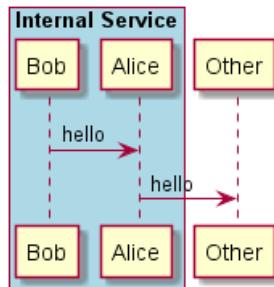
```
@startuml
```

```

box "Internal Service" #LightBlue
participant Bob
participant Alice
end box
participant Other

Bob -> Alice : hello
Alice -> Other : hello
  
```

```
@enduml
```



### 1.34 Removing Foot Boxes

You can use the `hide footbox` keywords to remove the foot boxes of the diagram.

```
@startuml
```

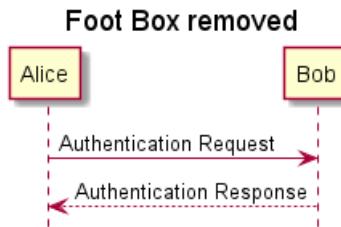
```

hide footbox
title Foot Box removed

Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response
  
```

```
@enduml
```





### 1.35 Skinparam

You can use the skinparam command to change colors and fonts for the drawing.

You can use this command:

- In the diagram definition, like any other commands,
- In an included file,
- In a configuration file, provided in the command line or the ANT task.

You can also change other rendering parameter, as seen in the following examples:

```

@startuml
skinparam sequenceArrowThickness 2
skinparam roundcorner 20
skinparam maxmessagesize 60
skinparam sequenceParticipant underline

actor User
participant "First Class" as A
participant "Second Class" as B
participant "Last Class" as C

User -> A: DoWork
activate A

A -> B: Create Request
activate B

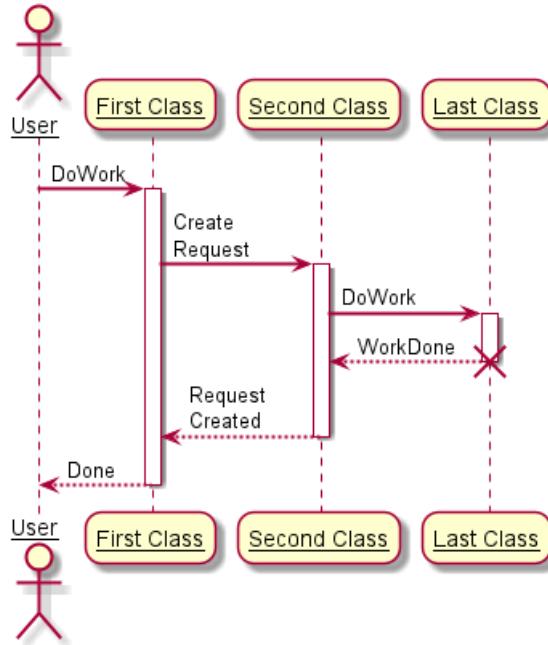
B -> C: DoWork
activate C
C --> B: WorkDone
destroy C

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml
  
```





```

@startuml
skinparam backgroundColor #EEEBDC
skinparam handwritten true

skinparam sequence {
    ArrowColor DeepSkyBlue
    ActorBorderColor DeepSkyBlue
    LifeLineBorderColor blue
    LifeLineBackgroundColor #A9DCDF

    ParticipantBorderColor DeepSkyBlue
    ParticipantBackgroundColor DodgerBlue
    ParticipantFontName Impact
    ParticipantFontSize 17
    ParticipantFontColor #A9DCDF

    ActorBackgroundColor aqua
    ActorFontColor DeepSkyBlue
    ActorFontSize 17
    ActorFontName Aapex
}

actor User
participant "First Class" as A
participant "Second Class" as B
participant "Last Class" as C

User -> A: DoWork
activate A

A -> B: Create Request
activate B

B -> C: DoWork
activate C
C --> B: WorkDone
destroy C

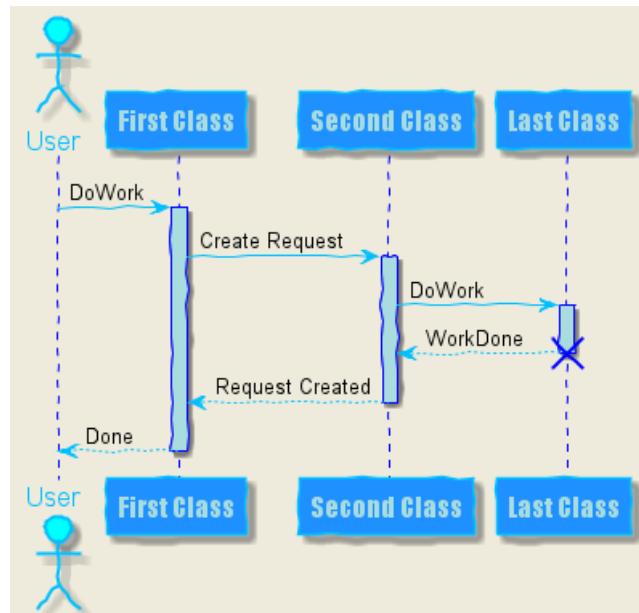
```



```
B --> A: Request Created
deactivate B
```

```
A --> User: Done
deactivate A
```

```
@enduml
```

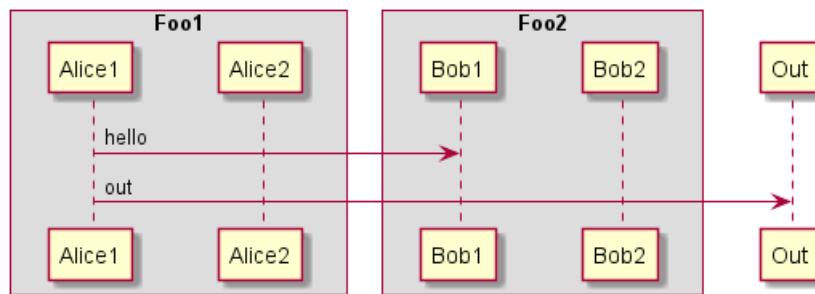


## 1.36 Changing padding

It is possible to tune some padding settings.

```
@startuml
skinparam ParticipantPadding 20
skinparam BoxPadding 10

box "Foo1"
participant Alice1
participant Alice2
end box
box "Foo2"
participant Bob1
participant Bob2
end box
Alice1 -> Bob1 : hello
Alice1 -> Out : out
@enduml
```

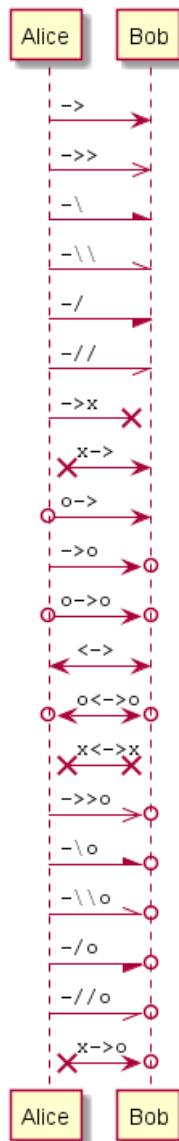


## 1.37 Appendix: Examples of all arrow type

### 1.37.1 Normal arrow

```
@startuml
participant Alice as a
participant Bob   as b
a ->    b : ""->    ""
a ->>   b : ""->>   ""
a -\    b : ""-\    ""
a -\\\" b : ""-\\\\\""
a -/    b : ""-/    ""
a -//   b : ""-//   ""
a ->x  b : ""->x  ""
a x->  b : ""x->  ""
a o->  b : ""o->  ""
a ->o  b : ""->o  ""
a o->o b : ""o->o ""
a <->  b : ""<->  ""
a o<->o b : ""o<->o""
a x<->x b : ""x<->x"""
a ->>o b : ""->>o """
a -\o   b : ""-\o   ""
a -\\\"o b : ""-\\\\\"o"""
a -/o   b : ""-/o   ""
a -//o  b : ""-//o  ""
a x->o b : ""x->o """
@enduml
```





### 1.37.2 Incoming and outgoing messages (with '[', ']')

### 1.37.3 Incoming messages (with '[')

```

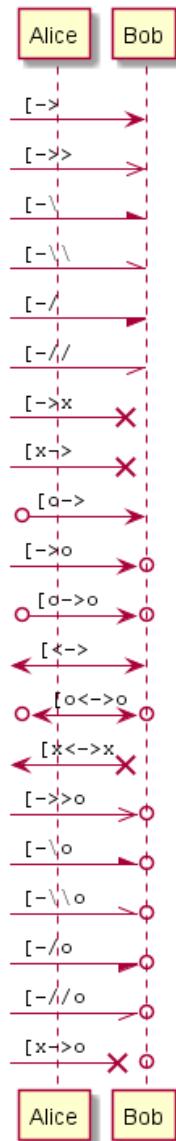
@startuml
participant Alice as a
participant Bob   as b
[->      b : ""[->    """
[->>    b : ""[->>   """
[-\      b : ""[-\    """
[-\\\"  b : ""[-\\\" """
[-/      b : ""[-/    """
[-//     b : ""[-//   """
[->x   b : ""[->x """
[x->   b : ""[x-> """
[o->   b : ""[o-> """
[->o   b : ""[->o """
[o->o  b : ""[o->o """
[<->   b : ""[<-> """
[o<->o b : ""[o<->o"""
[x<->x b : ""[x<->x"""
[->o   b : ""[->o """

```

```

[-\o      b : """[-\o   """
[-\\o     b : """[-\\\\o"""
[-/o      b : """[-/o   """
[-//o     b : """[-//o   """
[x->o    b : """[x->o   """
@enduml

```



#### 1.37.4 Outgoing messages (with ']')

```

@startuml
participant Alice as a
participant Bob   as b
a ->]      : """->]   """
a ->>]     : """->>]   """
a -\]       : """-\]   """
a -\\]      : """-\\\\"]   """
a -/]       : """-/]   """
a -//]      : """-//]   """
a ->x]     : """->x]   """
a x->]     : """x->]   """
a o->]     : """o->]   """
a ->o]     : """->o]   """

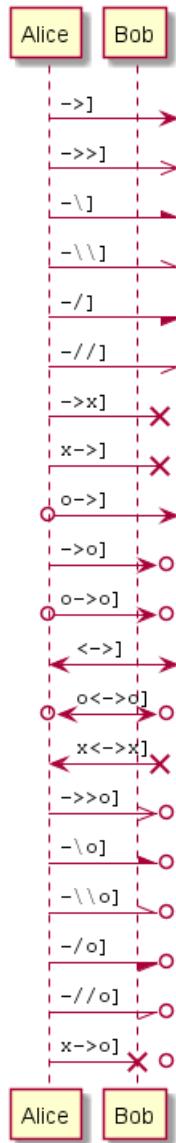
```



```

a o->o] : ""o->o] """
a <->] : ""<->] """
a o<->o] : ""o<->o] """
a x<->x] : ""x<->x] """
a ->>o] : ""->>o] """
a -\o] : ""-\o] """
a -\\o] : ""-\\o] """
a -/o] : ""-/o] """
a -//o] : ""-/o] """
a x->o] : ""x->o] """
@enduml

```



### 1.37.5 Short incoming and outgoing messages (with '?')

### 1.37.6 Short incoming (with '?')

```

@startuml
participant Alice as a
participant Bob as b
a -> b : //Long long label// 
?-> b : ""?-> """
?->> b : ""?->> """

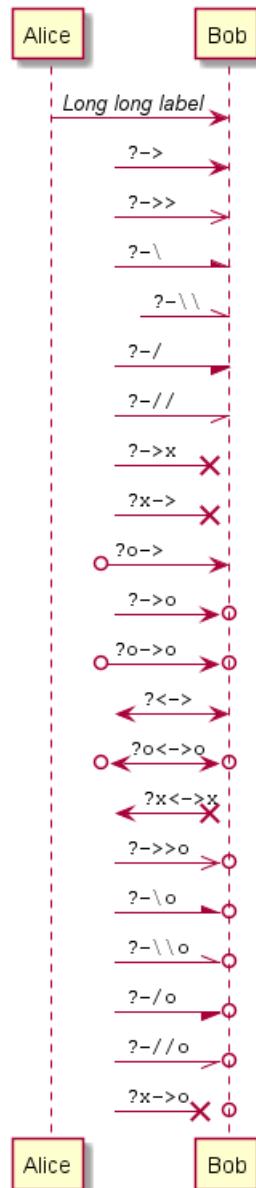
```



```

?-\  
b : """?-\  
"""  
?-\\\  
b : """?-\\\""""  
?-/  
b : """?-/-"""  
?-//  
b : """?-//"""  
?->x  
b : """?->x"""  
?x->  
b : """?x->"""  
?o->  
b : """?o->"""  
?->o  
b : """?->o"""  
?o->o  
b : """?o->o"""  
?<->  
b : """?<->"""  
?o<->o  
b : """?o<->o"""  
?x<->x  
b : """?x<->x"""  
?->>o  
b : """?->>o"""  
?-\\o  
b : """?-\\o"""  
?-\\\\o  
b : """?-\\\\o"""  
?-/o  
b : """?-/o"""  
?-//o  
b : """?-//o"""  
?x->o  
b : """?x->o"""  
@enduml

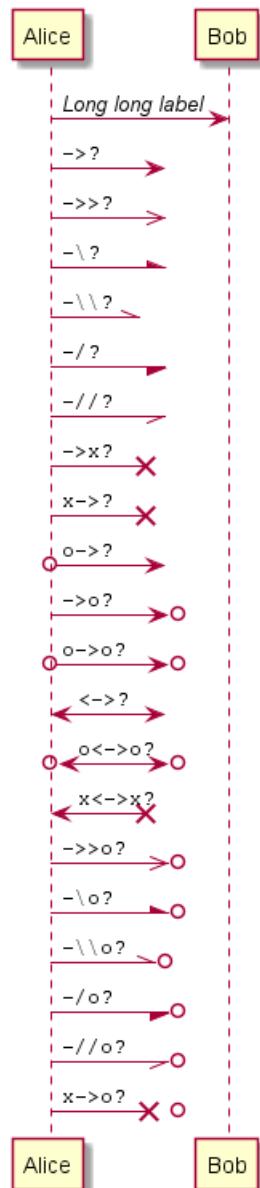
```



### 1.37.7 Short outgoing (with '?')

```
@startuml
participant Alice as a
participant Bob   as b
a ->    b : //Long long label// 
a ->?   : ""->?   ""
a ->>?  : ""->>?  ""
a -\?   : ""-\?   ""
a -\\?\? : ""-\\\\\?\?""
a -/?   : ""-/?   ""
a -//?  : ""-//?  ""
a ->x? : ""->x?  ""
a x->? : ""x->?  ""
a o->? : ""o->?  ""
a ->o? : ""->o?  ""
a o->o? : ""o->o?  ""
a <->? : ""<->?  ""
a o<->o? : ""o<->o?"""
a x<->x? : ""x<->x?"""
a ->>o? : ""->>o?  ""
a -\o?   : ""-\o?   ""
a -\\o?  : ""-\\\\o?"""
a -/o?   : ""-/o?   ""
a -//o?  : ""-//o?  ""
a x->o? : ""x->o?  ""
@enduml
```





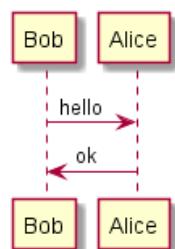
## 1.38 Specific SkinParameter

### 1.38.1 By default

```

@startuml
Bob -> Alice : hello
Alice -> Bob : ok
@enduml

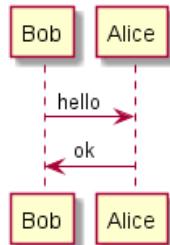
```



### 1.38.2 LifelineStrategy

- nosolid (*by default*)

```
@startuml
skinparam lifelineStrategy nosolid
Bob -> Alice : hello
Alice -> Bob : ok
@enduml
```

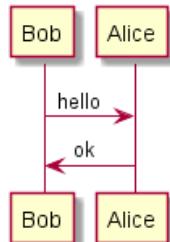


[Ref. QA-9016]

- solid

In order to have solid life line in sequence diagrams, you can use: `skinparam lifelineStrategy solid`

```
@startuml
skinparam lifelineStrategy solid
Bob -> Alice : hello
Alice -> Bob : ok
@enduml
```



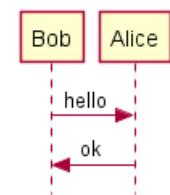
[Ref. QA-2794]

### 1.38.3 style strictuml

To be conform to strict UML (*for arrow style: emits triangle rather than sharp arrowheads*), you can use:

- `skinparam style strictuml`

```
@startuml
skinparam style strictuml
Bob -> Alice : hello
Alice -> Bob : ok
@enduml
```



[Ref. QA-1047]

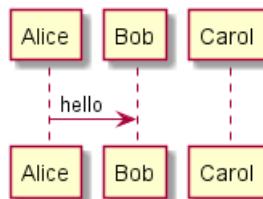


### 1.39 Hide unlinked participant

By default, all participants are displayed.

```
@startuml  
participant Alice  
participant Bob  
participant Carol
```

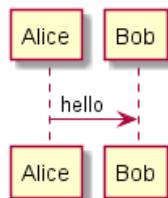
```
Alice -> Bob : hello  
@enduml
```



But you can `hide unlinked` participant.

```
@startuml  
hide unlinked  
participant Alice  
participant Bob  
participant Carol
```

```
Alice -> Bob : hello  
@enduml
```



[Ref. QA-4247]



## 2 Use Case Diagram

Let's have a few examples:

### 2.1 Usecases

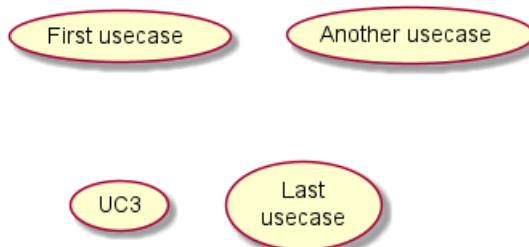
Use cases are enclosed using between parentheses (because two parentheses looks like an oval).

You can also use the `usecase` keyword to define a usecase. And you can define an alias, using the `as` keyword. This alias will be used later, when defining relations.

```
@startuml
```

```
(First usecase)
(Another usecase) as (UC2)
usecase UC3
usecase (Last\usecase) as UC4
```

```
@enduml
```



### 2.2 Actors

The name defining an actor is enclosed between colons.

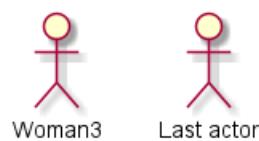
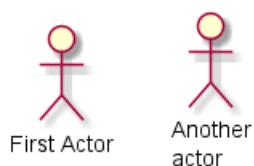
You can also use the `actor` keyword to define an actor. An alias can be assigned using the `as` keyword and can be used later instead of the actor's name, e. g. when defining relations.

You can see from the following examples, that the actor definitions are optional.

```
@startuml
```

```
:First Actor:
:Another\actor: as Man2
actor Woman3
actor :Last actor: as Person1
```

```
@enduml
```



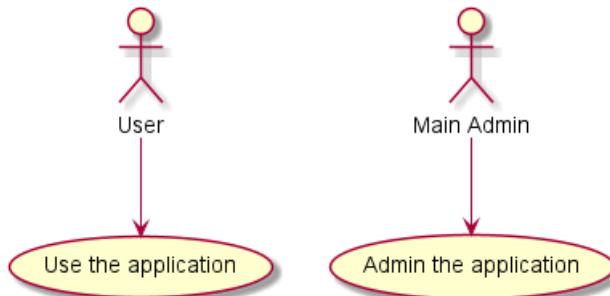
## 2.3 Change Actor style

You can change the actor style from stick man (*by default*) to:

- an awesome man with the `skinparam actorStyle awesome` command;
- a hollow man with the `skinparam actorStyle hollow` command.

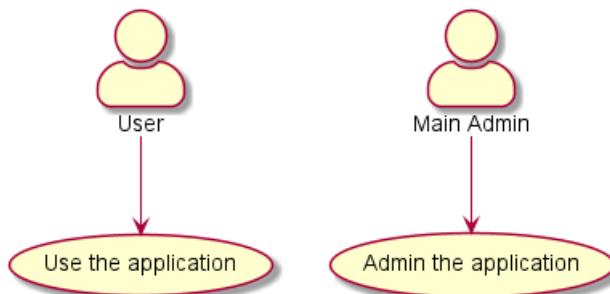
### 2.3.1 Stick man (*by default*)

```
@startuml
:User: --> (Use)
"Main Admin" as Admin
"Use the application" as (Use)
Admin --> (Admin the application)
@enduml
```



### 2.3.2 Awesome man

```
@startuml
skinparam actorStyle awesome
:User: --> (Use)
"Main Admin" as Admin
"Use the application" as (Use)
Admin --> (Admin the application)
@enduml
```

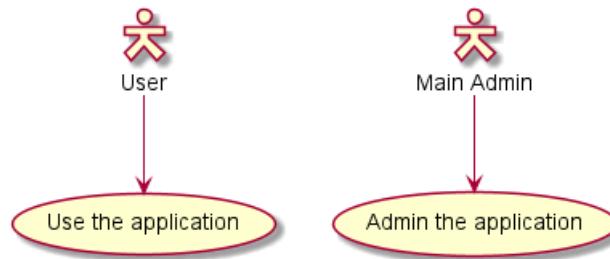


[Ref. QA-10493]

### 2.3.3 Hollow man

```
@startuml
skinparam actorStyle Hollow
:User: --> (Use)
"Main Admin" as Admin
"Use the application" as (Use)
Admin --> (Admin the application)
@enduml
```





[Ref. PR#396]

## 2.4 Usecases description

If you want to have a description spanning several lines, you can use quotes.

You can also use the following separators:

- -- (dashes)
- .. (periods)
- == (equals)
- \_\_ (underscores)

By using them pairwise and enclosing text between them, you can create separators with titles.

@startuml

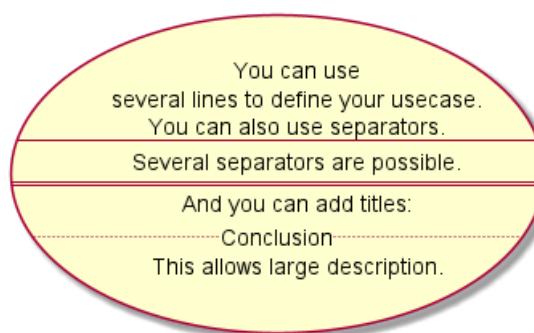
```

usecase UC1 as "You can use
several lines to define your usecase.
You can also use separators.

-- 
Several separators are possible.

== 
And you can add titles:
..Conclusion..
This allows large description."
  
```

@enduml



## 2.5 Use package

You can use packages to group actors or use cases.

```

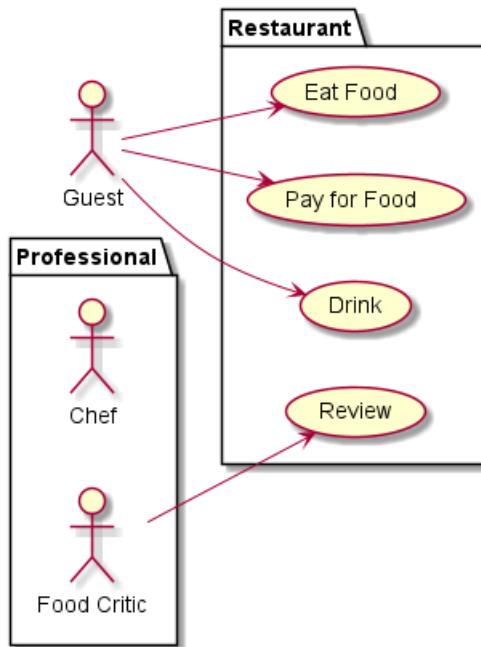
@startuml
left to right direction
actor Guest as g
package Professional {
    actor Chef as c
    actor "Food Critic" as fc
  
```



```

}
package Restaurant {
    usecase "Eat Food" as UC1
    usecase "Pay for Food" as UC2
    usecase "Drink" as UC3
    usecase "Review" as UC4
}
fc --> UC4
g --> UC1
g --> UC2
g --> UC3
@enduml

```



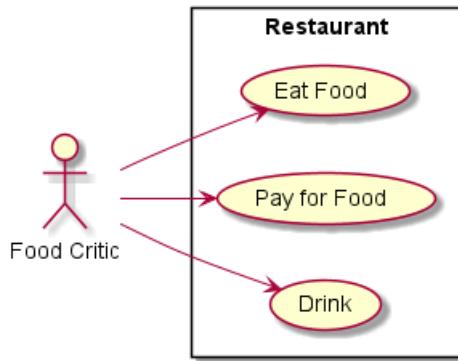
You can use rectangle to change the display of the package.

```

@startuml
left to right direction
actor "Food Critic" as fc
rectangle Restaurant {
    usecase "Eat Food" as UC1
    usecase "Pay for Food" as UC2
    usecase "Drink" as UC3
}
fc --> UC1
fc --> UC2
fc --> UC3
@enduml

```





## 2.6 Basic example

To link actors and use cases, the arrow `-->` is used.

The more dashes – in the arrow, the longer the arrow. You can add a label on the arrow, by adding a `:` character in the arrow definition.

In this example, you see that `User` has not been defined before, and is used as an actor.

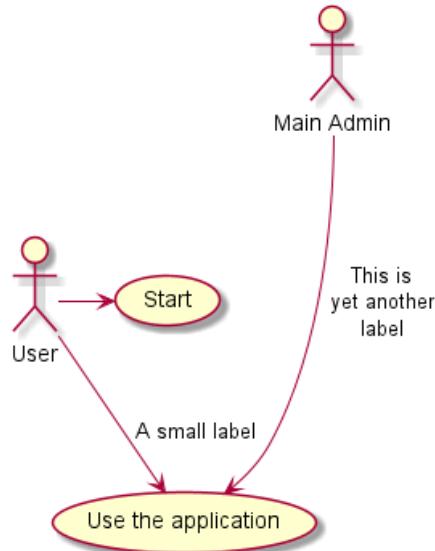
`@startuml`

```

User --> (Start)
User --> (Use the application) : A small label

:Main Admin: ---> (Use the application) : This is\nyet another\nlabel
  
```

`@enduml`



## 2.7 Extension

If one actor/use case extends another one, you can use the symbol `<|--`.

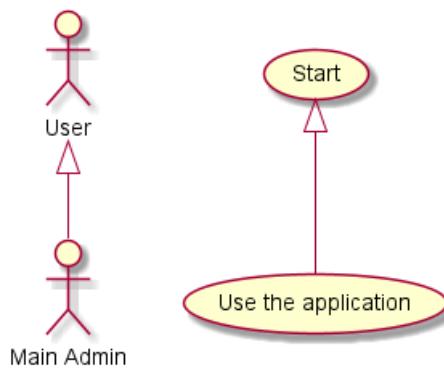
```

@startuml
:Main Admin: as Admin
(Use the application) as (Use)

User <|-- Admin
(Start) <|-- (Use)
  
```



@enduml



## 2.8 Using notes

You can use the `note left of`, `note right of`, `note top of`, `note bottom of` keywords to define notes related to a single object.

A note can be also define alone with the `note` keyword, then linked to other objects using the `..` symbol.

```
@startuml
:Main Admin: as Admin
(Use the application) as (Use)
```

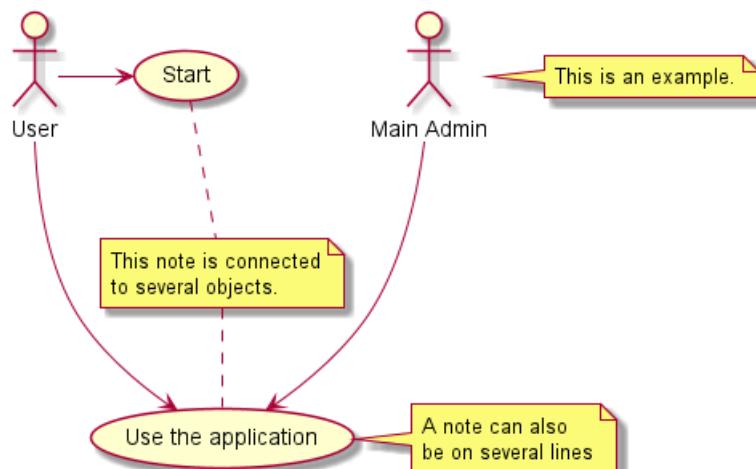
```
User -> (Start)
User --> (Use)
```

```
Admin ---> (Use)
```

```
note right of Admin : This is an example.
```

```
note right of (Use)
A note can also
be on several lines
end note
```

```
note "This note is connected\nto several objects." as N2
(Start) .. N2
N2 .. (Use)
@enduml
```



## 2.9 Stereotypes

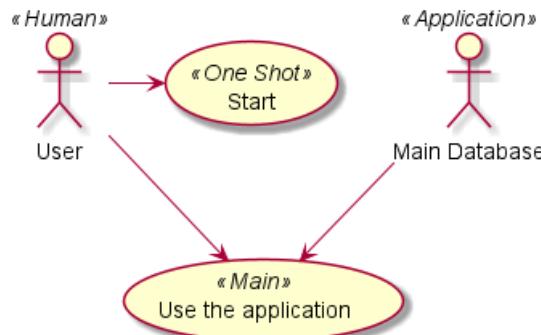
You can add stereotypes while defining actors and use cases using << and >>.

```
@startuml
User << Human >>
:Main Database: as MySql << Application >>
(Start) << One Shot >>
(Use the application) as (Use) << Main >>
```

```
User -> (Start)
User --> (Use)
```

```
MySql --> (Use)
```

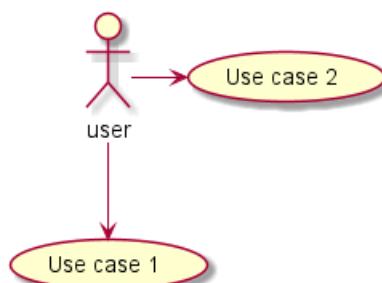
```
@enduml
```



## 2.10 Changing arrows direction

By default, links between classes have two dashes -- and are vertically oriented. It is possible to use horizontal link by putting a single dash (or dot) like this:

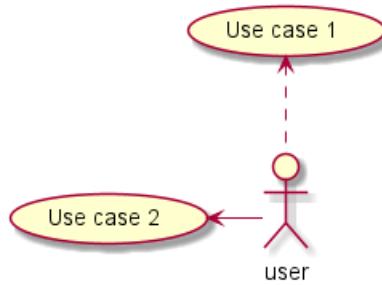
```
@startuml
:user: --> (Use case 1)
:user: -> (Use case 2)
@enduml
```



You can also change directions by reversing the link:

```
@startuml
(Use case 1) <.. :user:
(Use case 2) <- :user:
@enduml
```



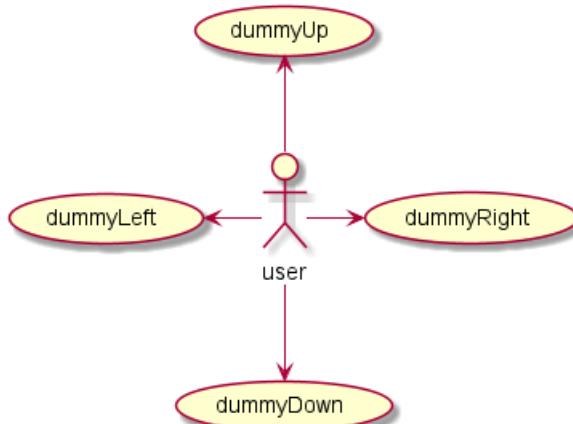


It is also possible to change arrow direction by adding `left`, `right`, `up` or `down` keywords inside the arrow:

```

@startuml
:user: -left-> (dummyLeft)
:user: -right-> (dummyRight)
:user: -up-> (dummyUp)
:user: -down-> (dummyDown)
@enduml

```



You can shorten the arrow by using only the first character of the direction (for example, `-d-` instead of `-down-`) or the two first characters (`-do-`).

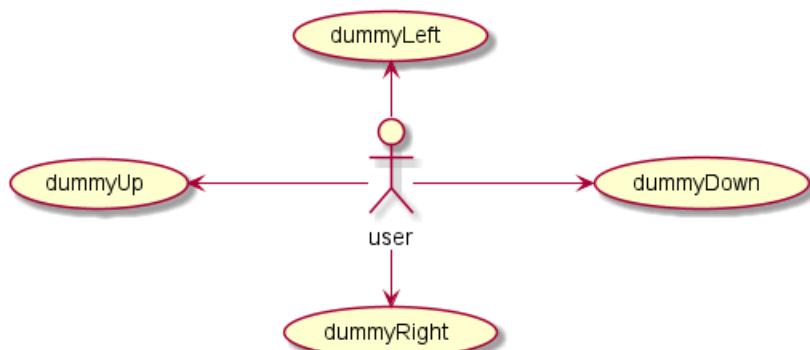
Please note that you should not abuse this functionality : *Graphviz* gives usually good results without tweaking.

And with the `left to right direction` parameter:

```

@startuml
left to right direction
:user: -left-> (dummyLeft)
:user: -right-> (dummyRight)
:user: -up-> (dummyUp)
:user: -down-> (dummyDown)
@enduml

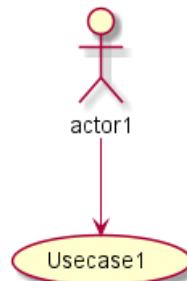
```



## 2.11 Splitting diagrams

The `newpage` keyword to split your diagram into several pages or images.

```
@startuml
:actor1: --> (Usecase1)
newpage
:actor2: --> (Usecase2)
@enduml
```

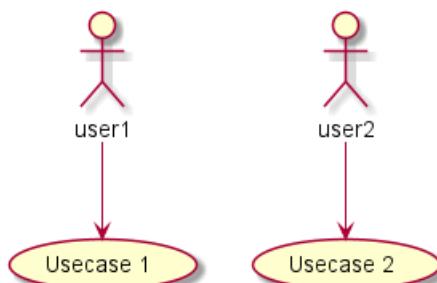


## 2.12 Left to right direction

The general default behavior when building diagram is **top to bottom**.

```
@startuml
'default
top to bottom direction
user1 --> (Usecase 1)
user2 --> (Usecase 2)

@enduml
```

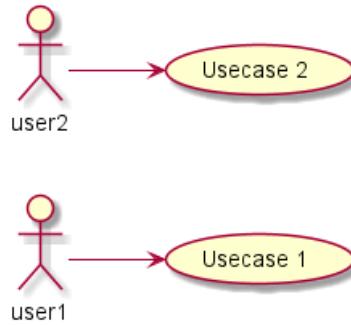


You may change to **left to right** using the `left to right direction` command. The result is often better with this direction.

```
@startuml
left to right direction
user1 --> (Usecase 1)
user2 --> (Usecase 2)

@enduml
```





## 2.13 Skinparam

You can use the skinparam command to change colors and fonts for the drawing.

You can use this command :

- In the diagram definition, like any other commands,
- In an included file,
- In a configuration file, provided in the command line or the ANT task.

You can define specific color and fonts for stereotyped actors and usecases.

```

@startuml
skinparam handwritten true

skinparam usecase {
BackgroundColor DarkSeaGreen
BorderColor DarkSlateGray

BackgroundColor<< Main >> YellowGreen
BorderColor<< Main >> YellowGreen

ArrowColor Olive
ActorBorderColor black
ActorFontName Courier

ActorBackgroundColor<< Human >> Gold
}

User << Human >>
:Main Database: as MySql << Application >>
(Start) << One Shot >>
(Use the application) as (Use) << Main >>

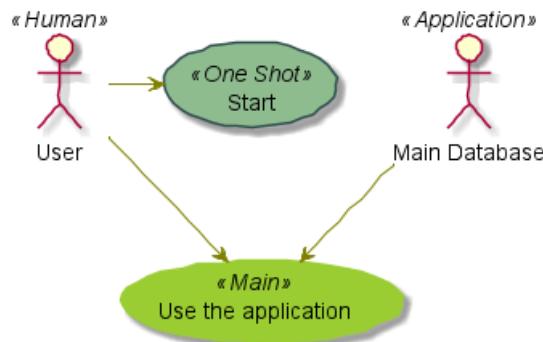
User -> (Start)
User --> (Use)

MySql --> (Use)

@enduml

```



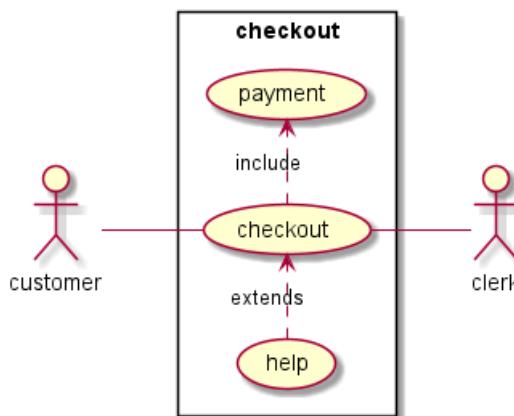


## 2.14 Complete example

```

@startuml
left to right direction
skinparam packageStyle rectangle
actor customer
actor clerk
rectangle checkout {
    customer -- (checkout)
    (checkout) .> (payment) : include
    (help) .> (checkout) : extends
    (checkout) -- clerk
}
@enduml

```



## 2.15 Business Use Case

You can add / to make Business Use Case.

### 2.15.1 Business Usecase

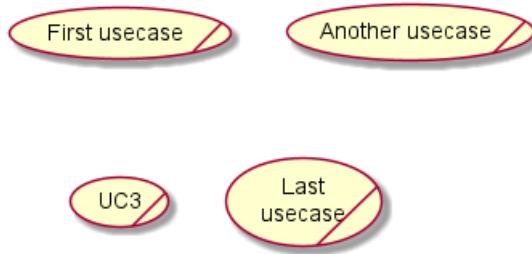
```

@startuml
(First usecase)/
(Another usecase)/ as (UC2)
usecase/ UC3
usecase/ (Last\nusecase) as UC4

```

@enduml





### 2.15.2 Business Actor

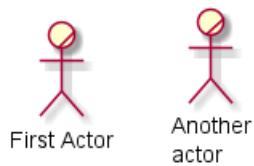
@startuml

```

:First Actor:/
:Another\actor:/ as Man2
actor/ Woman3
actor/ :Last actor: as Person1

```

@enduml



[Ref. QA-12179]

### 2.16 Change arrow color and style (inline style)

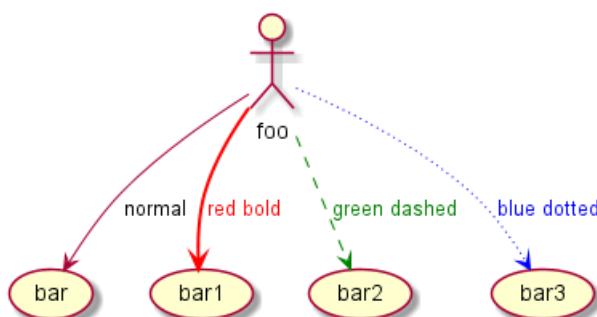
You can change the color or style of individual arrows using the inline following notation:

- #color;line.[bold|dashed|dotted];text:color

```

@startuml
actor foo
foo --> (bar) : normal
foo --> (bar1) #line:red;line.bold;text:red : red bold
foo --> (bar2) #green;line.dashed;text:green : green dashed
foo --> (bar3) #blue;line.dotted;text:blue : blue dotted
@enduml

```



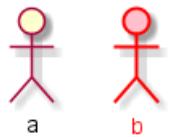
[Ref. QA-3770 and QA-3816] [See similar feature on deployment-diagram or class diagram]

## 2.17 Change element color and style (inline style)

You can change the color or style of individual element using the following notation:

- # [color|back:color];line:color;line.[bold|dashed|dotted];text:color

```
@startuml
actor a
actor b #pink;line:red;line.bold;text:red
usecase c #palegreen;line:green;line.dashed;text:green
usecase d #aliceblue;line:blue;line.dotted;text:blue
@enduml
```



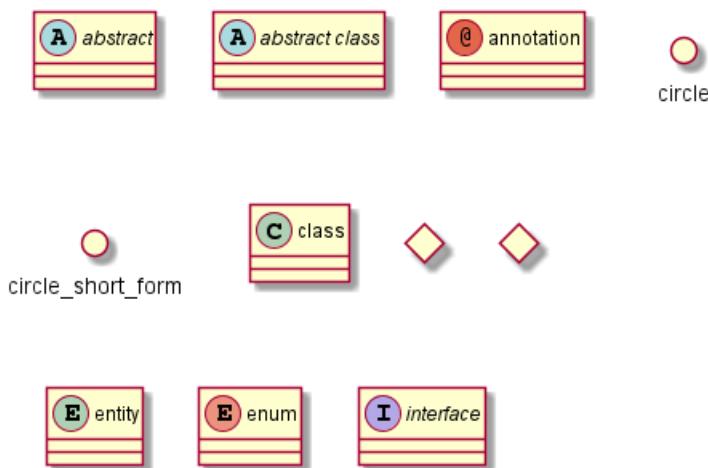
[Ref. QA-5340 and adapted from QA-6852]



## 3 Class Diagram

### 3.1 Declaring element

```
@startuml
abstract      abstract
abstract class "abstract class"
annotation    annotation
circle        circle
()            circle_short_form
class         class
diamond       diamond
<>           diamond_short_form
entity        entity
enum          enum
interface     interface
@enduml
```



### 3.2 Relations between classes

Relations between classes are defined using the following symbols :

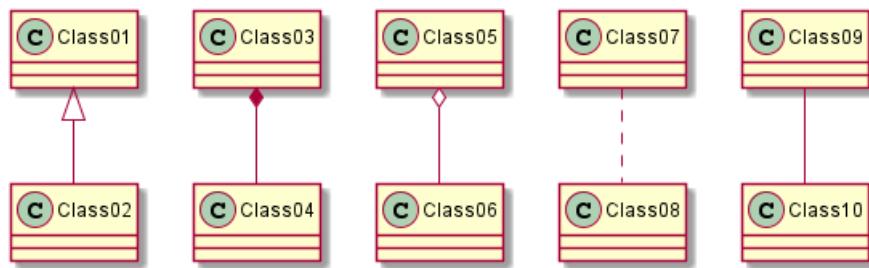
Type	Symbol	Drawing
Extension	< --	
Composition	*---	
Aggregation	o--	

It is possible to replace -- by .. to have a dotted line.

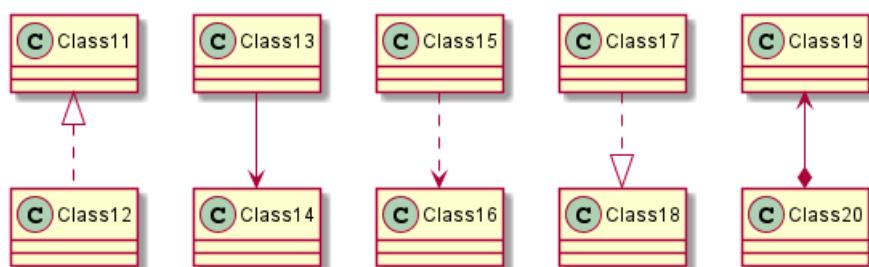
Knowing those rules, it is possible to draw the following drawings:

```
@startuml
Class01 <|-- Class02
Class03 *--- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10
@enduml
```

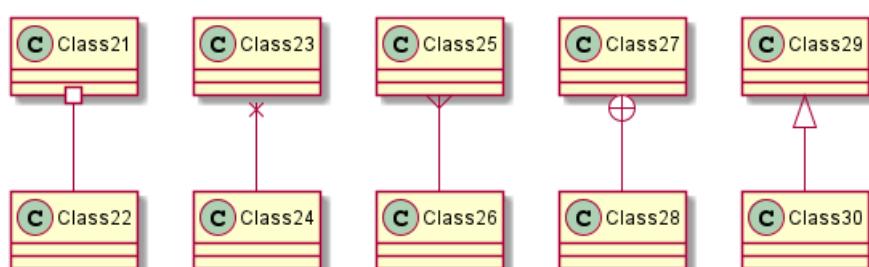




```
@startuml
Class11 <|.. Class12
Class13 --> Class14
Class15 ..> Class16
Class17 ..|> Class18
Class19 <--* Class20
@enduml
```



```
@startuml
Class21 #-- Class22
Class23 x-- Class24
Class25 }-- Class26
Class27 +--- Class28
Class29 ^-- Class30
@enduml
```



### 3.3 Label on relations

It is possible to add a label on the relation, using `:`, followed by the text of the label.

For cardinality, you can use double-quotes `" "` on each side of the relation.

```
@startuml
```

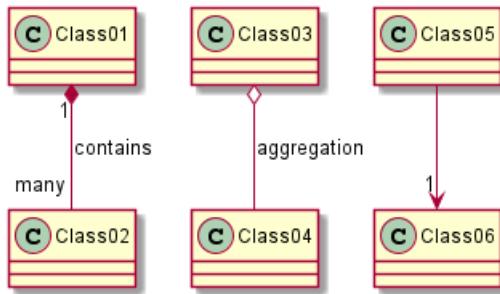
```
Class01 "1" *-- "many" Class02 : contains
```

```
Class03 o-- Class04 : aggregation
```

```
Class05 --> "1" Class06
```

```
@enduml
```





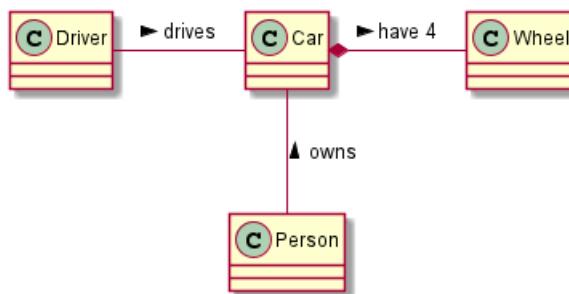
You can add an extra arrow pointing at one object showing which object acts on the other object, using < or > at the begin or at the end of the label.

```

@startuml
class Car

Driver -> Car : drives >
Car *-- Wheel : have 4 >
Car --> Person : < owns
  
```

@enduml



### 3.4 Adding methods

To declare fields and methods, you can use the symbol : followed by the field's or method's name.

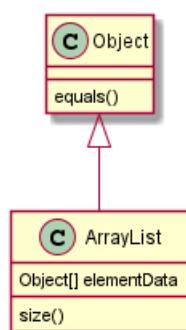
The system checks for parenthesis to choose between methods and fields.

```

@startuml
Object <|-- ArrayList

Object : equals()
ArrayList : Object[] elementData
ArrayList : size()

@enduml
  
```



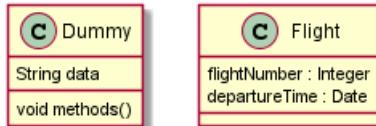
It is also possible to group between brackets {} all fields and methods.



Note that the syntax is highly flexible about type/name order.

```
@startuml
class Dummy {
    String data
    void methods()
}

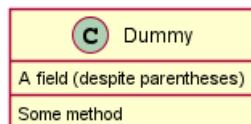
class Flight {
    flightNumber : Integer
    departureTime : Date
}
@enduml
```



You can use `{field}` and `{method}` modifiers to override default behaviour of the parser about fields and methods.

```
@startuml
class Dummy {
    {field} A field (despite parentheses)
    {method} Some method
}

@enduml
```



## 3.5 Defining visibility

When you define methods or fields, you can use characters to define the visibility of the corresponding item:

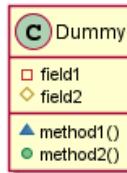
Character	Icon for field	Icon for method	Visibility
-	□	■	private
#	◊	◊	protected
~	△	△	package private
+	○	●	public

```
@startuml

class Dummy {
    -field1
    #field2
    ~method1()
    +method2()
}

@enduml
```





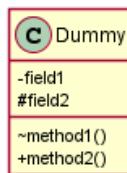
You can turn off this feature using the `skinparam classAttributeIconSize 0` command :

```

@startuml
skinparam classAttributeIconSize 0
class Dummy {
    -field1
    #field2
    ~method1()
    +method2()
}

@enduml

```



## 3.6 Abstract and Static

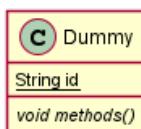
You can define static or abstract methods or fields using the `{static}` or `{abstract}` modifier.

These modifiers can be used at the start or at the end of the line. You can also use `{classifier}` instead of `{static}`.

```

@startuml
class Dummy {
    {static} String id
    {abstract} void methods()
}
@enduml

```



## 3.7 Advanced class body

By default, methods and fields are automatically regrouped by PlantUML. You can use separators to define your own way of ordering fields and methods. The following separators are possible : `-- .. == _-`.

You can also use titles within the separators:

```

@startuml
class Foo1 {
    You can use
    several lines
    ..
    as you want
    and group
    ==

```



```

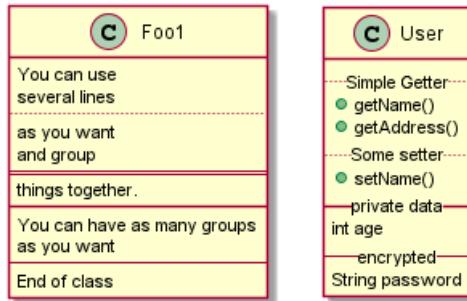
things together.

-- You can have as many groups
as you want
--
End of class
}

class User {
    ... Simple Getter ...
    + getName()
    + getAddress()
    ... Some setter ...
    + setName()
    -- private data --
    int age
    -- encrypted --
    String password
}

```

@enduml



## 3.8 Notes and stereotypes

Stereotypes are defined with the `class` keyword, `<>` and `>>`.

You can also define notes using `note left of`, `note right of`, `note top of`, `note bottom of` keywords.

You can also define a note on the last defined class using `note left`, `note right`, `note top`, `note bottom`.

A note can be also define alone with the `note` keywords, then linked to other objects using the `..` symbol.

```

@startuml
class Object << general >>
Object <|-- ArrayList

note top of Object : In java, every class\nextends this one.

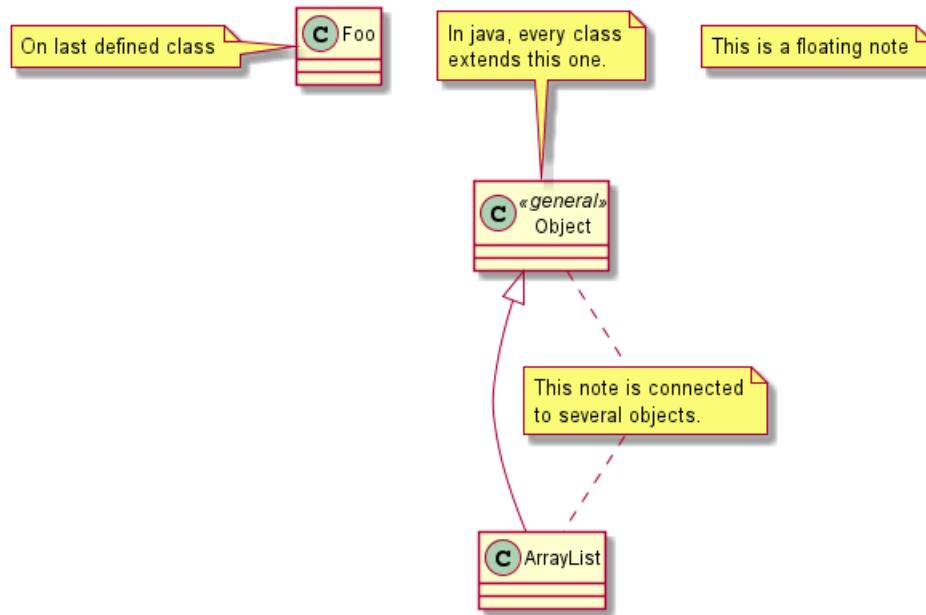
note "This is a floating note" as N1
note "This note is connected\nto several objects." as N2
Object .. N2
N2 .. ArrayList

class Foo
note left: On last defined class

@enduml

```





### 3.9 More on notes

It is also possible to use few HTML tags (See Creole expression) like :

- <b>
- <u>
- <i>
- <s>, <del>, <strike>
- <font color="#AAAAAA"> or <font color="colorName">
- <color:#AAAAAA> or <color:colorName>
- <size:nn> to change font size
-  or <img:file>: the file must be accessible by the filesystem

You can also have a note on several lines.

You can also define a note on the last defined class using `note left`, `note right`, `note top`, `note bottom`.

`@startuml`

```

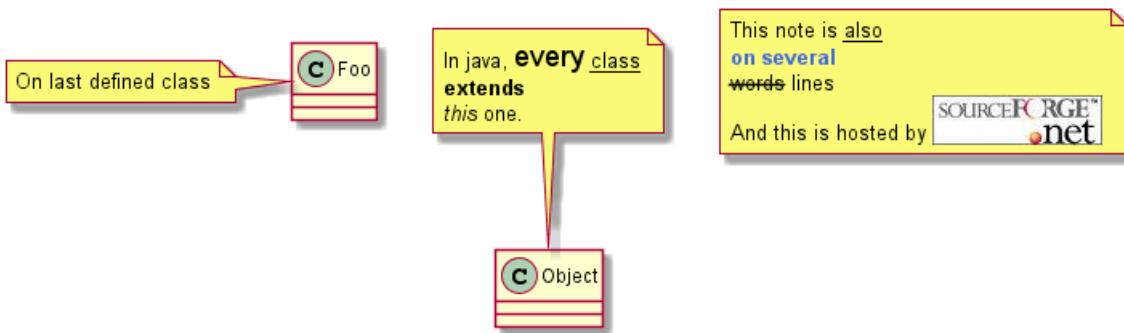
class Foo
note left: On last defined class

note top of Object
  In java, <size:18>every</size> <u>class</u>
  <b>extends</b>
  <i>this</i> one.
end note

note as N1
  This note is <u>also</u>
  <b><color:royalBlue>on several</color>
  <s>words</s> lines
  And this is hosted by <img:sourceforge.jpg>
end note
  
```



@enduml

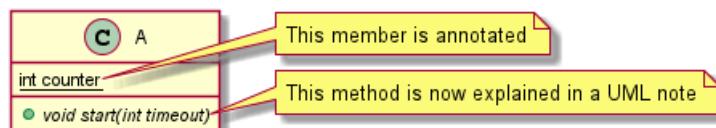


### 3.10 Note on field (field, attribute, member) or method

It is possible to add a note on field (field, attribut, member) or on method.

#### 3.10.1 Note on field or method

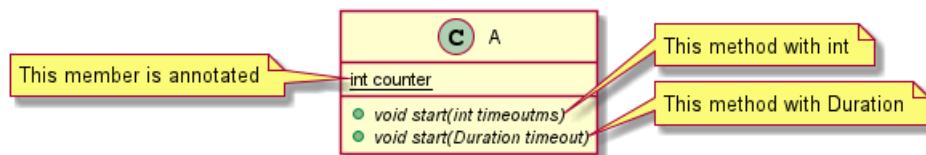
```
@startuml
class A {
{static} int counter
+void {abstract} start(int timeout)
}
note right of A::counter
    This member is annotated
end note
note right of A::start
    This method is now explained in a UML note
end note
@enduml
```



#### 3.10.2 Note on method with the same name

```
@startuml
class A {
{static} int counter
+void {abstract} start(int timeoutms)
+void {abstract} start(Duration timeout)
}
note left of A::counter
    This member is annotated
end note
note right of A::"start(int timeoutms)"
    This method with int
end note
note right of A::"start(Duration timeout)"
    This method with Duration
end note
@enduml
```





[Ref. QA-3474 and QA-5835]

### 3.11 Note on links

It is possible to add a note on a link, just after the link definition, using `note on link`.

You can also use `note left on link`, `note right on link`, `note top on link`, `note bottom on link` if you want to change the relative position of the note with the label.

`@startuml`

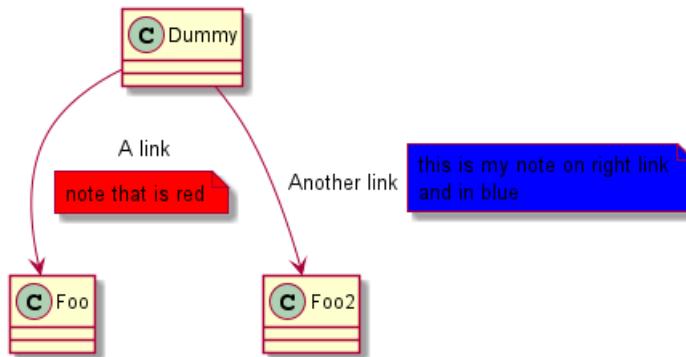
```

class Dummy
Dummy --> Foo : A link
note on link #red: note that is red

Dummy --> Foo2 : Another link
note right on link #blue
this is my note on right link
and in blue
end note

```

`@enduml`



### 3.12 Abstract class and interface

You can declare a class as abstract using `abstract` or `abstract class` keywords.

The class will be printed in *italic*.

You can use the `interface`, `annotation` and `enum` keywords too.

`@startuml`

```

abstract class AbstractList
abstract AbstractCollection
interface List
interface Collection

List <|-- AbstractList
Collection <|-- AbstractCollection

Collection <|- List
AbstractCollection <|- AbstractList

```



```
AbstractList <|-- ArrayList
```

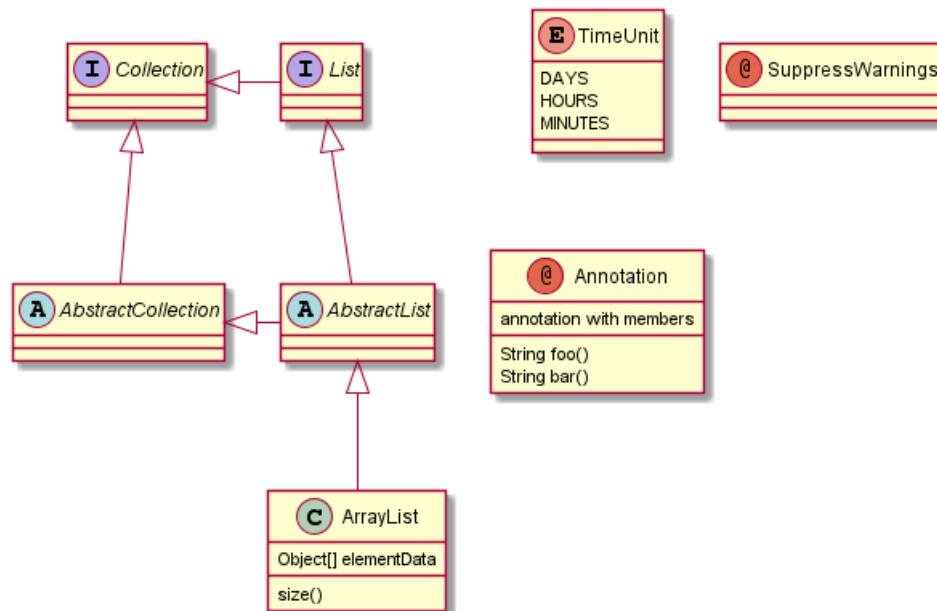
```
class ArrayList {
    Object[] elementData
    size()
}

enum TimeUnit {
    DAYS
    HOURS
    MINUTES
}

annotation SuppressWarnings
```

annotation Annotation {  
 annotation with members  
 String foo()  
 String bar()  
}

```
@enduml
```



[Ref. 'Annotation with members' Issue#458]

### 3.13 Using non-letters

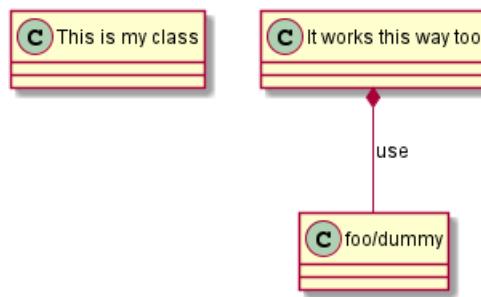
If you want to use non-letters in the class (or enum...) display, you can either :

- Use the `as` keyword in the class definition
- Put quotes "" around the class name

```
@startuml
class "This is my class" as class1
class class2 as "It works this way too"

class2 *-- "foo/dummy" : use
@enduml
```





### 3.14 Hide attributes, methods...

You can parameterize the display of classes using the `hide/show` command.

The basic command is: `hide empty members`. This command will hide attributes or methods if they are empty.

Instead of `empty members`, you can use:

- `empty fields` or `empty attributes` for empty fields,
- `empty methods` for empty methods,
- `fields` or `attributes` which will hide fields, even if they are described,
- `methods` which will hide methods, even if they are described,
- `members` which will hide fields and methods, even if they are described,
- `circle` for the circled character in front of class name,
- `stereotype` for the stereotype.

You can also provide, just after the `hide` or `show` keyword:

- `class` for all classes,
- `interface` for all interfaces,
- `enum` for all enums,
- `<<foo1>>` for classes which are stereotyped with `foo1`,
- an existing class name.

You can use several `show/hide` commands to define rules and exceptions.

`@startuml`

```

class Dummy1 {
    +myMethods()
}

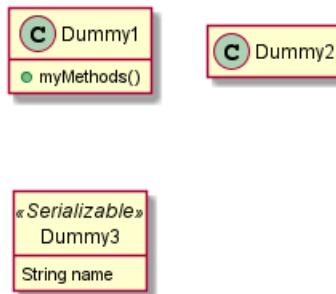
class Dummy2 {
    +hiddenMethod()
}

class Dummy3 <<Serializable>> {
String name
}

hide members
hide <<Serializable>> circle
show Dummy1 methods
show <<Serializable>> fields
  
```



```
@enduml
```



### 3.15 Hide classes

You can also use the `show/hide` commands to hide classes.

This may be useful if you define a large !included file, and if you want to hide some classes after file inclusion.

```
@startuml
```

```

class Foo1
class Foo2

Foo2 *-- Foo1

hide Foo2

```

```
@enduml
```



### 3.16 Remove classes

You can also use the `remove` commands to remove classes.

This may be useful if you define a large !included file, and if you want to remove some classes after file inclusion.

```
@startuml
```

```

class Foo1
class Foo2

Foo2 *-- Foo1

remove Foo2

```

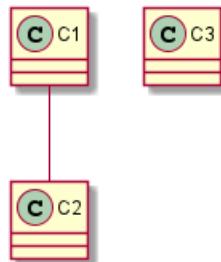
```
@enduml
```



### 3.17 Hide or Remove unlinked class

By default, all classes are displayed:

```
@startuml
class C1
class C2
class C3
C1 -- C2
@enduml
```



But you can:

- hide @unlinked classes:

```
@startuml
class C1
class C2
class C3
C1 -- C2

hide @unlinked
@enduml
```



- or remove @unlinked classes:

```
@startuml
class C1
class C2
class C3
C1 -- C2

remove @unlinked
@enduml
```





[Adapted from QA-11052]

### 3.18 Use generics

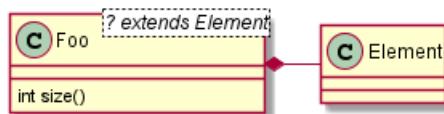
You can also use bracket < and > to define generics usage in a class.

```
@startuml
```

```

class Foo<? extends Element> {
    int size()
}
Foo *-- Element
  
```

```
@enduml
```



It is possible to disable this drawing using `skinparam genericDisplay old` command.

### 3.19 Specific Spot

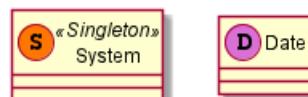
Usually, a spotted character (C, I, E or A) is used for classes, interface, enum and abstract classes.

But you can define your own spot for a class when you define the stereotype, adding a single character and a color, like in this example:

```
@startuml
```

```

class System << (S,#FF7700) Singleton >>
class Date << (D,orchid) >>
@enduml
  
```



### 3.20 Packages

You can define a package using the `package` keyword, and optionally declare a background color for your package (Using a html color code or name).

Note that package definitions can be nested.

```
@startuml
```

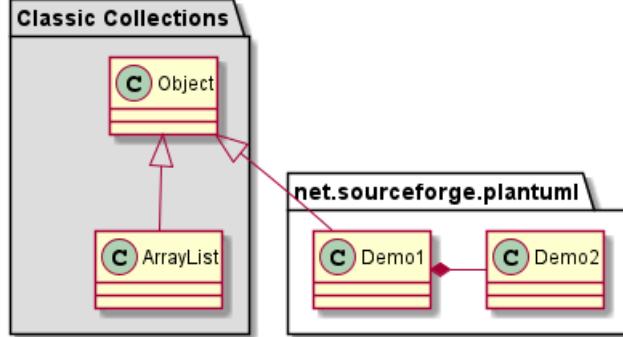
```

package "Classic Collections" #DDDDDD {
    Object <|-- ArrayList
}
  
```



```
package net.sourceforge.plantuml {
    Object <|-- Demo1
    Demo1 *- Demo2
}
```

```
@enduml
```



### 3.21 Packages style

There are different styles available for packages.

You can specify them either by setting a default style with the command : `skinparam packageStyle`, or by using a stereotype on the package:

```
@startuml
scale 750 width
package foo1 <<Node>> {
    class Class1
}

package foo2 <<Rectangle>> {
    class Class2
}

package foo3 <<Folder>> {
    class Class3
}

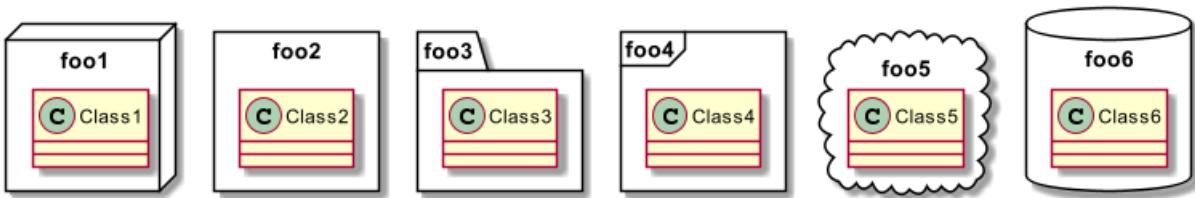
package foo4 <<Frame>> {
    class Class4
}

package foo5 <<Cloud>> {
    class Class5
}

package foo6 <<Database>> {
    class Class6
}

@enduml
```





You can also define links between packages, like in the following example:

```
@startuml

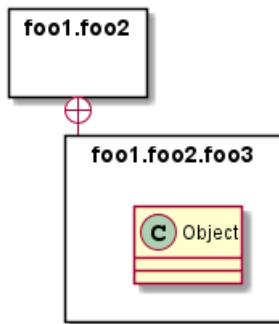
skinparam packageStyle rectangle

package foo1.foo2 {
}

package foo1.foo2.foo3 {
    class Object
}

foo1.foo2 +-- foo1.foo2.foo3

@enduml
```



## 3.22 Namespaces

In packages, the name of a class is the unique identifier of this class. It means that you cannot have two classes with the very same name in different packages.

In that case, you should use namespaces instead of packages.

You can refer to classes from other namespaces by fully qualify them. Classes from the default namespace are qualified with a starting dot.

Note that you don't have to explicitly create namespace : a fully qualified class is automatically put in the right namespace.

```
@startuml

class BaseClass

namespace net.dummy #DDDDDD {
    .BaseClass <|-- Person
    Meeting o-- Person

    .BaseClass <|- Meeting
}

namespace net.foo {
```



```

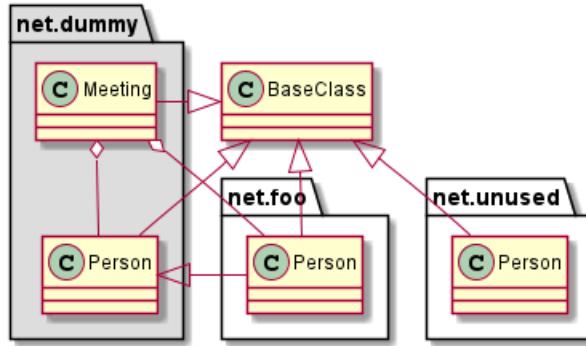
net(dummy).Person <|- Person
.BaseClass <|-- Person

net(dummy).Meeting o-- Person
}

BaseClass <|-- net.unused.Person

@enduml

```



### 3.23 Automatic namespace creation

You can define another separator (other than the dot) using the command : `set namespaceSeparator ???.`

```
@startuml
```

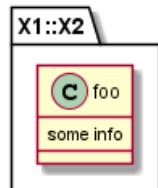
```

set namespaceSeparator ::

class X1::X2::foo {
    some info
}

```

```
@enduml
```



You can disable automatic package creation using the command `set namespaceSeparator none`.

```
@startuml
```

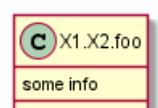
```

set namespaceSeparator none

class X1.X2.foo {
    some info
}

```

```
@enduml
```

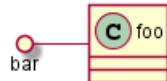


### 3.24 Lollipop interface

You can also define lollipops interface on classes, using the following syntax:

- bar ()- foo
- bar ()-- foo
- foo -() bar

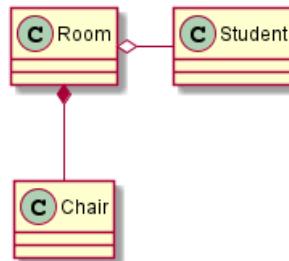
```
@startuml
class foo
bar ()- foo
@enduml
```



### 3.25 Changing arrows direction

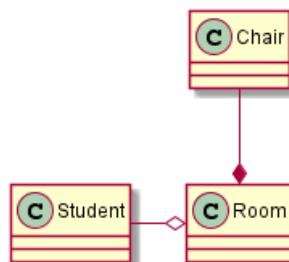
By default, links between classes have two dashes -- and are vertically oriented. It is possible to use horizontal link by putting a single dash (or dot) like this:

```
@startuml
Room o- Student
Room *--- Chair
@enduml
```



You can also change directions by reversing the link:

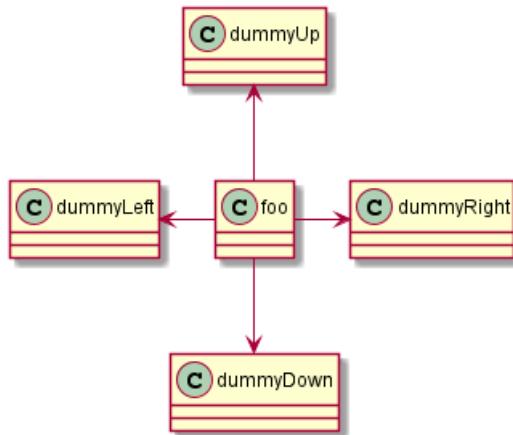
```
@startuml
Student -o Room
Chair --* Room
@enduml
```



It is also possible to change arrow direction by adding `left`, `right`, `up` or `down` keywords inside the arrow:

```
@startuml
foo -left-> dummyLeft
foo -right-> dummyRight
foo -up-> dummyUp
foo -down-> dummyDown
@enduml
```





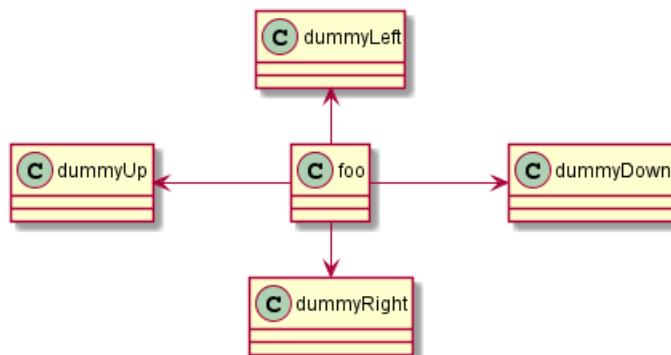
You can shorten the arrow by using only the first character of the direction (for example, `-d-` instead of `-down-`) or the two first characters (`-do-`).

Please note that you should not abuse this functionality : *Graphviz* gives usually good results without tweaking.

And with the `left to right direction` parameter:

```

@startuml
left to right direction
foo -left-> dummyLeft
foo -right-> dummyRight
foo -up-> dummyUp
foo -down-> dummyDown
@enduml
  
```



## 3.26 Association classes

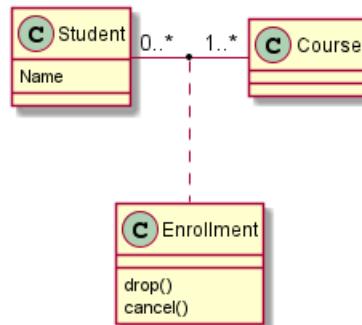
You can define *association class* after that a relation has been defined between two classes, like in this example:

```

@startuml
class Student {
    Name
}
Student "0..*" - "1..*" Course
(Student, Course) .. Enrollment

class Enrollment {
    drop()
    cancel()
}
@enduml
  
```





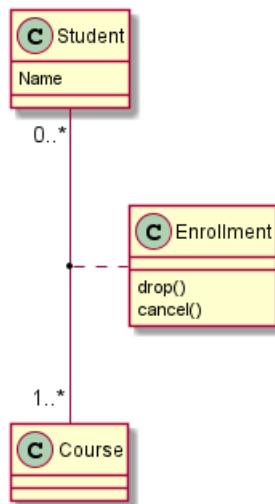
You can define it in another direction:

```

@startuml
class Student {
    Name
}
Student "0..*" -- "1..*" Course
(Student, Course) . Enrollment

class Enrollment {
    drop()
    cancel()
}
@enduml

```



### 3.27 Association on same classe

```

@startuml
class Station {
    +name: string
}

class StationCrossing {
    +cost: TimeInterval
}

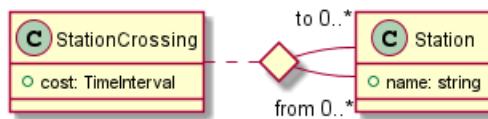
<> diamond

StationCrossing . diamond
diamond - "from 0..*" Station

```



```
diamond - "to 0..*" Station
@enduml
```



[Ref. Incubation: Associations]

### 3.28 Skinparam

You can use the skinparam command to change colors and fonts for the drawing.

You can use this command :

- In the diagram definition, like any other commands,
- In an included file,
- In a configuration file, provided in the command line or the ANT task.

```
@startuml
```

```

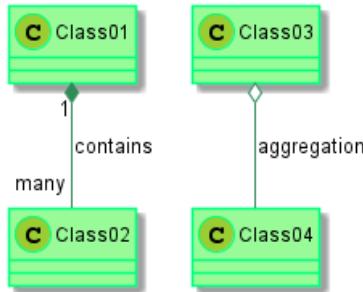
skinparam class {
    BackgroundColor PaleGreen
    ArrowColor SeaGreen
    BorderColor SpringGreen
}
skinparam stereotypeCBackgroundColor YellowGreen

```

```
Class01 "1" *-- "many" Class02 : contains
```

```
Class03 o-- Class04 : aggregation
```

```
@enduml
```



### 3.29 Skinned Stereotypes

You can define specific color and fonts for stereotyped classes.

```
@startuml
```

```

skinparam class {
    BackgroundColor PaleGreen
    ArrowColor SeaGreen
    BorderColor SpringGreen
    BackgroundColor<<Foo>> Wheat
    BorderColor<<Foo>> Tomato
}
skinparam stereotypeCBackgroundColor YellowGreen
skinparam stereotypeCBackgroundColor<< Foo >> DimGray

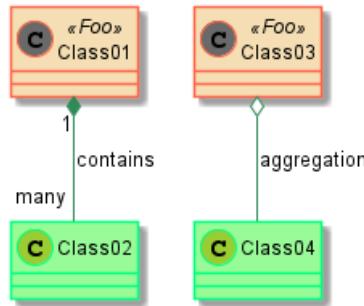
```



```
Class01 <<Foo>>
Class03 <<Foo>>
Class01 "1" *-- "many" Class02 : contains
```

```
Class03 o-- Class04 : aggregation
```

```
@enduml
```



### 3.30 Color gradient

You can declare individual colors for classes, notes etc using the # notation.

You can use standard color names or RGB codes in various notations, see Colors.

You can also use color gradient for background colors, with the following syntax: two colors names separated either by:

- |,
- /,
- \, or
- -

depending on the direction of the gradient.

For example:

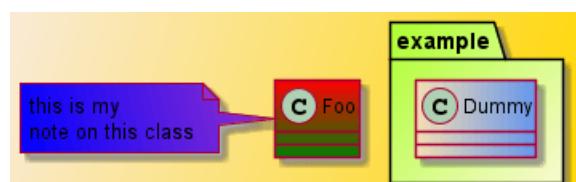
```
@startuml
```

```
skinparam backgroundColor AntiqueWhite/Gold
skinparam classBackgroundColor Wheat|CornflowerBlue
```

```
class Foo #red-green
note left of Foo #blue\9932CC
    this is my
    note on this class
end note
```

```
package example #GreenYellow/LightGoldenRodYellow {
    class Dummy
}
```

```
@enduml
```



### 3.31 Help on layout

Sometimes, the default layout is not perfect...

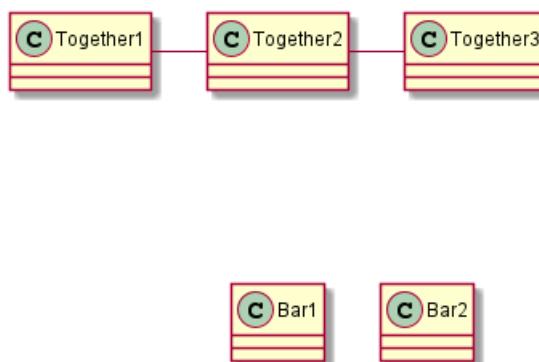
You can use `together` keyword to group some classes together : the layout engine will try to group them (as if they were in the same package).

You can also use `hidden` links to force the layout.

```
@startuml
```

```
class Bar1
class Bar2
together {
    class Together1
    class Together2
    class Together3
}
Together1 - Together2
Together2 - Together3
Together2 -[hidden]--> Bar1
Bar1 -[hidden]> Bar2
```

```
@enduml
```



### 3.32 Splitting large files

Sometimes, you will get some very large image files.

You can use the `page (hpages)x(vpages)` command to split the generated image into several files :

`hpages` is a number that indicated the number of horizontal pages, and `vpages` is a number that indicated the number of vertical pages.

You can also use some specific `skinparam` settings to put borders on splitted pages (see example).

```
@startuml
' Split into 4 pages
page 2x2
skinparam pageMargin 10
skinparam pageExternalColor gray
skinparam pageBorderColor black

class BaseClass

namespace net.dummy #DDDDDD {
    .BaseClass <|-- Person
    Meeting o--- Person
```

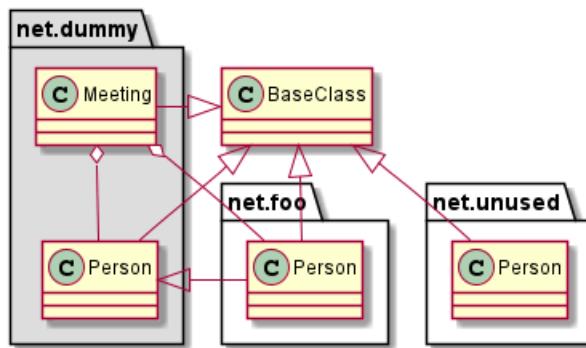


```
.BaseClass <|- Meeting
}

namespace net.foo {
    net.dummy.Person <|- Person
    .BaseClass <|-- Person

    net.dummy.Meeting o-- Person
}

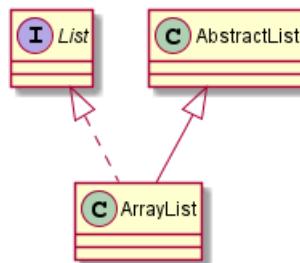
BaseClass <|-- net.unused.Person
@enduml
```



### 3.33 Extends and implements

It is also possible to use **extends** and **implements** keywords.

```
@startuml
class ArrayList implements List
class ArrayList extends AbstractList
@enduml
```



### 3.34 Bracketed relations (linking or arrow) style

#### 3.34.1 Line style

It's also possible to have explicitly **bold**, **dashed**, **dotted**, **hidden** or **plain** relation, links or arrows:

- without label

```
@startuml
title Bracketed line style without label
class foo
class bar
bar1 : [bold]
bar2 : [dashed]
bar3 : [dotted]
```

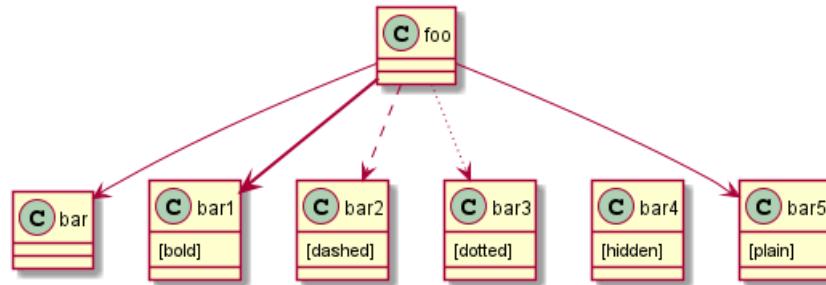


```

bar4 : [hidden]
bar5 : [plain]

foo --> bar
foo -[bold]-> bar1
foo -[dashed]-> bar2
foo -[dotted]-> bar3
foo -[hidden]-> bar4
foo -[plain]-> bar5
@enduml

```

**Bracketed line style without label**

- with label

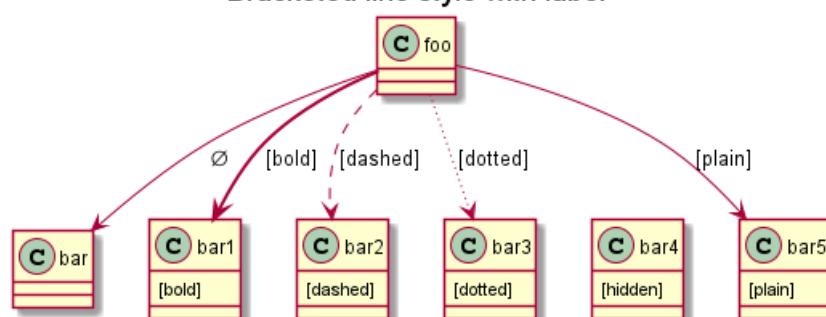
```

@startuml
title Bracketed line style with label
class foo
class bar
bar1 : [bold]
bar2 : [dashed]
bar3 : [dotted]
bar4 : [hidden]
bar5 : [plain]

foo --> bar      :
foo -[bold]-> bar1 : [bold]
foo -[dashed]-> bar2 : [dashed]
foo -[dotted]-> bar3 : [dotted]
foo -[hidden]-> bar4 : [hidden]
foo -[plain]-> bar5 : [plain]

@enduml

```

**Bracketed line style with label**

[Adapted from QA-4181]

### 3.34.2 Line color

```
@startuml
```

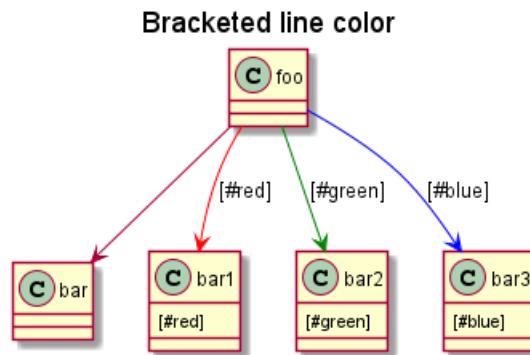


```

title Bracketed line color
class foo
class bar
bar1 : [#red]
bar2 : [#green]
bar3 : [#blue]

foo --> bar
foo -[#red]-> bar1      : [#red]
foo -[#green]-> bar2     : [#green]
foo -[#blue]-> bar3      : [#blue]
'foo -[#blue;#yellow;#green]-> bar4
@enduml

```



### 3.34.3 Line thickness

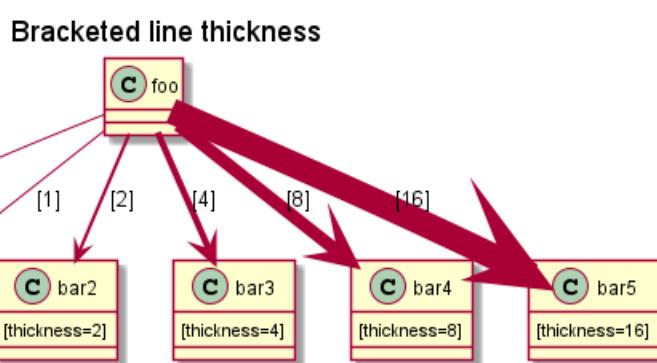
```

@startuml
title Bracketed line thickness
class foo
class bar
bar1 : [thickness=1]
bar2 : [thickness=2]
bar3 : [thickness=4]
bar4 : [thickness=8]
bar5 : [thickness=16]

foo --> bar      :
foo -[thickness=1]-> bar1   : [1]
foo -[thickness=2]-> bar2   : [2]
foo -[thickness=4]-> bar3   : [4]
foo -[thickness=8]-> bar4   : [8]
foo -[thickness=16]-> bar5  : [16]

@enduml

```

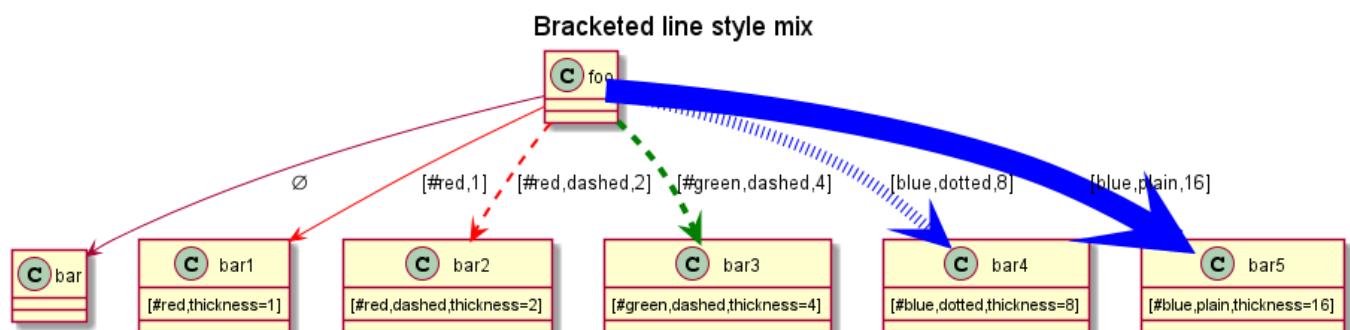


[Ref. QA-4949]

### 3.34.4 Mix

```
@startuml
title Bracketed line style mix
class foo
class bar
bar1 : [#red,thickness=1]
bar2 : [#red,dashed,thickness=2]
bar3 : [#green,dashed,thickness=4]
bar4 : [#blue,dotted,thickness=8]
bar5 : [#blue,plain,thickness=16]

foo --> bar
foo -[#red,thickness=1]-> bar1 : [#red,1]
foo -[#red,dashed,thickness=2]-> bar2 : [#red,dashed,2]
foo -[#green,dashed,thickness=4]-> bar3 : [#green,dashed,4]
foo -[#blue,dotted,thickness=8]-> bar4 : [blue,dotted,8]
foo -[#blue,plain,thickness=16]-> bar5 : [blue,plain,16]
@enduml
```

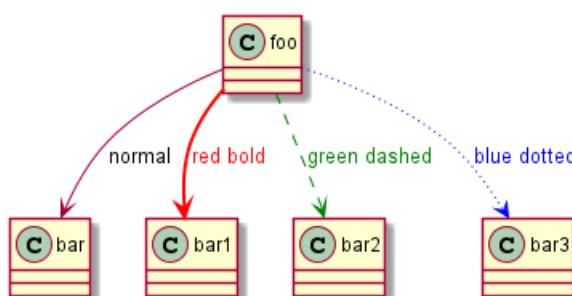


### 3.35 Change relation (linking or arrow) color and style (inline style)

You can change the color or style of individual relation or arrows using the inline following notation:

- #color;line.[bold|dashed|dotted];text:color

```
@startuml
class foo
foo --> bar : normal
foo --> bar1 #line:red;line.bold;text:red : red bold
foo --> bar2 #green;line.dashed;text:green : green dashed
foo --> bar3 #blue;line.dotted;text:blue : blue dotted
@enduml
```



[See similar feature on deployment]



### 3.36 Change class color and style (inline style)

You can change the color or style of individual class using the following notation:

- `#[color|back:color];header:color;line:color;line.[bold|dashed|dotted];text:color`

```
@startuml
abstract abstract
annotation annotation #pink;line:red;line.bold;text:red
class class      #palegreen;line:green;line.dashed;text:green
interface interface #aliceblue;line:blue;line.dotted;text:blue
@enduml
```



First original example:

```
@startuml
class bar #line:green;back:lightblue
class bar2 #lightblue;line:green

class Foo1 #back:red;line:00FFFF
class FooDashed #line.dashed:blue
class FooDotted #line.dotted:blue
class FooBold #line.bold
class Demo1 #back:lightgreen|yellow;header:blue/red
@enduml
```



[Ref. QA-3770]

### 3.37 Arrows from/to class members

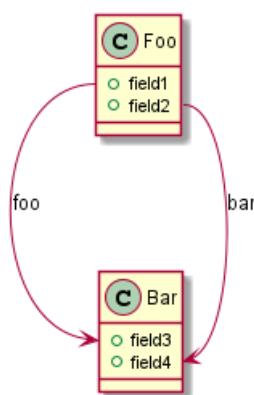
```
@startuml
class Foo {
+ field1
+ field2
}

class Bar {
```



```
+ field3
+ field4
}

Foo::field1 --> Bar::field3 : foo
Foo::field2 --> Bar::field4 : bar
@enduml
```



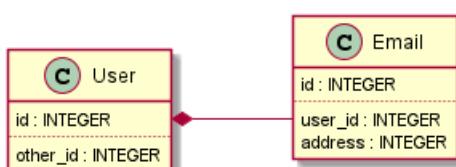
[Ref. QA-3636]

```
@startuml
left to right direction
```

```
class User {
    id : INTEGER
    ..
    other_id : INTEGER
}

class Email {
    id : INTEGER
    ..
    user_id : INTEGER
    address : INTEGER
}

User::id *-- Email::user_id
@enduml
```



[Ref. QA-5261]

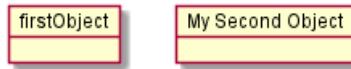


## 4 Object Diagram

### 4.1 Definition of objects

You define instance of objects using the `object` keywords.

```
@startuml
object firstObject
object "My Second Object" as o2
@enduml
```



### 4.2 Relations between objects

Relations between objects are defined using the following symbols :

Type	Symbol	Image
Extension	< --	
Composition	*--	
Aggregation	o--	

It is possible to replace `--` by `..` to have a dotted line.

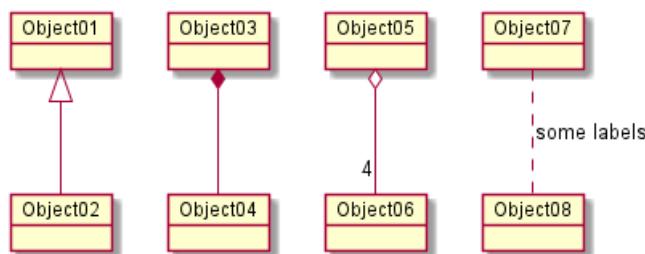
Knowing those rules, it is possible to draw the following drawings.

It is possible to add a label on the relation, using : followed by the text of the label.

For cardinality, you can use double-quotes `"4"` on each side of the relation.

```
@startuml
object Object01
object Object02
object Object03
object Object04
object Object05
object Object06
object Object07
object Object08

Object01 <|-- Object02
Object03 *--- Object04
Object05 o-- "4" Object06
Object07 .. Object08 : some labels
@enduml
```



### 4.3 Associations objects

```
@startuml
object o1
object o2
diamond dia
```



```
object o3
```

```
o1 --> dia
o2 --> dia
dia --> o3
@enduml
```



## 4.4 Adding fields

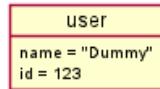
To declare fields, you can use the symbol : followed by the field's name.

```
@startuml
```

```
object user
```

```
user : name = "Dummy"
user : id = 123
```

```
@enduml
```

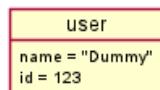


It is also possible to group all fields between brackets {}.

```
@startuml
```

```
object user {
  name = "Dummy"
  id = 123
}
```

```
@enduml
```



## 4.5 Common features with class diagrams

- Hide attributes, methods...
- Defines notes
- Use packages
- Skin the output



## 4.6 Map table or associative array

You can define a map table or associative array, with `map` keyword and `=>` separator.

```
@startuml
map CapitalCity {
    UK => London
    USA => Washington
    Germany => Berlin
}
@enduml
```

CapitalCity	
UK	London
USA	Washington
Germany	Berlin

```
@startuml
map "Map **Country => CapitalCity**" as CC {
    UK => London
    USA => Washington
    Germany => Berlin
}
@enduml
```

Map Country => CapitalCity	
UK	London
USA	Washington
Germany	Berlin

```
@startuml
map "map: Map<Integer, String>" as users {
    1 => Alice
    2 => Bob
    3 => Charlie
}
@enduml
```

map: Map<Integer, String>	
1	Alice
2	Bob
3	Charlie

And add link with object.

```
@startuml
object London

map CapitalCity {
    UK *-> London
    USA => Washington
    Germany => Berlin
}
@enduml
```



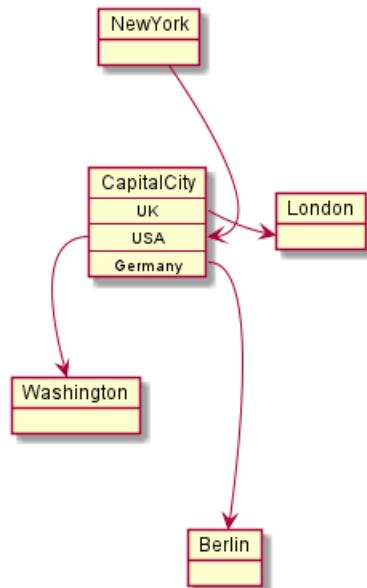
```
@startuml
object London
```



```
object Washington
object Berlin
object NewYork

map CapitalCity {
    UK *-> London
    USA *--> Washington
    Germany *---> Berlin
}

NewYork --> CapitalCity::USA
@enduml
```



[Ref. #307]



## 5 Activity Diagram (legacy)

This is the old **Activity Diagram (legacy)** syntax, to see the new current version see: [Activity Diagram \(new\)](#).

### 5.1 Simple Action

You can use (\*) for the starting point and ending point of the activity diagram.

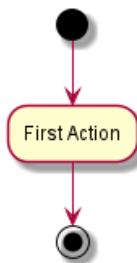
In some occasion, you may want to use (\*top) to force the starting point to be at the top of the diagram.

Use --> for arrows.

```
@startuml
```

```
(*) --> "First Action"
"First Action" --> (*)
```

```
@enduml
```



### 5.2 Label on arrows

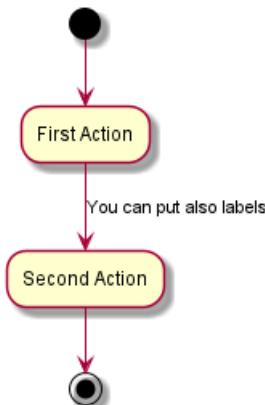
By default, an arrow starts at the last used activity.

You can put a label on an arrow using brackets [ and ] just after the arrow definition.

```
@startuml
```

```
(*) --> "First Action"
-->[You can put also labels] "Second Action"
--> (*)
```

```
@enduml
```



### 5.3 Changing arrow direction

You can use -> for horizontal arrows. It is possible to force arrow's direction using the following syntax:

- -down-> (default arrow)

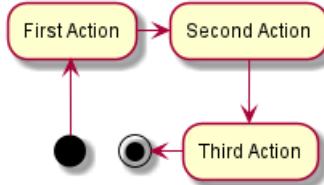


- -right-> or ->
- -left->
- -up->

@startuml

```
(*) -up-> "First Action"
-right-> "Second Action"
--> "Third Action"
-left-> (*)
```

@enduml



## 5.4 Branches

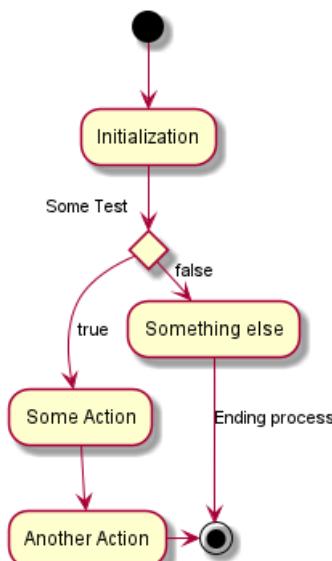
You can use `if/then/else` keywords to define branches.

@startuml

```
(*) --> "Initialization"

if "Some Test" then
    -->[true] "Some Action"
    --> "Another Action"
    -right-> (*)
else
    ->[false] "Something else"
    -->[Ending process] (*)
endif
```

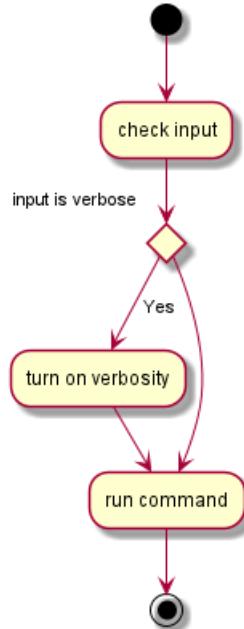
@enduml



Unfortunately, you will have to sometimes repeat the same activity in the diagram text:



```
@startuml
(*) --> "check input"
If "input is verbose" then
--> [Yes] "turn on verbosity"
--> "run command"
else
--> "run command"
Endif
-->(*)
@enduml
```



## 5.5 More on Branches

By default, a branch is connected to the last defined activity, but it is possible to override this and to define a link with the `if` keywords.

It is also possible to nest branches.

```
@startuml

(*) --> if "Some Test" then

  -->[true] "action 1"

  if "" then
    -> "action 3" as a3
  else
    if "Other test" then
      -left-> "action 5"
    else
      --> "action 6"
    endif
  endif

else

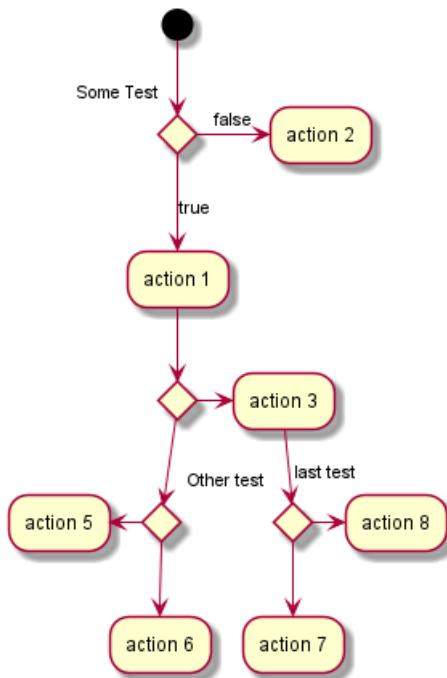
  ->[false] "action 2"

endif
```



```
a3 --> if "last test" then
    --> "action 7"
else
    -> "action 8"
endif

@enduml
```



## 5.6 Synchronization

You can use === code === to display synchronization bars.

```
@startuml
```

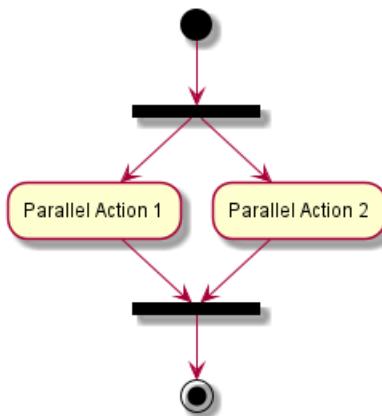
```
(*) --> ===B1===
--> "Parallel Action 1"
--> ===B2===

====B1==== --> "Parallel Action 2"
--> ===B2===

--> (*)
```

```
@enduml
```





## 5.7 Long action description

When you declare activities, you can span on several lines the description text. You can also add `in` in the description.

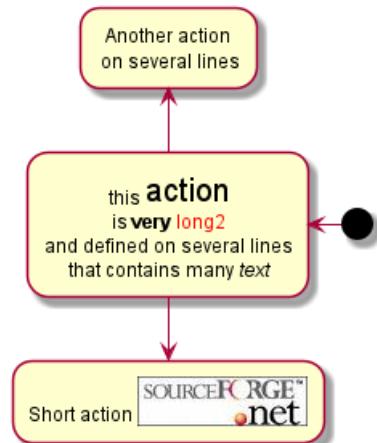
You can also give a short code to the activity with the `as` keyword. This code can be used latter in the diagram description.

```

@startuml
(*) -left-> "this <size:20>action</size>
is <b>very</b> <color:red>long2</color>
and defined on several lines
that contains many <i>text</i>" as A1

-up-> "Another action\n on several lines"

A1 --> "Short action <img:sourceforge.jpg>"
@enduml
  
```



## 5.8 Notes

You can add notes on a activity using the commands `note left`, `note right`, `note top` or `note bottom`, just after the description of the activity you want to note.

If you want to put a note on the starting point, define the note at the very beginning of the diagram description.

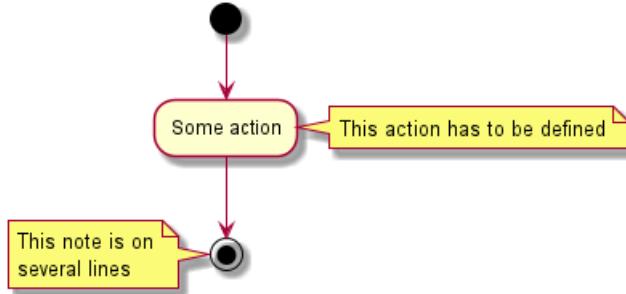
You can also have a note on several lines, using the `endnote` keywords.

```
@startuml
```



```
(*) --> "Some action"
note right: This action has to be defined
"Some action" --> (*)
note left
  This note is on
  several lines
end note

@enduml
```



## 5.9 Partition

You can define a partition using the `partition` keyword, and optionally declare a background color for your partition (Using a html color code or name)

When you declare activities, they are automatically put in the last used partition.

You can close the partition definition using a closing bracket `}`.

```
@startuml
```

```
partition Conductor {
  (*) --> "Climbs on Platform"
  --> === S1 ===
  --> Bows
}

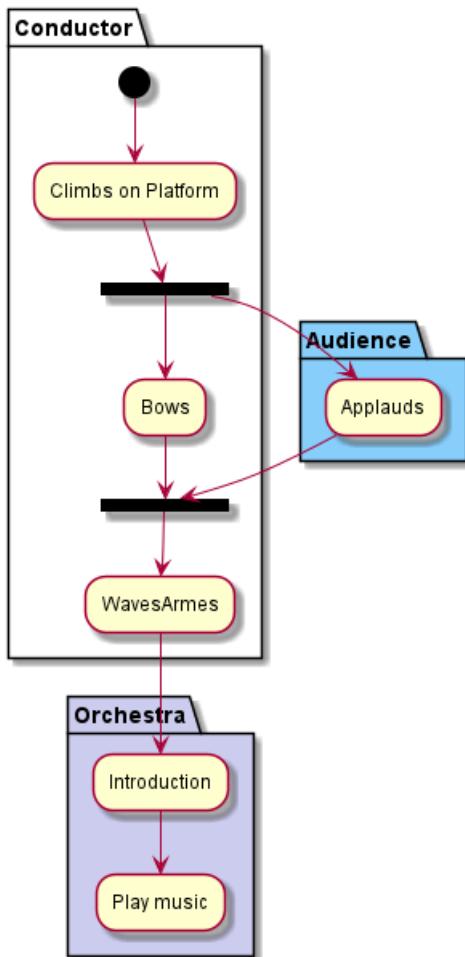
partition Audience #LightSkyBlue {
  === S1 === --> Applauds
}

partition Conductor {
  Bows --> === S2 ===
  --> WavesArmes
  Applauds --> === S2 ===
}

partition Orchestra #CCCCEE {
  WavesArmes --> Introduction
  --> "Play music"
}

@enduml
```





## 5.10 Skinparam

You can use the skinparam command to change colors and fonts for the drawing.

You can use this command :

- In the diagram definition, like any other commands,
- In an included file,
- In a configuration file, provided in the command line or the ANT task.

You can define specific color and fonts for stereotyped activities.

@startuml

```

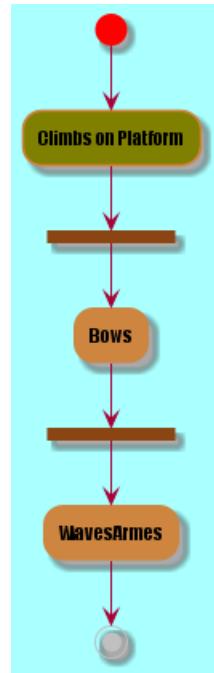
skinparam backgroundColor #AAFFFF
skinparam activity {
    StartColor red
    BarColor SaddleBrown
    EndColor Silver
    BackgroundColor Peru
    BackgroundColor<< Begin >> Olive
    BorderColor Peru
    FontName Impact
}
(*) --> "Climbs on Platform" << Begin >>
--> === S1 ===
--> Bows

```



```
--> === S2 ===
--> WavesArmes
--> (*)
```

@enduml



## 5.11 Octagon

You can change the shape of activities to octagon using the `skinparam activityShape octagon` command.

```
@startuml
'Default is skinparam activityShape roundBox
skinparam activityShape octagon

(*) --> "First Action"
"First Action" --> (*)
```

@enduml



## 5.12 Complete example

```
@startuml
title Servlet Container

(*) --> "ClickServlet.handleRequest()"
--> "new Page"
```



```
if "Page.onSecurityCheck" then
->[true] "Page.onInit()"

if "isForward?" then
->[no] "Process controls"

if "continue processing?" then
-->[yes] ===RENDERING===
else
-->[no] ===REDIRECT_CHECK===
endif

else
-->[yes] ===RENDERING===
endif

if "is Post?" then
-->[yes] "Page.onPost()"
--> "Page.onRender()" as render
--> ===REDIRECT_CHECK===
else
-->[no] "Page.onGet()"
--> render
endif

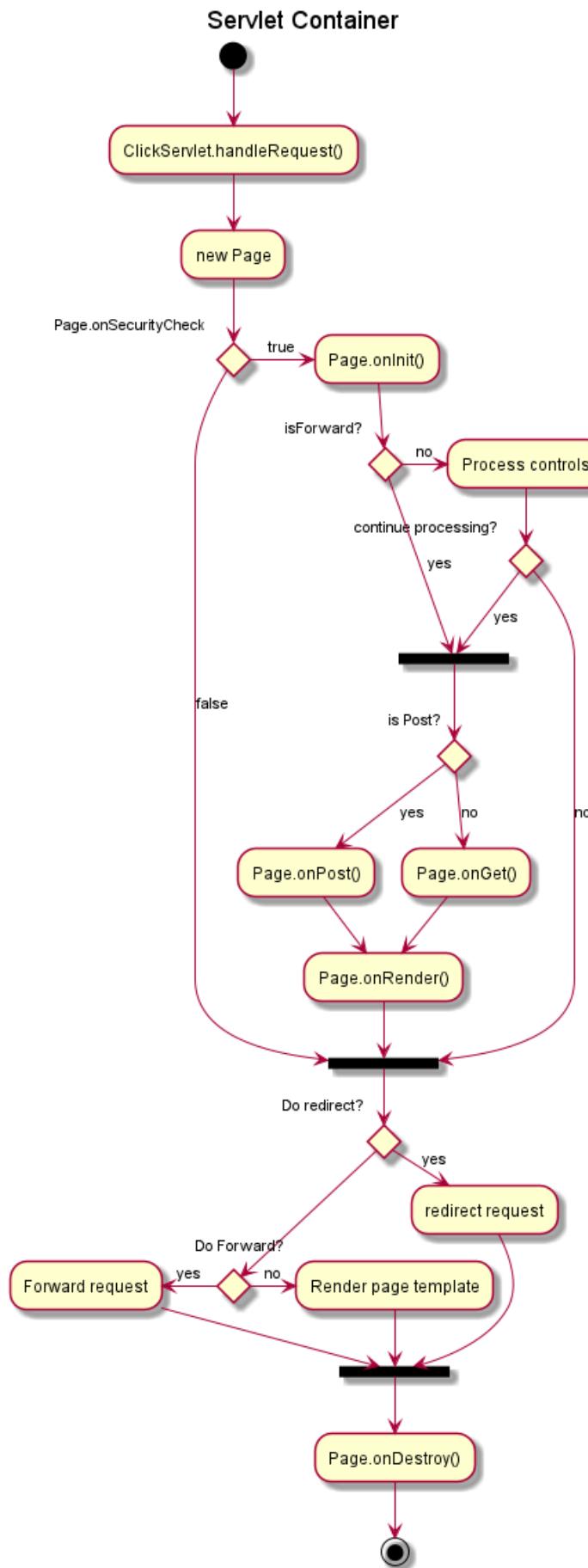
else
-->[false] ===REDIRECT_CHECK===
endif

if "Do redirect?" then
->[yes] "redirect request"
--> ==BEFORE_DESTROY===
else
if "Do Forward?" then
-left->[yes] "Forward request"
--> ==BEFORE_DESTROY===
else
-right->[no] "Render page template"
--> ==BEFORE_DESTROY===
endif
endif

--> "Page.onDestroy()"
-->(*)
```

@enduml





## 6 Activity Diagram (new)

Old syntax for activity diagram had several limitations and drawbacks (for example, it's difficult to maintain).

So a completely new syntax and implementation is now available to users. Another advantage of this implementation is that it's done without the need of having Graphviz installed (as for sequence diagrams).

This syntax will replace the old legacy one. However, for compatibility reason, the old syntax will still be recognized, to ensure *ascending compatibility*.

Users are simply encouraged to migrate to the new syntax.

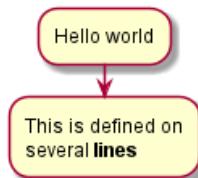
### 6.1 Simple action

Activities label starts with : and ends with ;.

Text formatting can be done using creole wiki syntax.

They are implicitly linked in their definition order.

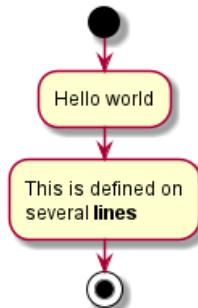
```
@startuml
:Hello world;
:This is defined on
several **lines**;
@enduml
```



### 6.2 Start/Stop/End

You can use **start** and **stop** keywords to denote the beginning and the end of a diagram.

```
@startuml
start
:Hello world;
:This is defined on
several **lines**;
stop
@enduml
```

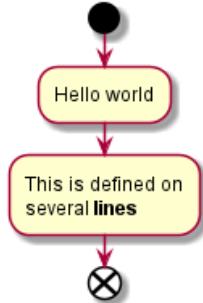


You can also use the **end** keyword.

```
@startuml
start
:Hello world;
:This is defined on
end
```



```
several **lines**;
end
@enduml
```



### 6.3 Conditional

You can use `if`, `then` and `else` keywords to put tests in your diagram. Labels can be provided using parentheses.

The 3 syntaxes are possible:

- `if (...) then (...)`

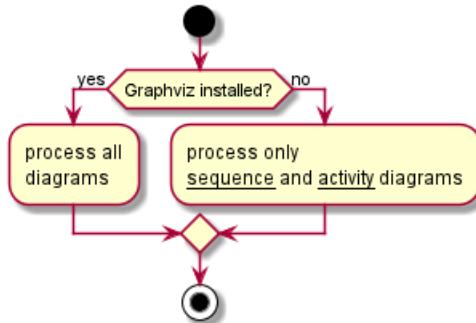
```
@startuml
```

`start`

```
if (Graphviz installed?) then (yes)
    :process all\ndiagrams;
else (no)
    :process only
    __sequence__ and __activity__ diagrams;
endif
```

`stop`

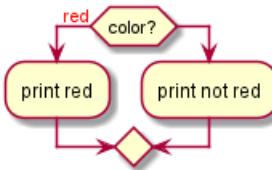
```
@enduml
```



- `if (...) is (...) then`

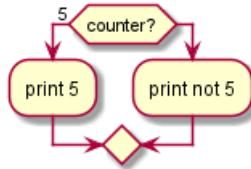
```
@startuml
if (color?) is (<color:red>red) then
:print red;
else
:print not red;
@enduml
```





- if (...) equals (...) then

```
@startuml
if (counter?) equals (5) then
:print 5;
else
:print not 5;
@enduml
```

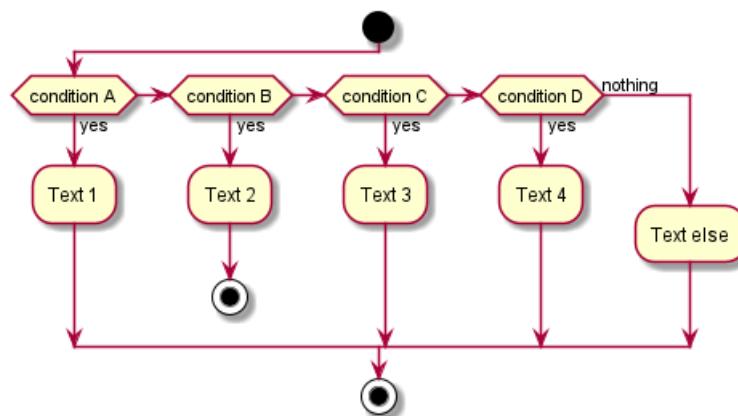


[Ref. QA-301]

### 6.3.1 Several tests (horizontal mode)

You can use the `elseif` keyword to have several tests (*by default, it is the horizontal mode*):

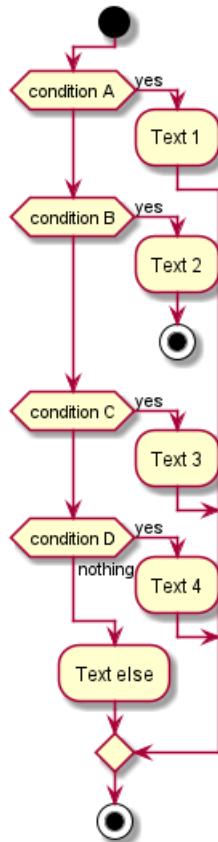
```
@startuml
start
if (condition A) then (yes)
  :Text 1;
elseif (condition B) then (yes)
  :Text 2;
  stop
elseif (condition C) then (yes)
  :Text 3;
elseif (condition D) then (yes)
  :Text 4;
else (nothing)
  :Text else;
endif
stop
@enduml
```



### 6.3.2 Several tests (vertical mode)

You can use the command `!pragma useVerticalIf` on to have the tests in vertical mode:

```
@startuml
!pragma useVerticalIf on
start
if (condition A) then (yes)
    :Text 1;
elseif (condition B) then (yes)
    :Text 2;
    stop
elseif (condition C) then (yes)
    :Text 3;
elseif (condition D) then (yes)
    :Text 4;
else (nothing)
    :Text else;
endif
stop
@enduml
```



[Ref. QA-3931]

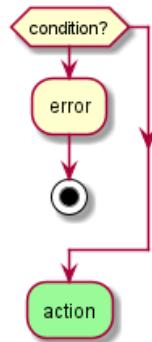
## 6.4 Conditional with stop on an action [kill, detach]

You can stop action on a if loop.

```
@startuml
if (condition?) then
    :error;
    stop
```



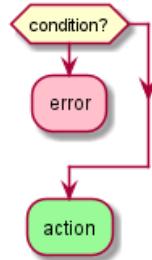
```
endif
#palegreen:action;
@enduml
```



But if you want to stop at an precise action, you can use the `kill` or `detach` keyword:

- `kill`

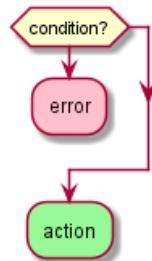
```
@startuml
if (condition?) then
  #pink:error;
  kill
endif
#palegreen:action;
@enduml
```



[Ref. QA-265]

- `detach`

```
@startuml
if (condition?) then
  #pink:error;
  detach
endif
#palegreen:action;
@enduml
```



## 6.5 Repeat loop

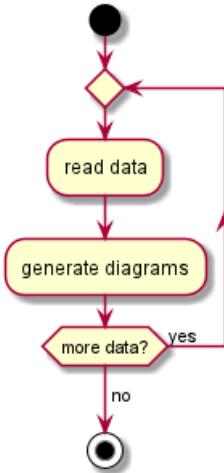
You can use `repeat` and `repeatwhile` keywords to have repeat loops.

```
@startuml
```

```
start
```

```
repeat
  :read data;
  :generate diagrams;
repeat while (more data?) is (yes)
->no;
stop
```

```
@enduml
```



It is also possible to use a full action as `repeat` target and insert an action in the return path using the `backward` keyword.

```
@startuml
```

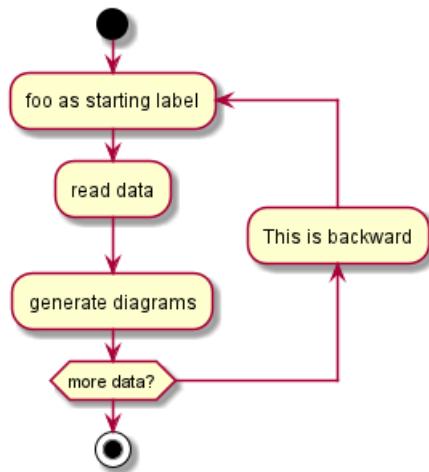
```
start
```

```
repeat :foo as starting label;
  :read data;
  :generate diagrams;
backward:This is backward;
repeat while (more data?)
```

```
stop
```

```
@enduml
```



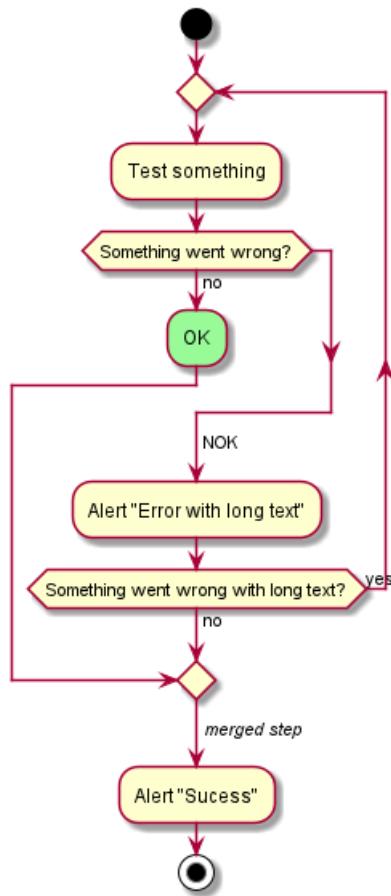


## 6.6 Break on a repeat loop [break]

You can break after an action on a loop.

```
@startuml
start
repeat
    :Test something;
    if (Something went wrong?) then (no)
        #palegreen:OK;
        break
    endif
    ->NOK;
    :Alert "Error with long text";
repeat while (Something went wrong with long text?) is (yes) not (no)
->//merged step//;
:Alert "Sucess";
stop
@enduml
```





[Ref. QA-6105]

## 6.7 While loop

You can use `while` and `end while` keywords to have repeat loops.

@startuml

start

```

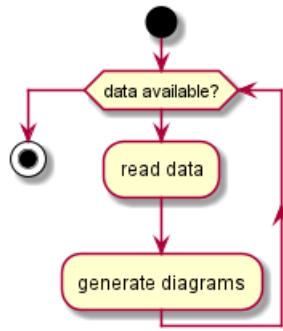
while (data available?)
    :read data;
    :generate diagrams;
endwhile

```

stop

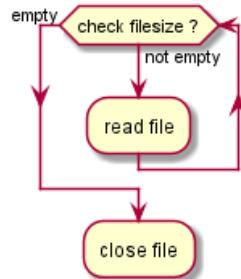
@enduml





It is possible to provide a label after the `endwhile` keyword, or using the `is` keyword.

```
@startuml
while (check filesize ?) is (not empty)
  :read file;
endwhile (empty)
:close file;
@enduml
```



## 6.8 Parallel processing

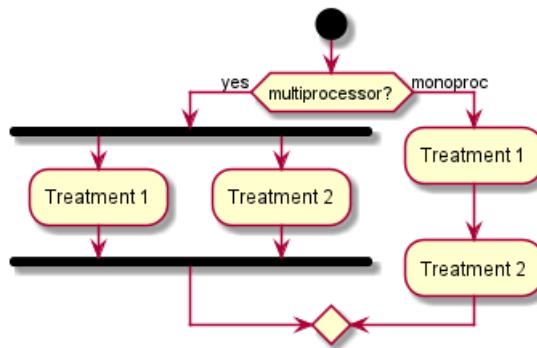
You can use `fork`, `fork again` and `end fork` keywords to denote parallel processing.

```
@startuml
start

if (multiprocessor?) then (yes)
  fork
    :Treatment 1;
  fork again
    :Treatment 2;
  end fork
else (monoproc)
  :Treatment 1;
  :Treatment 2;
endif

@enduml
```





## 6.9 Split processing

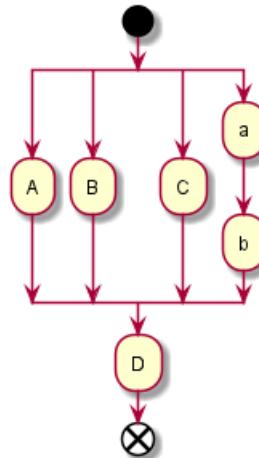
### 6.9.1 Split

You can use `split`, `split again` and `end split` keywords to denote split processing.

```

@startuml
start
split
  :A;
split again
  :B;
split again
  :C;
split again
  :a;
  :b;
end split
:D;
end
@enduml

```



### 6.9.2 Input split (multi-start)

You can use hidden arrows to make an input split (multi-start):

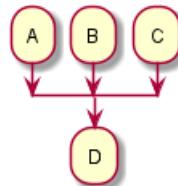
```

@startuml
split
  -[hidden]->
  :A;
split again
  -[hidden]->

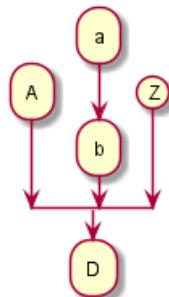
```



```
:B;
split again
-[hidden]->
:C;
end split
:D;
@enduml
```



```
@startuml
split
-[hidden]->
:A;
split again
-[hidden]->
:a;
:b;
split again
-[hidden]->
(Z)
end split
:D;
@enduml
```



[Ref. QA-8662]

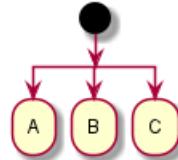
### 6.9.3 Output split (multi-end)

You can use `kill` or `detach` to make an output split (multi-end):

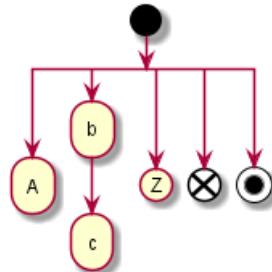
```
@startuml
start
split
:A;
kill
split again
:B;
detach
split again
:C;
kill
```



```
end split
@enduml
```



```
@startuml
start
split
  :A;
  kill
split again
  :b;
  :c;
  detach
split again
  (Z)
  detach
split again
  end
split again
  stop
end split
@enduml
```



## 6.10 Notes

Text formatting can be done using creole wiki syntax.

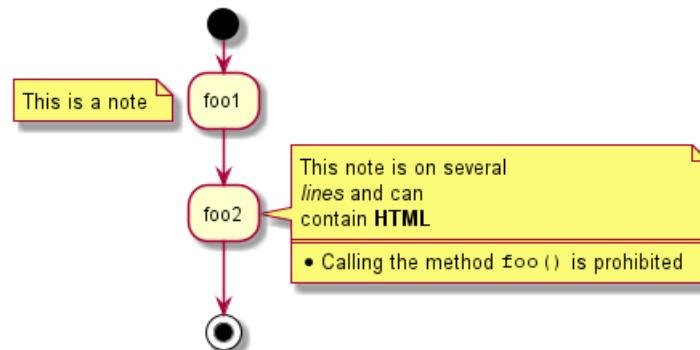
A note can be floating, using `floating` keyword.

```
@startuml

start
:foo1;
floating note left: This is a note
:foo2;
note right
  This note is on several
  //lines// and can
  contain <b>HTML</b>
=====
  * Calling the method ""foo()"" is prohibited
end note
stop
```

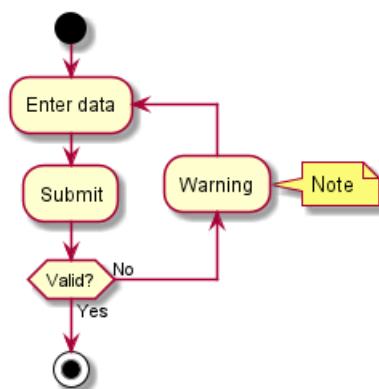


```
@enduml
```



You can add note on backward activity.

```
@startuml
start
repeat :Enter data;
:Submit;
backward :Warning;
note right: Note
repeat while (Valid?) is (No) not (Yes)
stop
@enduml
```



[Ref. QA-11788]

## 6.11 Colors

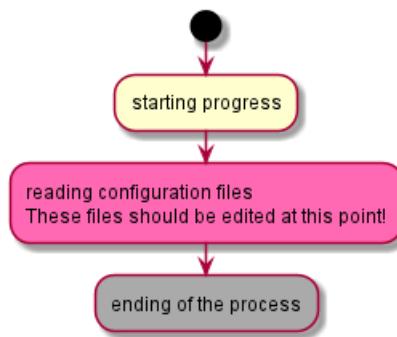
You can specify a color for some activities.

```
@startuml

start
:starting progress;
#HotPink:reading configuration files
These files should be edited at this point!;
#AAAAAA:ending of the process;

@enduml
```



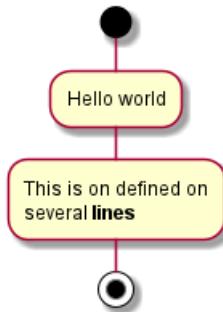


## 6.12 Lines without arrows

You can use `skinparam ArrowHeadColor none` in order to connect activities using lines only, without arrows.

```

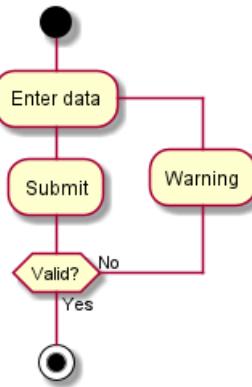
@startuml
skinparam ArrowHeadColor none
start
:Hello world;
:This is on defined on
several **lines**;
stop
@enduml
  
```



```

@startuml
skinparam ArrowHeadColor none
start
repeat :Enter data;
:Submit;
backward :Warning;
repeat while (Valid?) is (No) not (Yes)
stop
@enduml
  
```





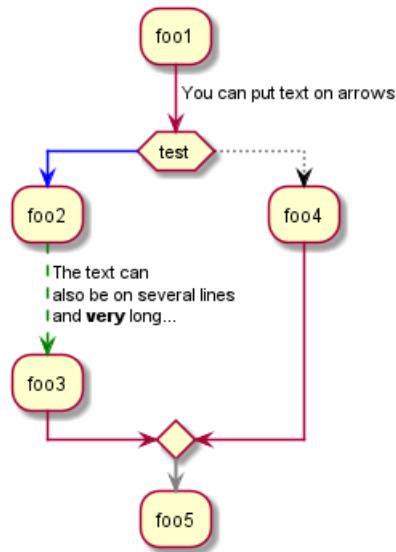
### 6.13 Arrows

Using the `->` notation, you can add texts to arrow, and change their color.

It's also possible to have dotted, dashed, bold or hidden arrows.

```

@startuml
:foo1;
-> You can put text on arrows;
if (test) then
  -[#blue]->
  :foo2;
  -[#green,dashed]-> The text can
  also be on several lines
  and **very** long...;
  :foo3;
else
  -[#black,dotted]->
  :foo4;
endif
-[#gray,bold]->
:foo5;
@enduml
  
```

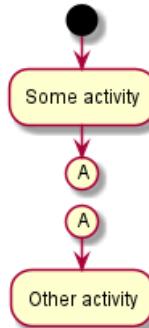


### 6.14 Connector

You can use parentheses to denote connector.



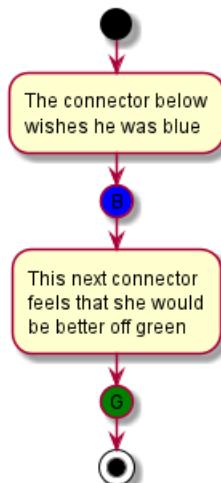
```
@startuml
start
:Some activity;
(A)
detach
(A)
:Other activity;
@enduml
```



## 6.15 Color on connector

You can add color on connector.

```
@startuml
start
:The connector below
wishes he was blue;
#blue:(B)
:This next connector
feels that she would
be better off green;
#green:(G)
stop
@enduml
```



[Ref. QA-10077]

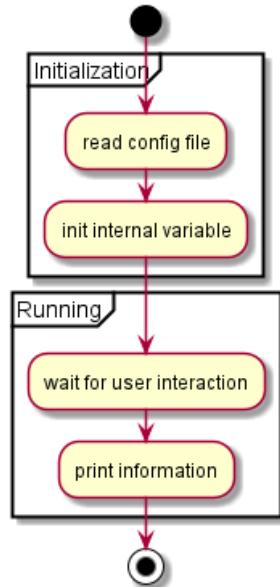
## 6.16 Grouping or partition

You can group activity together by defining partition:



```
@startuml
start
partition Initialization {
    :read config file;
    :init internal variable;
}
partition Running {
    :wait for user interaction;
    :print information;
}

stop
@enduml
```



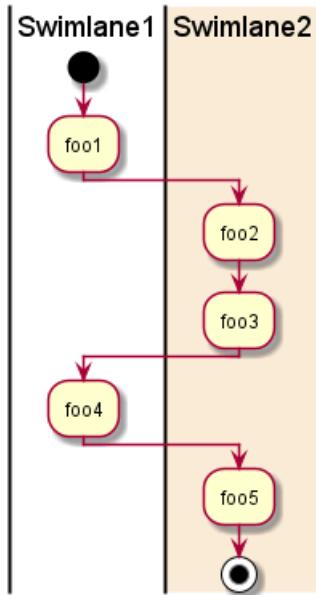
## 6.17 Swimlanes

Using pipe |, you can define swimlanes.

It's also possible to change swimlanes color.

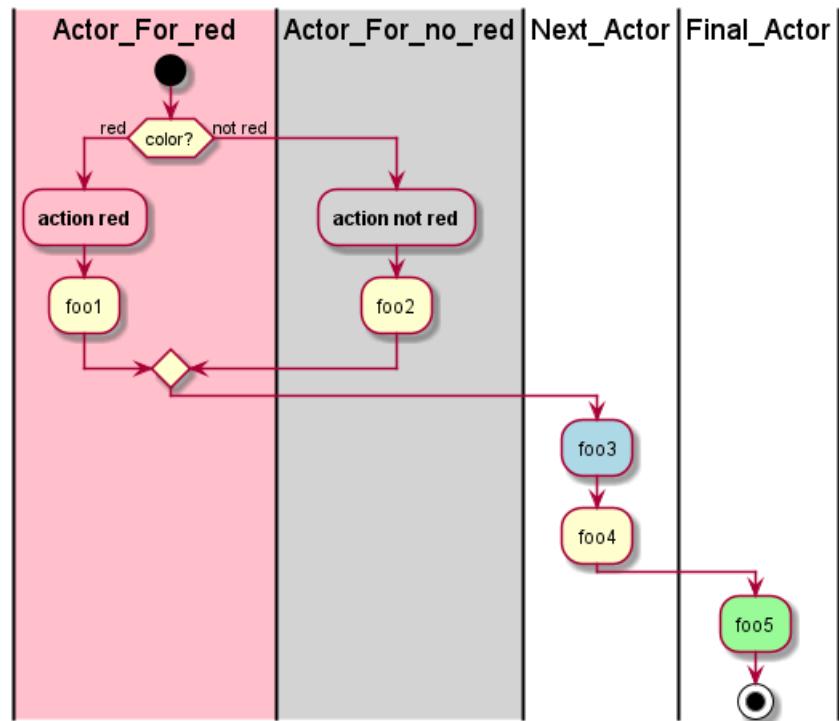
```
@startuml
|Swimlane1|
start
:foo1;
|#AntiqueWhite|Swimlane2|
:foo2;
:foo3;
|Swimlane1|
:foo4;
|Swimlane2|
:foo5;
stop
@enduml
```





You can add **if** conditional or **repeat** or **while** loop within swimlanes.

```
@startuml
|#pink|Actor_For_red|
start
if (color?) is (red) then
#pink:**action red**;
:foo1;
else (not red)
|#lightgray|Actor_For_no_red|
#lightgray:**action not red**;
:foo2;
endif
|Next_Actor|
#lightblue:foo3;
:foo4;
|Final_Actor|
#palegreen:foo5;
stop
@enduml
```



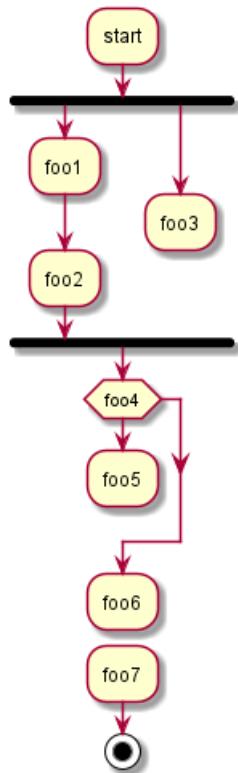
## 6.18 Detach or kill [detach, kill]

It's possible to remove an arrow using the detach or kill keyword:

- detach

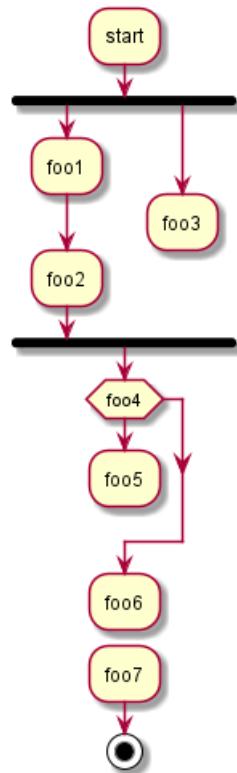
```
@startuml
:start;
fork
:foo1;
:foo2;
fork again
:foo3;
detach
endfork
if (foo4) then
:foo5;
detach
endif
:foo6;
detach
:foo7;
stop
@enduml
```





- kill

```
@startuml
:start;
fork
:foo1;
:foo2;
fork again
:foo3;
kill
endfork
if (foo4) then
:foo5;
kill
endif
:foo6;
kill
:foo7;
stop
@enduml
```



## 6.19 SDL (Specification and Description Language)

By changing the final ; separator, you can set different rendering for the activity:

- |
- <
- >
- /
- \\
- ]
- }

```

@startuml
:Ready;
:next(o)| 
:Receiving;
split
  :nak(i)<
  :ack(o)>
split again
  :ack(i)<
  :next(o)
on several lines|
  :i := i + 1]
  :ack(o)>
split again
  :err(i)<
  :nak(o)>
split again
  :foo/

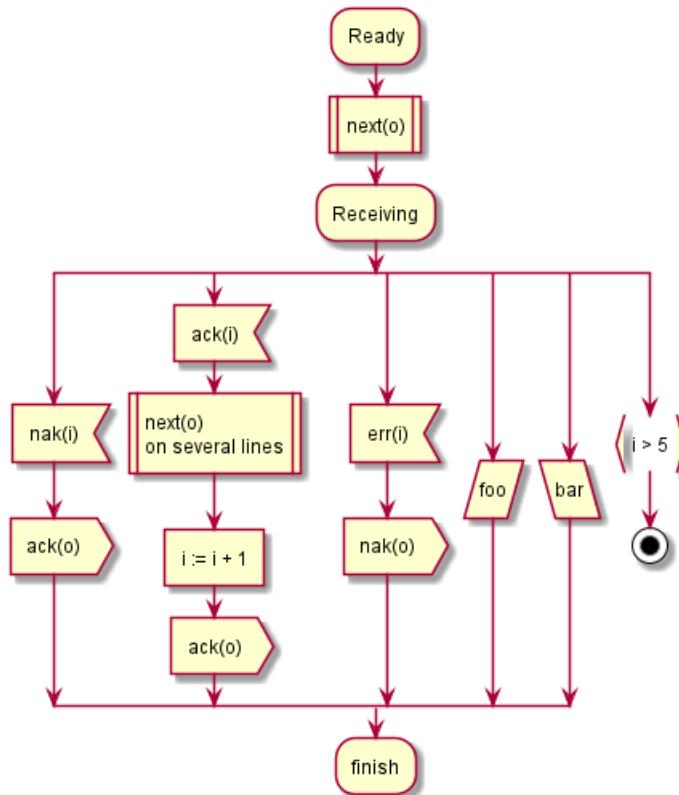
```



```

split again
:bar\\
split again
:i > 5}
stop
end split
:finish;
@enduml

```



## 6.20 Complete example

@startuml

```

start
:ClickServlet.handleRequest();
:new page;
if (Page.onSecurityCheck) then (true)
:Page.onInit();
if (isForward?) then (no)
:Process controls;
if (continue processing?) then (no)
stop
endif

if (isPost?) then (yes)
:Page.onPost();
else (no)
:Page.onGet();
endif
:Page.onRender();
endif
else (false)

```



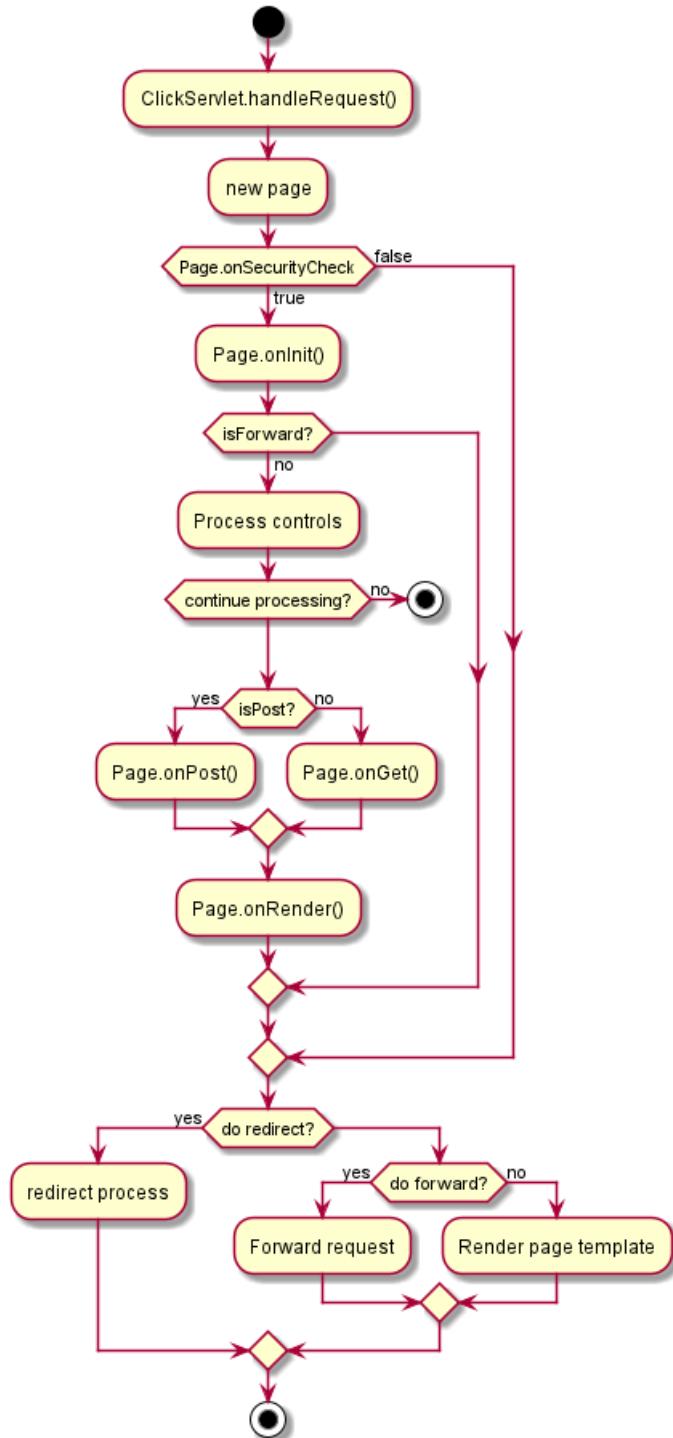
```
endif

if (do redirect?) then (yes)
    :redirect process;
else
    if (do forward?) then (yes)
        :Forward request;
    else (no)
        :Render page template;
    endif
endif

stop

@enduml
```





## 6.21 Condition Style

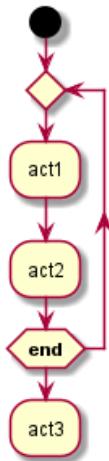
### 6.21.1 Inside style (by default)

```

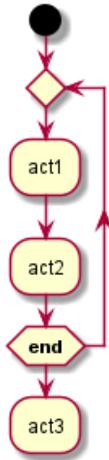
@startuml
skinparam conditionStyle inside
start
repeat
  :act1;
  :act2;
repeatwhile (<b>end</b>)
  :act3;
  
```



@enduml



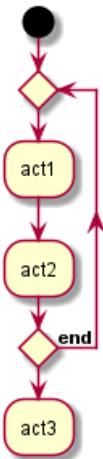
```
@startuml
start
repeat
:act1;
:act2;
repeatwhile (<b>end</b>)
:act3;
@enduml
```



### 6.21.2 Diamond style

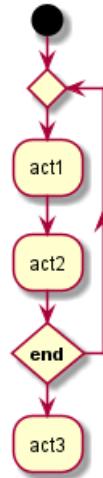
```
@startuml
skinparam conditionStyle diamond
start
repeat
:act1;
:act2;
repeatwhile (<b>end</b>)
:act3;
@enduml
```



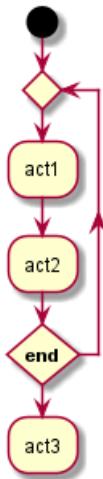


### 6.21.3 InsideDiamond (or *Foo1*) style

```
@startuml
skinparam conditionStyle InsideDiamond
start
repeat
  :act1;
  :act2;
repeatwhile (<b>end</b>)
  :act3;
@enduml
```



```
@startuml
skinparam conditionStyle foo1
start
repeat
  :act1;
  :act2;
repeatwhile (<b>end</b>)
  :act3;
@enduml
```



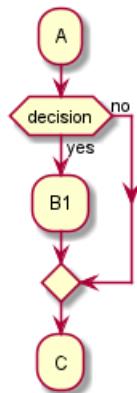
[Ref. QA-1290 and #400]

## 6.22 Condition End Style

### 6.22.1 Diamond style (by default)

- With one branch

```
@startuml
skinparam ConditionEndStyle diamond
:A;
if (decision) then (yes)
  :B1;
else (no)
endif
:C;
@enduml
```

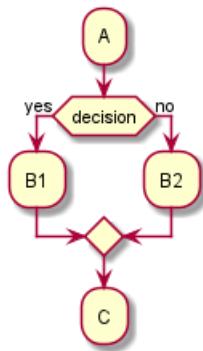


- With two branches (B1, B2)

```
@startuml
skinparam ConditionEndStyle diamond
:A;
if (decision) then (yes)
  :B1;
else (no)
  :B2;
endif
:C;
@enduml
```



@enduml

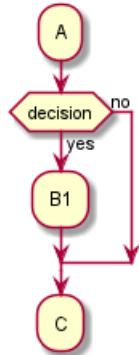


### 6.22.2 Horizontal line (hline) style

- With one branch

```

@startuml
skinparam ConditionEndStyle hline
:A;
if (decision) then (yes)
  :B1;
else (no)
endif
:C;
@enduml
  
```

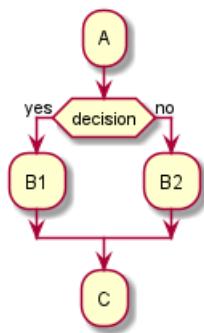


- With two branches (B1, B2)

```

@startuml
skinparam ConditionEndStyle hline
:A;
if (decision) then (yes)
  :B1;
else (no)
  :B2;
endif
:C;
@enduml
@enduml
  
```





[Ref. QA-4015]

## 7 Component Diagram

Let's have few examples.

### 7.1 Components

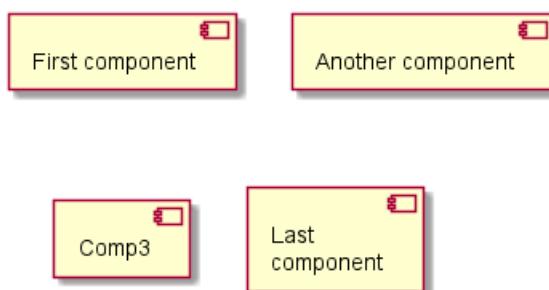
Components must be bracketed.

You can also use the `component` keyword to define a component. And you can define an alias, using the `as` keyword. This alias will be used later, when defining relations.

```
@startuml
```

```
[First component]
[Another component] as Comp2
component Comp3
component [Last\ncomponent] as Comp4
```

```
@enduml
```



### 7.2 Interfaces

Interface can be defined using the `()` symbol (because this looks like a circle).

You can also use the `interface` keyword to define an interface. And you can define an alias, using the `as` keyword. This alias will be used latter, when defining relations.

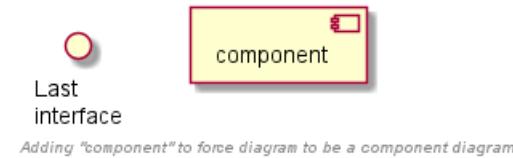
We will see latter that interface definition is optional.

```
@startuml
```

```
() "First Interface"
() "Another interface" as Interf2
interface Interf3
interface "Last\ninterface" as Interf4

[component]
footer //Adding "component" to force diagram to be a **component diagram**//
```

```
@enduml
```



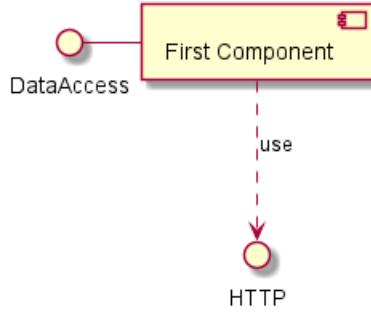
### 7.3 Basic example

Links between elements are made using combinations of dotted line (..), straight line (--), and arrows (-->) symbols.

```
@startuml
```

```
DataAccess - [First Component]
[First Component] ..> HTTP : use
```

```
@enduml
```



### 7.4 Using notes

You can use the `note left of`, `note right of`, `note top of`, `note bottom of` keywords to define notes related to a single object.

A note can be also define alone with the `note` keywords, then linked to other objects using the `..` symbol.

```
@startuml
```

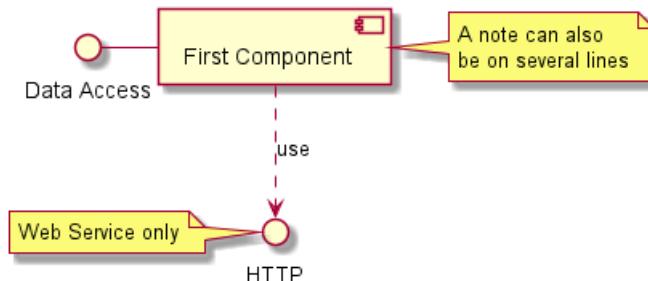
```
interface "Data Access" as DA

DA - [First Component]
[First Component] ..> HTTP : use

note left of HTTP : Web Service only

note right of [First Component]
  A note can also
  be on several lines
end note
```

```
@enduml
```



### 7.5 Grouping Components

You can use several keywords to group components and interfaces together:

- package



- node
- folder
- frame
- cloud
- database

```
@startuml
```

```
package "Some Group" {
    HTTP - [First Component]
    [Another Component]
}

node "Other Groups" {
    FTP - [Second Component]
    [First Component] --> FTP
}

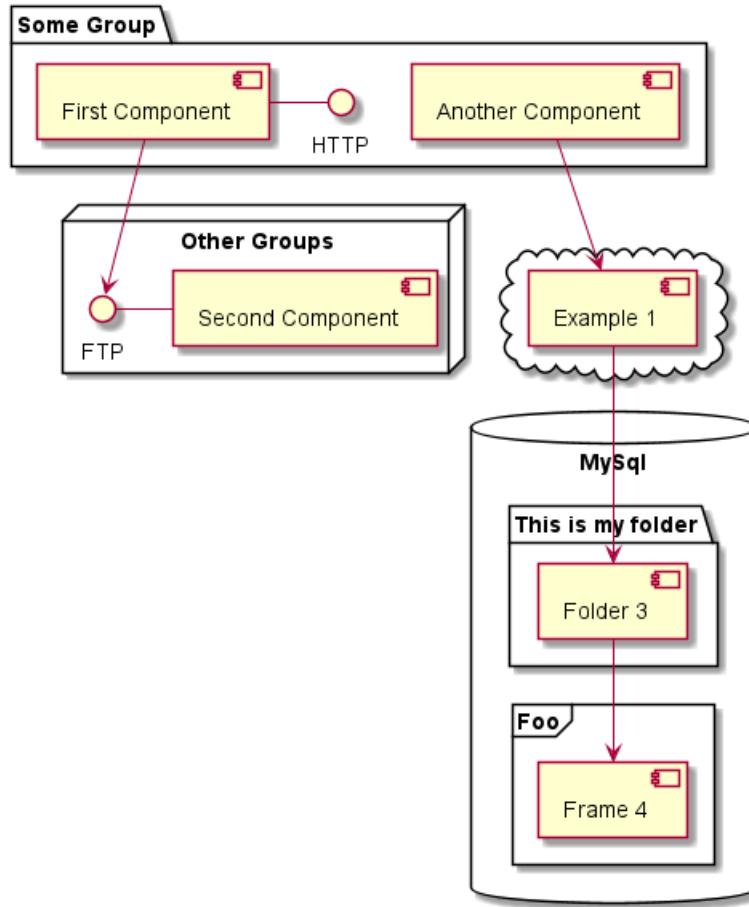
cloud {
    [Example 1]
}

database "MySql" {
    folder "This is my folder" {
        [Folder 3]
    }
    frame "Foo" {
        [Frame 4]
    }
}

[Another Component] --> [Example 1]
[Example 1] --> [Folder 3]
[Folder 3] --> [Frame 4]
```

```
@enduml
```

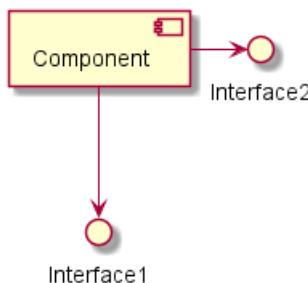




## 7.6 Changing arrows direction

By default, links between classes have two dashes -- and are vertically oriented. It is possible to use horizontal link by putting a single dash (or dot) like this:

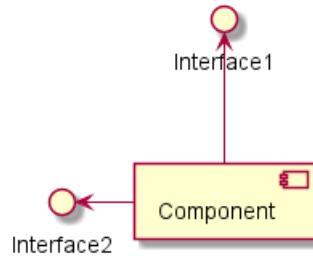
```
@startuml
[Component] --> Interface1
[Component] -> Interface2
@enduml
```



You can also change directions by reversing the link:

```
@startuml
Interface1 <-- [Component]
Interface2 <- [Component]
@enduml
```

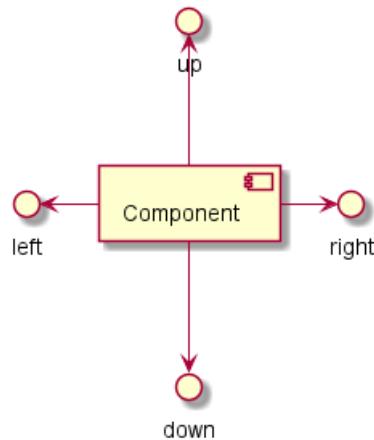




It is also possible to change arrow direction by adding `left`, `right`, `up` or `down` keywords inside the arrow:

```

@startuml
[Component] -left-> left
[Component] -right-> right
[Component] -up-> up
[Component] -down-> down
@enduml
  
```



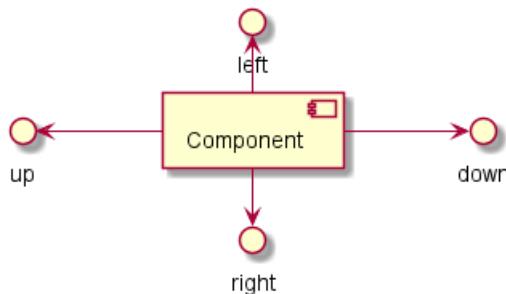
You can shorten the arrow by using only the first character of the direction (for example, `-d-` instead of `-down-`) or the two first characters (`-do-`).

Please note that you should not abuse this functionality : *Graphviz* gives usually good results without tweaking.

And with the `left to right direction` parameter:

```

@startuml
left to right direction
[Component] -left-> left
[Component] -right-> right
[Component] -up-> up
[Component] -down-> down
@enduml
  
```



## 7.7 Use UML2 notation

By default (*from v1.2020.13-14*), UML2 notation is used.

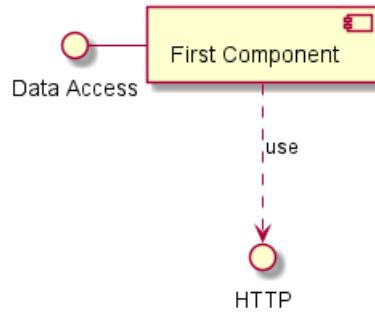
```
@startuml
```

```
interface "Data Access" as DA
```

```
DA - [First Component]
```

```
[First Component] ..> HTTP : use
```

```
@enduml
```



## 7.8 Use UML1 notation

The `skinparam componentStyle uml1` command is used to switch to UML1 notation.

```
@startuml
```

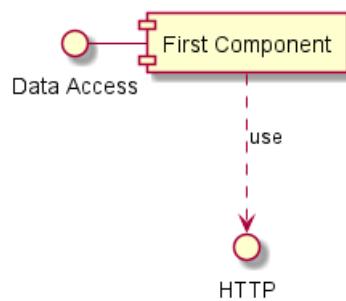
```
skinparam componentStyle uml1
```

```
interface "Data Access" as DA
```

```
DA - [First Component]
```

```
[First Component] ..> HTTP : use
```

```
@enduml
```



## 7.9 Use rectangle notation (remove UML notation)

The `skinparam componentStyle rectangle` command is used to switch to rectangle notation (*without any UML notation*).

```
@startuml
```

```
skinparam componentStyle rectangle
```

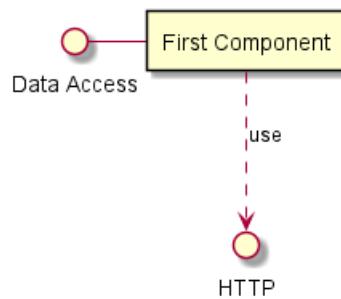
```
interface "Data Access" as DA
```

```
DA - [First Component]
```

```
[First Component] ..> HTTP : use
```



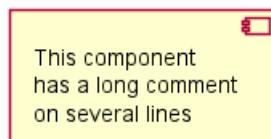
```
@enduml
```



## 7.10 Long description

It is possible to put description on several lines using square brackets.

```
@startuml
component comp1 [
This component
has a long comment
on several lines
]
@enduml
```



## 7.11 Individual colors

You can specify a color after component definition.

```
@startuml
component [Web Server] #Yellow
@enduml
```



## 7.12 Using Sprite in Stereotype

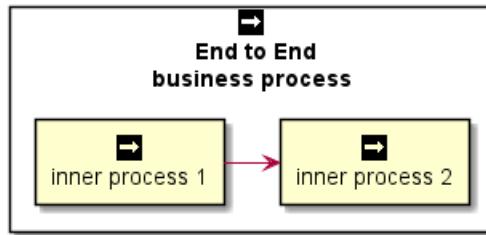
You can use sprites within stereotype components.

```
@startuml
sprite $businessProcess [16x16/16] {
FFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFOFFFFF
FFFFFFFFFFFFFOFFFFF
FF000000000000FF
FF0000000000000FF
FF0000000000000FFF
FFFFFFFFFFFFFOFFFFF
FF0000000000000FFF
FFFFFFFFFFFFFOFFFFF
FFFFFFFFFFFFFOFFFFF
FFFFFFFFFFFFFOFFFFF
FFFFFFFFFFFFFOFFFFF
}
```



```
FFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFF
}
```

```
rectangle " End to End\nbusiness process" <<$businessProcess>> {
    rectangle "inner process 1" <<$businessProcess>> as src
    rectangle "inner process 2" <<$businessProcess>> as tgt
    src -> tgt
}
@enduml
```



## 7.13 Skinparam

You can use the skinparam command to change colors and fonts for the drawing.

You can use this command :

- In the diagram definition, like any other commands;
- In an included file;
- In a configuration file, provided in the command line or the Ant task.

You can define specific color and fonts for stereotyped components and interfaces.

```
@startuml
```

```
skinparam interface {
    backgroundColor RosyBrown
    borderColor orange
}

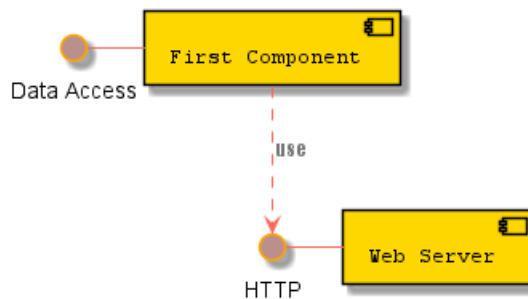
skinparam component {
    FontSize 13
    BackgroundColor<<Apache>> Red
    BorderColor<<Apache>> #FF6655
    FontName Courier
    BorderColor black
    BackgroundColor gold
    ArrowFontName Impact
    ArrowColor #FF6655
    ArrowFontColor #777777
}

() "Data Access" as DA

DA - [First Component]
[First Component] ..> () HTTP : use
HTTP - [Web Server] << Apache >>
```



```
@enduml
```



```
@startuml
```

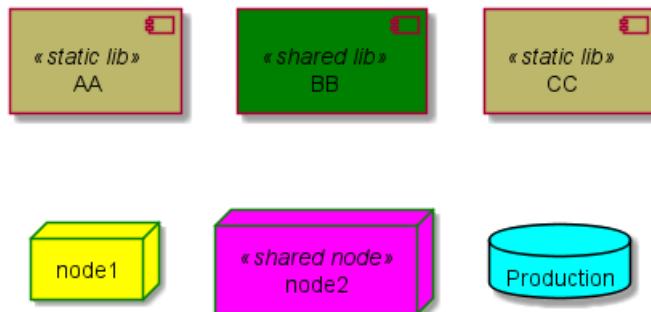
```
[AA] <<static lib>>
[BB] <<shared lib>>
[CC] <<static lib>>
```

```
node node1
node node2 <<shared node>>
database Production
```

```
skinparam component {
    backgroundColor<<static lib>> DarkKhaki
    backgroundColor<<shared lib>> Green
}
```

```
skinparam node {
borderColor Green
backgroundColor Yellow
backgroundColor<<shared node>> Magenta
}
skinparam databaseBackgroundColor Aqua
```

```
@enduml
```



## 7.14 Specific SkinParameter

### 7.14.1 componentStyle

- By default (or with `skinparam componentStyle uml2`), you have an icon for component

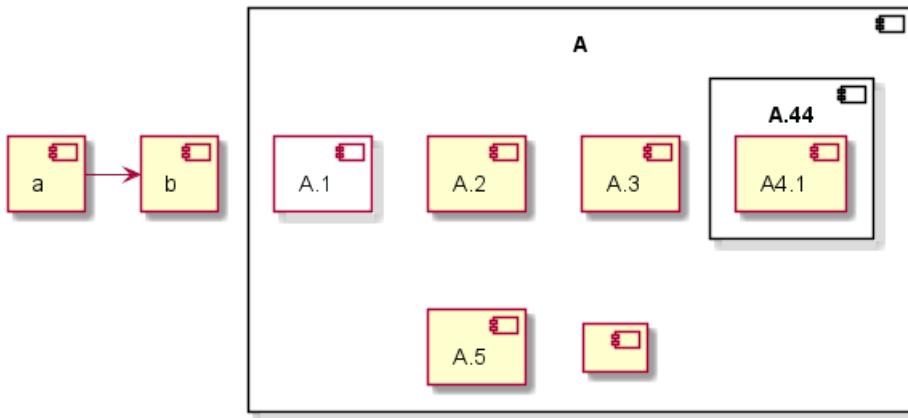
```
@startuml
skinparam BackgroundColor transparent
skinparam componentStyle uml2
component A {
    component "A.1" {
    }
    component A.44 {
        [A4.1]
```



```

}
component "A.2"
[A.3]
component A.5 [
A.5]
component A.6 [
]
}
[a]->[b]
@enduml

```

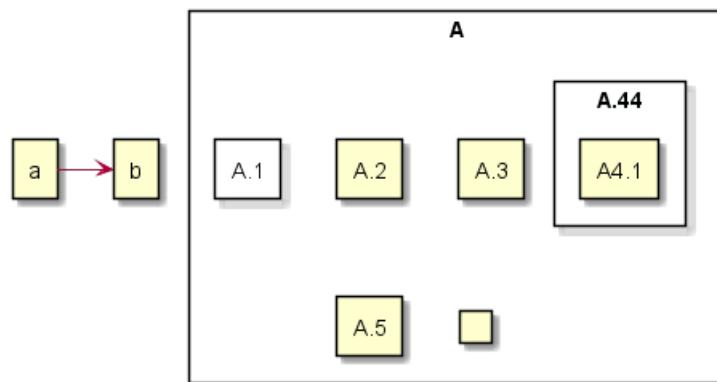


- If you want to suppress it, and to have only the rectangle, you can use `skinparam componentStyle rectangle`

```

@startuml
skinparam BackgroundColor transparent
skinparam componentStyle rectangle
component A {
    component "A.1" {
    }
    component A.44 {
        [A4.1]
    }
    component "A.2"
    [A.3]
    component A.5 [
A.5]
    component A.6 [
    ]
}
[a]->[b]
@enduml

```

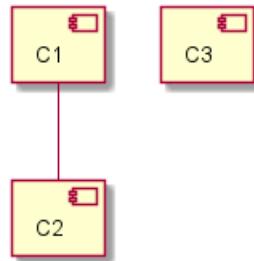


[Ref. 10798]

## 7.15 Hide or Remove unlinked component

By default, all components are displayed:

```
@startuml
component C1
component C2
component C3
C1 -- C2
@enduml
```



But you can:

- hide @unlinked components:

```
@startuml
component C1
component C2
component C3
C1 -- C2

hide @unlinked
@enduml
```

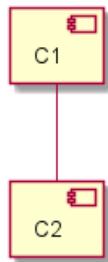


- or remove @unlinked components:

```
@startuml
component C1
component C2
component C3
C1 -- C2

remove @unlinked
@enduml
```





[Ref. QA-11052]

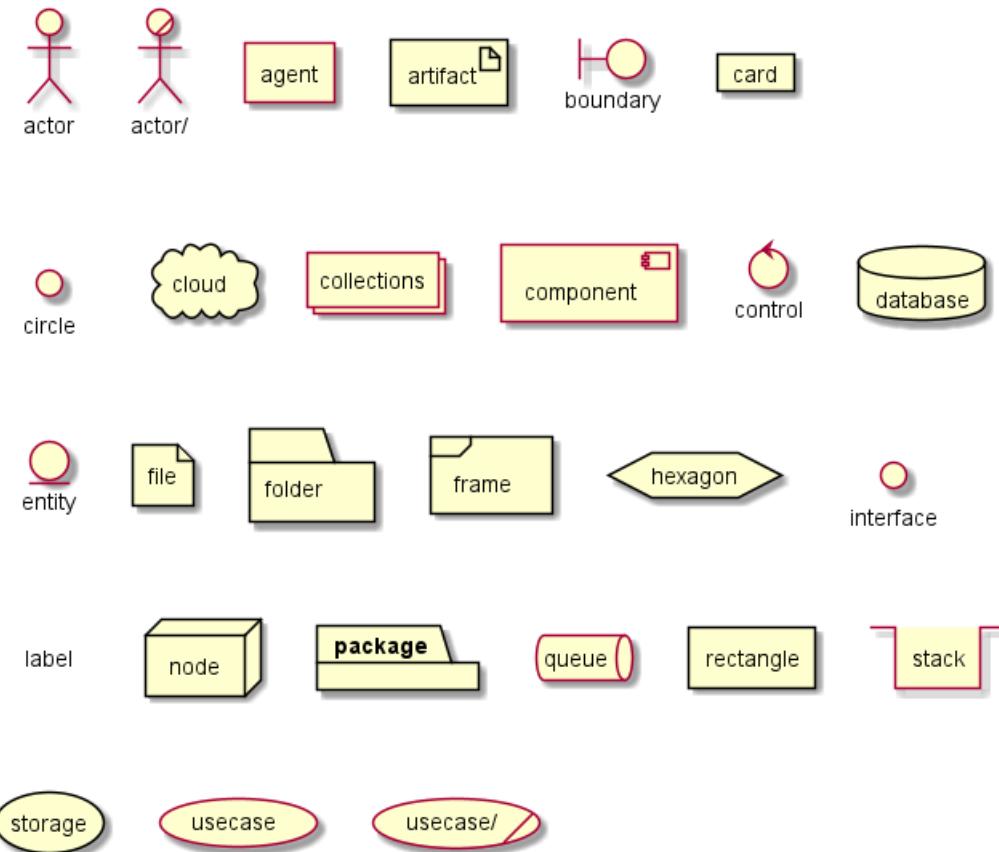


## 8 Deployment Diagram

### 8.1 Declaring element

```
@startuml
actor actor
actor/ "actor/"
agent agent
artifact artifact
boundary boundary
card card
circle circle
cloud cloud
collections collections
component component
control control
database database
entity entity
file file
folder folder
frame frame
hexagon hexagon
interface interface
label label
node node
package package
queue queue
rectangle rectangle
stack stack
storage storage
usecase usecase
usecase/ "usecase/"
@enduml
```





You can optionally put text using bracket [] for a long description.

```
@startuml
folder folder [
This is a <b>folder
-----
You can use separator
=====
of different kind
....
and style
]

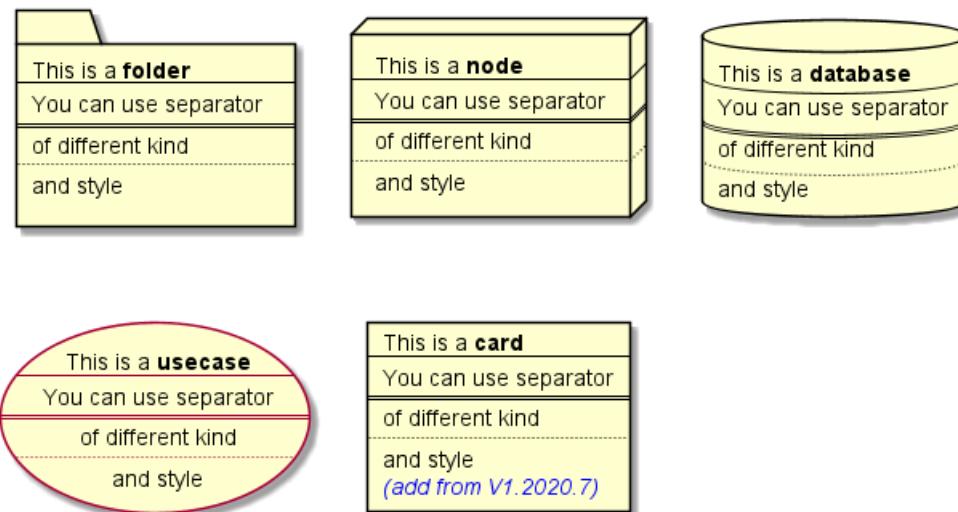
node node [
This is a <b>node
-----
You can use separator
=====
of different kind
....
and style
]

database database [
This is a <b>database
-----
You can use separator
=====
of different kind
....
and style
```



```
]
usecase usecase [
This is a <b>usecase
-----
You can use separator
=====
of different kind
....
and style
]
```

```
card card [
This is a <b>card
-----
You can use separator
=====
of different kind
....
and style
<i><color:blue>(add from V1.2020.7)</color></i>
]
@enduml
```



## 8.2 Declaring element (using short form)

We can declare element using some short forms.

Long form Keyword	Short form Keyword	Long form example	Short form example	Ref.
actor	: a :	actor actor1	:actor2:	Actors
component	[ c ]	component component1	[component2]	Components
interface	() i	interface interface1	() "interface2"	Interfaces
usecase	( u )	usecase usecase1	(usecase2)	Usecases

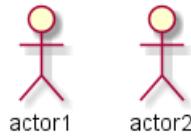
### 8.2.1 Actor

```
@startuml
```

```
actor actor1
:actor2:

@enduml
```





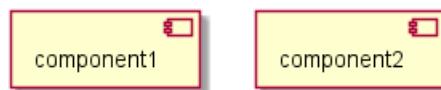
**NB:** There is an old syntax for actor with guillemet which is now deprecated and will be removed some days. Please do not use in your diagram.

### 8.2.2 Component

```
@startuml
```

```
component component1
[component2]
```

```
@enduml
```



### 8.2.3 Interface

```
@startuml
```

```
interface interface1
() "interface2"

label "//interface example//"
@enduml
```



*interface example*

### 8.2.4 Usecase

```
@startuml
```

```
usecase usecase1
(usecase2)
```

```
@enduml
```



## 8.3 Linking or arrow

You can create simple links between elements with or without labels:

```
@startuml
```

```
node node1
node node2
node node3
```

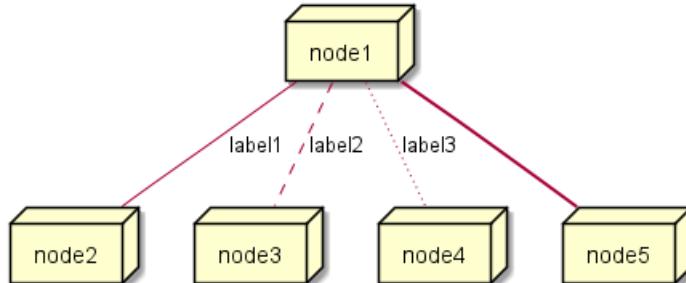


```

node node4
node node5
node1 -- node2 : label1
node1 .. node3 : label2
node1 ~~ node4 : label3
node1 == node5

```

@enduml



It is possible to use several types of links:

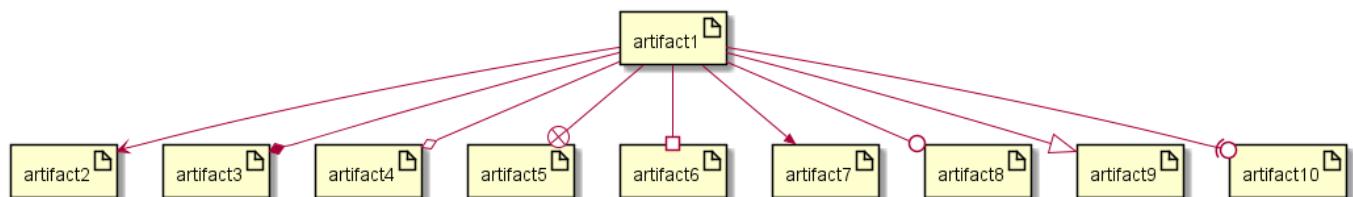
@startuml

```

artifact artifact1
artifact artifact2
artifact artifact3
artifact artifact4
artifact artifact5
artifact artifact6
artifact artifact7
artifact artifact8
artifact artifact9
artifact artifact10
artifact1 --> artifact2
artifact1 --* artifact3
artifact1 --o artifact4
artifact1 ---+ artifact5
artifact1 --# artifact6
artifact1 -->> artifact7
artifact1 --0 artifact8
artifact1 --^ artifact9
artifact1 --(0 artifact10

```

@enduml



You can also have the following types:

@startuml

```

cloud cloud1
cloud cloud2
cloud cloud3

```

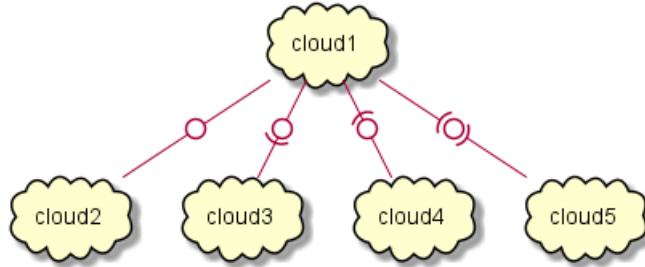


```

cloud cloud4
cloud cloud5
cloud1 -0- cloud2
cloud1 -0)- cloud3
cloud1 -(0- cloud4
cloud1 -(0)- cloud5

```

@enduml



or another example:

```

@startuml
actor foo1
actor foo2
foo1 <-0-> foo2
foo1 <-(0)-> foo2

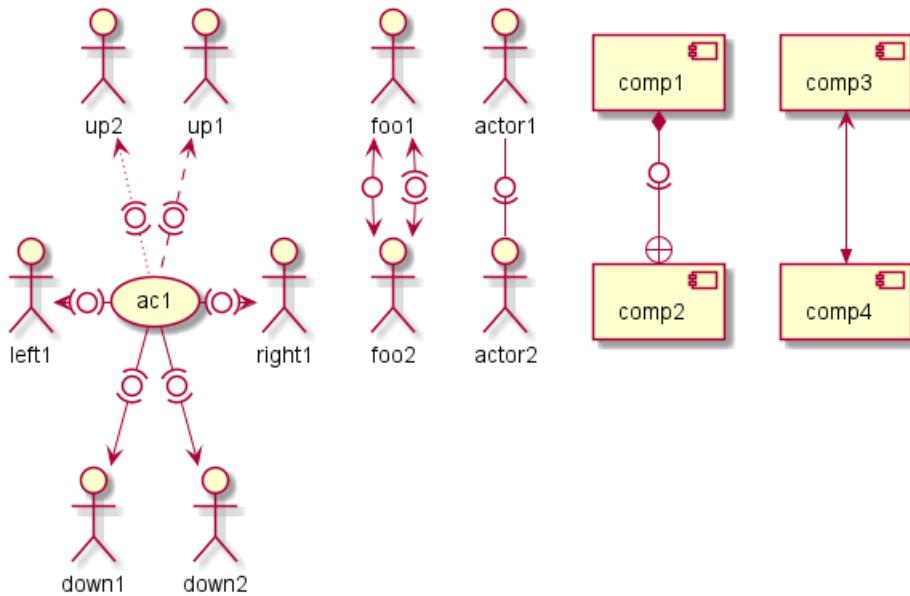
(ac1) -le(0)-> left1
ac1 -ri(0)-> right1
ac1 .up(0).> up1
ac1 ~up(0)~> up2
ac1 -do(0)-> down1
ac1 -do(0)-> down2

actor1 -0)- actor2

component comp1
component comp2
comp1 *-0)--+ comp2
[comp3] <-->> [comp4]
@enduml

```





[Ref. QA-1736]

See all type on **Appendix**.

## 8.4 Bracketed arrow style

*Similar as Bracketed class relations (linking or arrow) style*

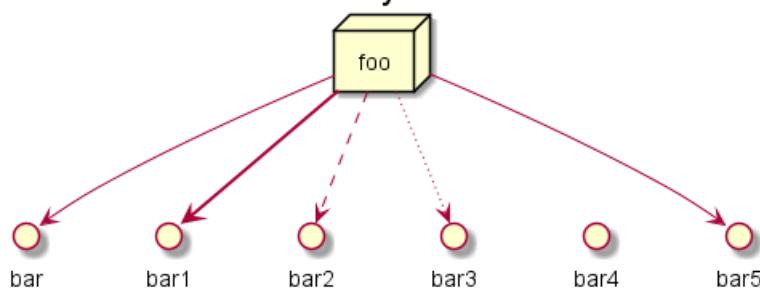
### 8.4.1 Line style

It's also possible to have explicitly bold, dashed, dotted, hidden or plain arrows:

- without label

```
@startuml
node foo
title Bracketed line style without label
foo --> bar
foo -[bold]-> bar1
foo -[dashed]-> bar2
foo -[dotted]-> bar3
foo -[hidden]-> bar4
foo -[plain]-> bar5
@enduml
```

Bracketed line style without label



- with label

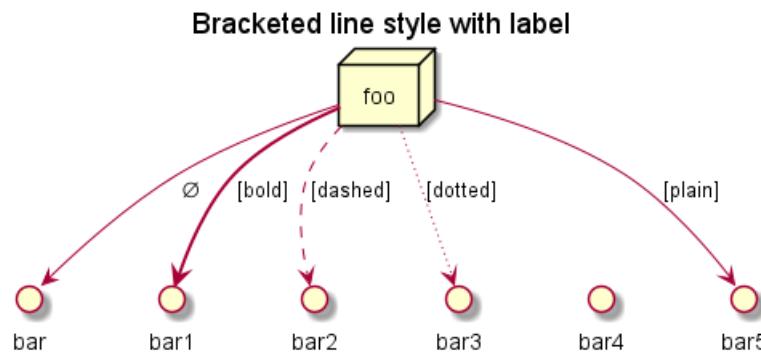
```
@startuml
title Bracketed line style with label
node foo
```



```

foo --> bar      :
foo -[bold]-> bar1 : [bold]
foo -[dashed]-> bar2 : [dashed]
foo -[dotted]-> bar3 : [dotted]
foo -[hidden]-> bar4 : [hidden]
foo -[plain]-> bar5 : [plain]
@enduml

```



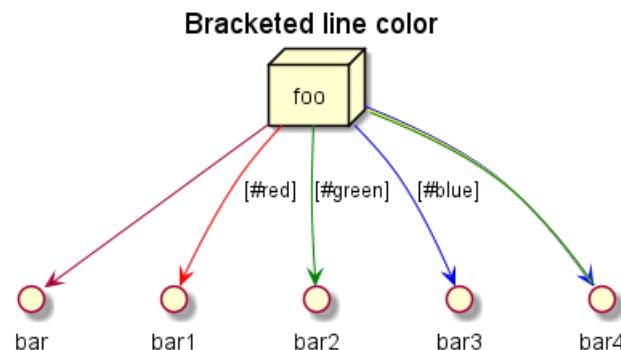
[Adapted from QA-4181]

#### 8.4.2 Line color

```

@startuml
title Bracketed line color
node foo
foo --> bar
foo -[#red]-> bar1 : [#red]
foo -[#green]-> bar2 : [#green]
foo -[#blue]-> bar3 : [#blue]
foo -[#blue;#yellow;#green]-> bar4
@enduml

```



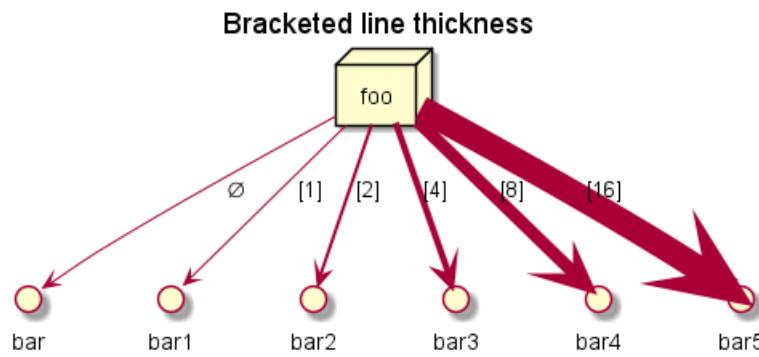
#### 8.4.3 Line thickness

```

@startuml
title Bracketed line thickness
node foo
foo --> bar      :
foo -[thickness=1]-> bar1 : [1]
foo -[thickness=2]-> bar2 : [2]
foo -[thickness=4]-> bar3 : [4]
foo -[thickness=8]-> bar4 : [8]
foo -[thickness=16]-> bar5 : [16]
@enduml

```



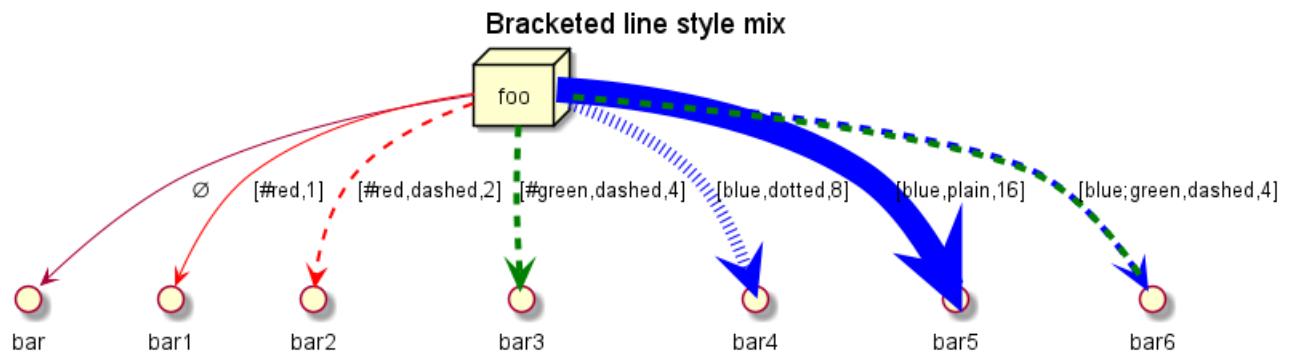


[Adapted from QA-4949]

#### 8.4.4 Mix

```

@startuml
title Bracketed line style mix
node foo
foo --> bar
foo -[#red,thickness=1]-> bar1 : [#red,1]
foo -[#red,dashed,thickness=2]-> bar2 : [#red,dashed,2]
foo -[#green,dashed,thickness=4]-> bar3 : [#green,dashed,4]
foo -[#blue,dotted,thickness=8]-> bar4 : [blue,dotted,8]
foo -[#blue,plain,thickness=16]-> bar5 : [blue,plain,16]
foo -[#blue;#green,dashed,thickness=4]-> bar6 : [blue;green,dashed,4]
@enduml
  
```



## 8.5 Change arrow color and style (inline style)

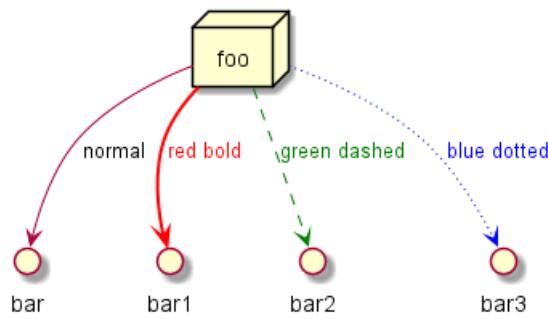
You can change the color or style of individual arrows using the inline following notation:

- `#color;line.[bold|dashed|dotted];text:color`

```

@startuml
node foo
foo --> bar : normal
foo --> bar1 #line:red;line.bold;text:red : red bold
foo --> bar2 #green;line.dashed;text:green : green dashed
foo --> bar3 #blue;line.dotted;text:blue : blue dotted
@enduml
  
```





[Ref. QA-3770 and QA-3816] [See similar feature on class diagram]

## 8.6 Change element color and style (inline style)

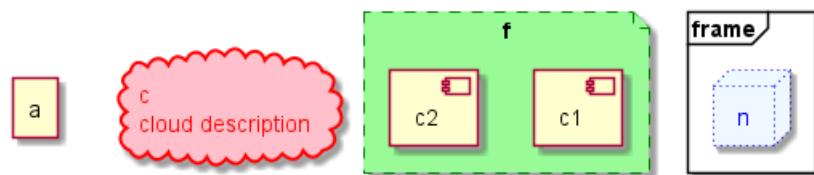
You can change the color or style of individual element using the following notation:

- #**[color|back:color];line:color;line.[bold|dashed|dotted];text:color**

```
@startuml
agent a
cloud c #pink;line:red;line.bold;text:red
file f #palegreen;line:green;line.dashed;text:green
node n #aliceblue;line:blue;line.dotted;text:blue
@enduml
```



```
@startuml
agent a
cloud c #pink;line:red;line.bold;text:red [
c
cloud description
]
file f #palegreen;line:green;line.dashed;text:green {
[c1]
[c2]
}
frame frame {
node n #aliceblue;line:blue;line.dotted;text:blue
}
@enduml
```



[Ref. QA-6852]



## 8.7 Nestable elements

Here are the nestable elements:

```
@startuml
artifact artifact {
}
card card {
}
cloud cloud {
}
component component {
}
database database {
}
file file {
}
folder folder {
}
frame frame {
}
hexagon hexagon {
}
node node {
}
package package {
}
queue queue {
}
rectangle rectangle {
}
stack stack {
}
storage storage {
}
@enduml
```



## 8.8 Packages and nested elements

### 8.8.1 Example with one level

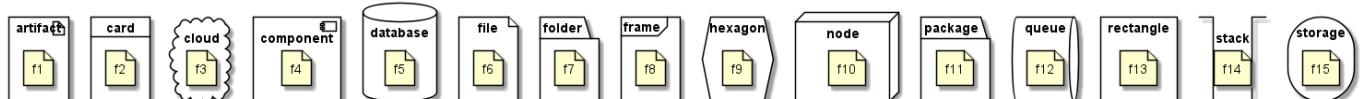
```
@startuml
artifact      artifactVeryL00000000000000000000000g      as "artifact" {
file f1
}
card        cardVeryL00000000000000000000000g      as "card" {
file f2
}
cloud        cloudVeryL00000000000000000000000g      as "cloud" {
file f3
}
component    componentVeryL00000000000000000000000g   as "component" {
file f4
}
database     databaseVeryL00000000000000000000000g    as "database" {
file f5
}
```



```

}
file      fileVeryL00000000000000000000g      as "file" {
file f6
}
folder    folderVeryL00000000000000000000g     as "folder" {
file f7
}
frame     frameVeryL00000000000000000000g      as "frame" {
file f8
}
hexagon   hexagonVeryL00000000000000000000g     as "hexagon" {
file f9
}
node      nodeVeryL00000000000000000000g      as "node" {
file f10
}
package   packageVeryL00000000000000000000g     as "package" {
file f11
}
queue     queueVeryL00000000000000000000g      as "queue" {
file f12
}
rectangle rectangleVeryL00000000000000000000g   as "rectangle" {
file f13
}
stack     stackVeryL00000000000000000000g      as "stack" {
file f14
}
storage   storageVeryL00000000000000000000g     as "storage" {
file f15
}
@enduml

```



### 8.8.2 Other example

```

@startuml
artifact Foo1 {
    folder Foo2
}

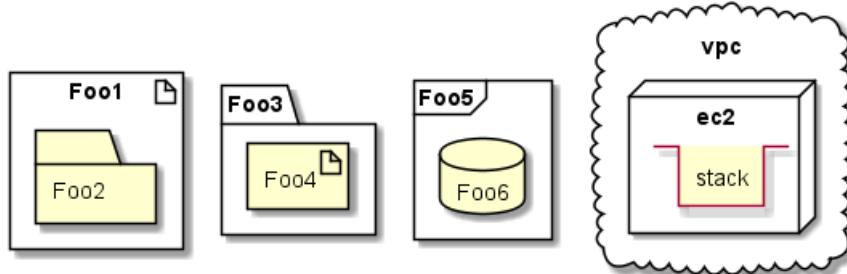
folder Foo3 {
    artifact Foo4
}

frame Foo5 {
    database Foo6
}

cloud vpc {
    node ec2 {
        stack stack
    }
}

```

```
@enduml
```

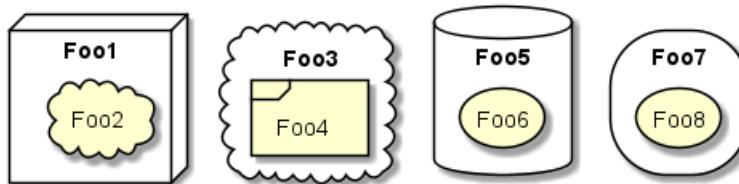


```
@startuml
node Foo1 {
    cloud Foo2
}

cloud Foo3 {
    frame Foo4
}

database Foo5 {
    storage Foo6
}

storage Foo7 {
    storage Foo8
}
@enduml
```



### 8.8.3 Full nesting

Here is all the nested elements:

- by alphabetical order:

```
@startuml
artifact artifact {
card card {
cloud cloud {
component component {
database database {
file file {
folder folder {
frame frame {
hexagon hexagon {
node node {
package package {
queue queue {
rectangle rectangle {
stack stack {
storage storage {
```



```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

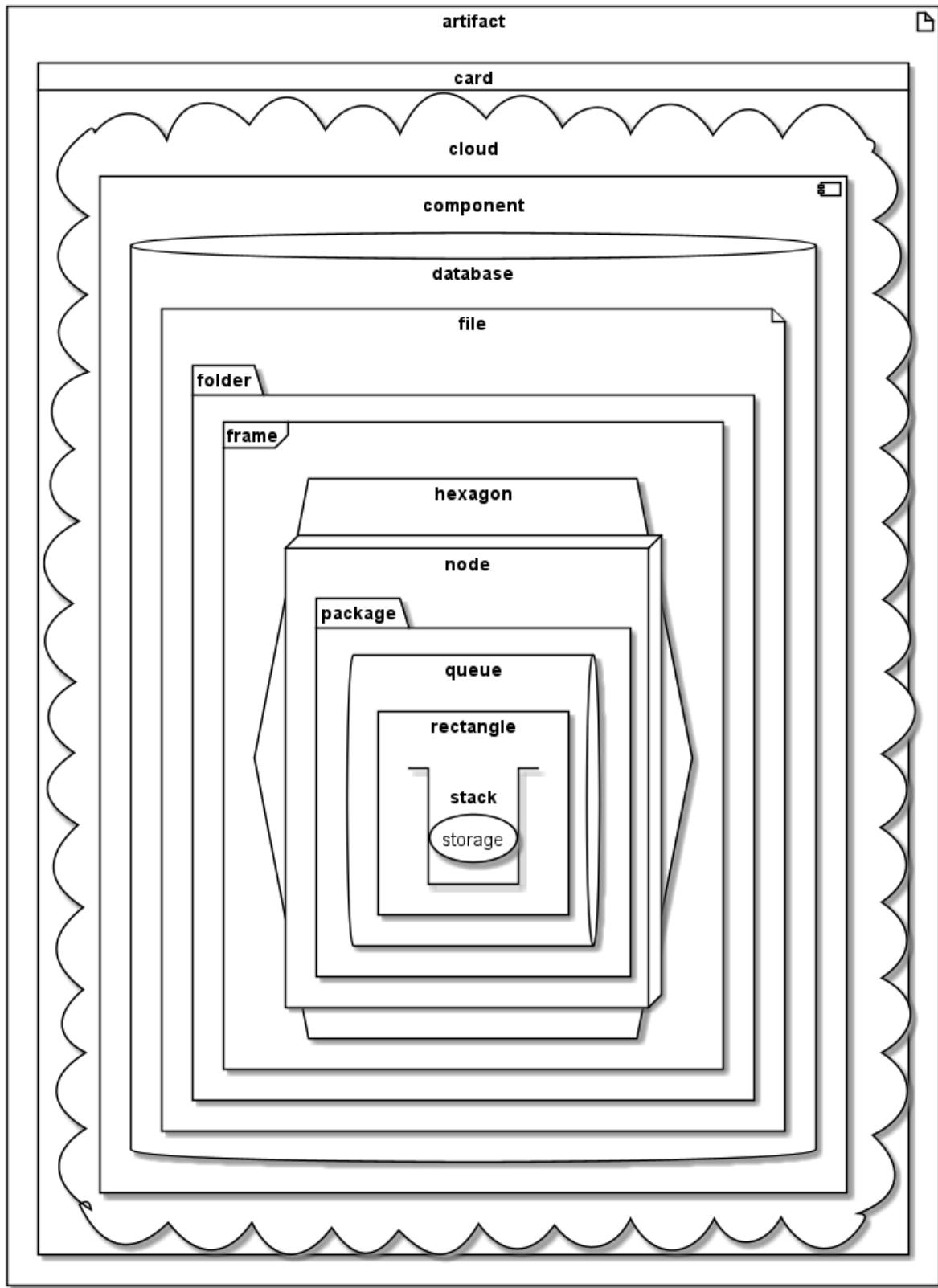
```
}
```

```
}
```

```
}
```

```
@enduml
```



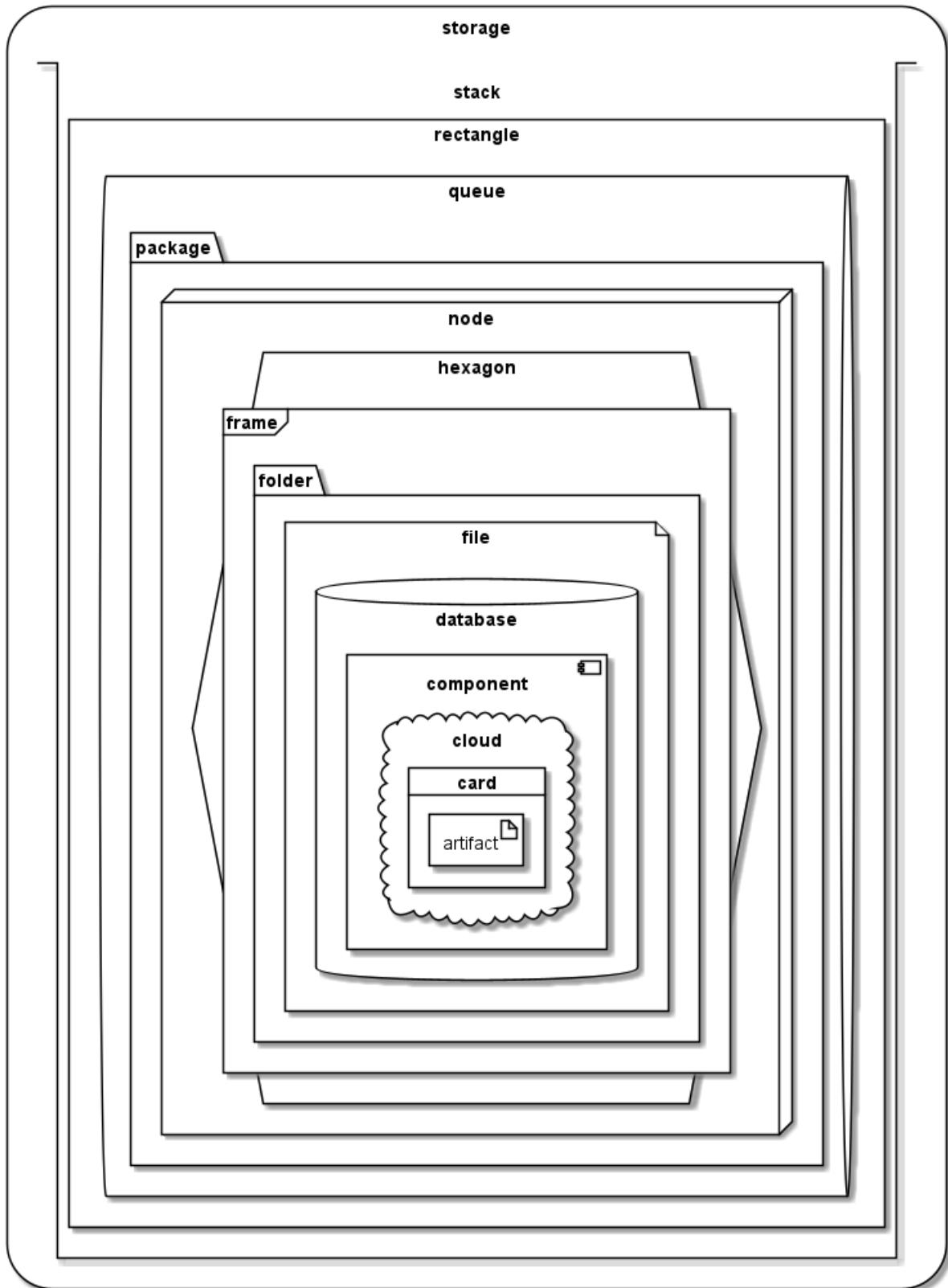


- or reverse alphabetical order

```
@startuml  
storage storage {  
stack stack {  
rectangle rectangle {  
queue queue {
```







## 8.9 Alias

### 8.9.1 Simple alias with as

```
@startuml  
node Node1 as n1  
node "Node 2" as n2
```

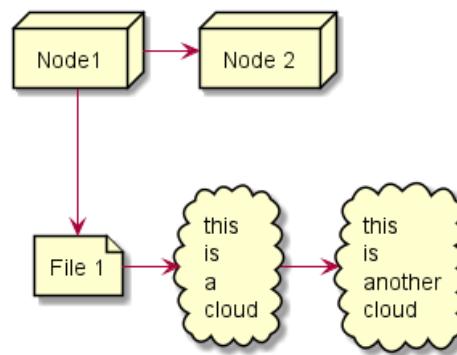


```

file f1 as "File 1"
cloud c1 as "this
is
a
cloud"
cloud c2 [this
is
another
cloud]

n1 -> n2
n1 --> f1
f1 -> c1
c1 -> c2
@enduml

```



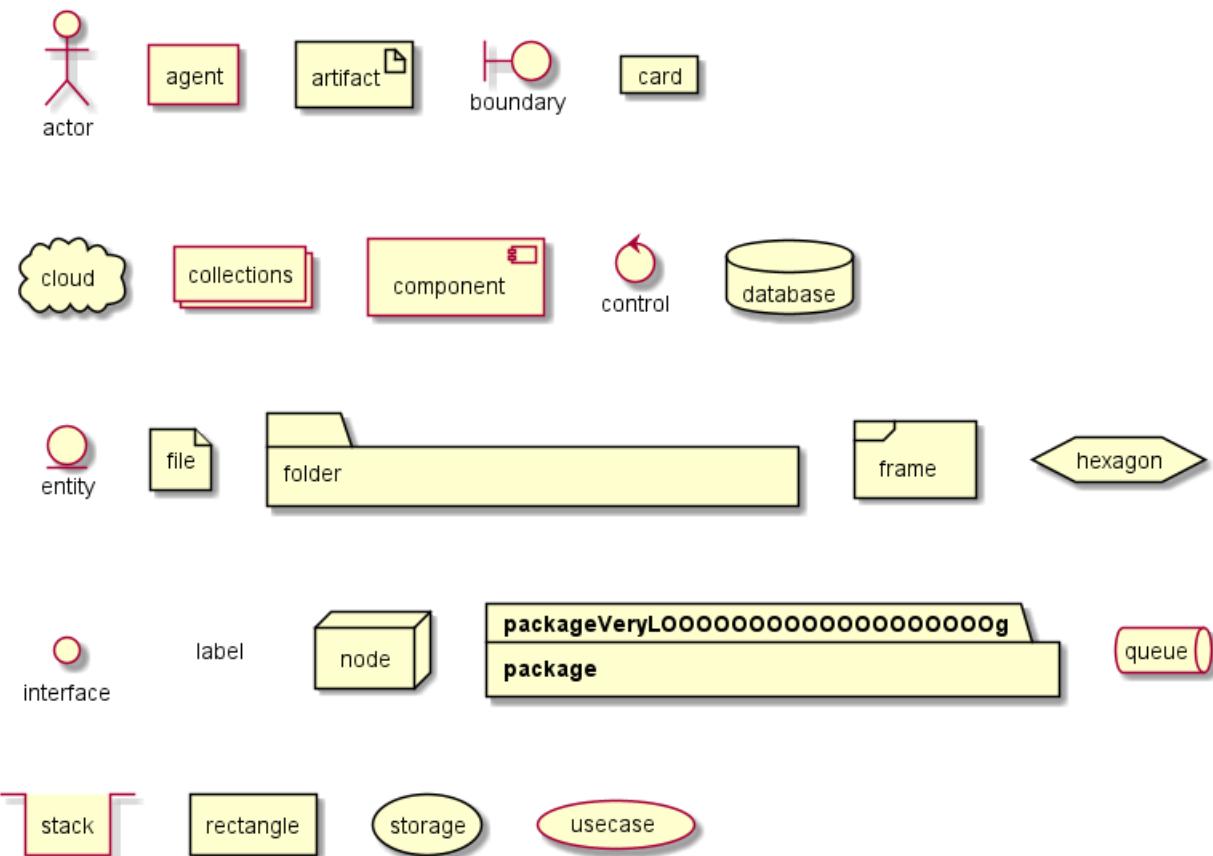
### 8.9.2 Examples of long alias

```

@startuml
actor      "actor"      as actorVeryLooooooooooooooooooooo0g
agent      "agent"       as agentVeryLooooooooooooooooooooo0g
artifact   "artifact"    as artifactVeryLooooooooooooooooooooo0g
boundary   "boundary"   as boundaryVeryLooooooooooooooooooooo0g
card       "card"        as cardVeryLooooooooooooooooooooo0g
cloud      "cloud"       as cloudVeryLooooooooooooooooooooo0g
collections "collections" as collectionsVeryLoooooooooooooo0g
component   "component"  as componentVeryLoooooooooooooo0g
control     "control"    as controlVeryLoooooooooooooo0g
database   "database"   as databaseVeryLoooooooooooooo0g
entity      "entity"     as entityVeryLoooooooooooooo0g
file        "file"        as fileVeryLoooooooooooooo0g
folder      "folder"     as folderVeryLoooooooooooooo0g
frame       "frame"      as frameVeryLoooooooooooooo0g
hexagon    "hexagon"    as hexagonVeryLoooooooooooooo0g
interface   "interface"  as interfaceVeryLoooooooooooooo0g
label       "label"       as labelVeryLoooooooooooooo0g
node        "node"        as nodeVeryLoooooooooooooo0g
package     "package"    as packageVeryLoooooooooooooo0g
queue       "queue"      as queueVeryLoooooooooooooo0g
stack       "stack"      as stackVeryLoooooooooooooo0g
rectangle   "rectangle"  as rectangleVeryLoooooooooooooo0g
storage     "storage"    as storageVeryLoooooooooooooo0g
usecase     "usecase"    as usecaseVeryLoooooooooooooo0g
@enduml

```



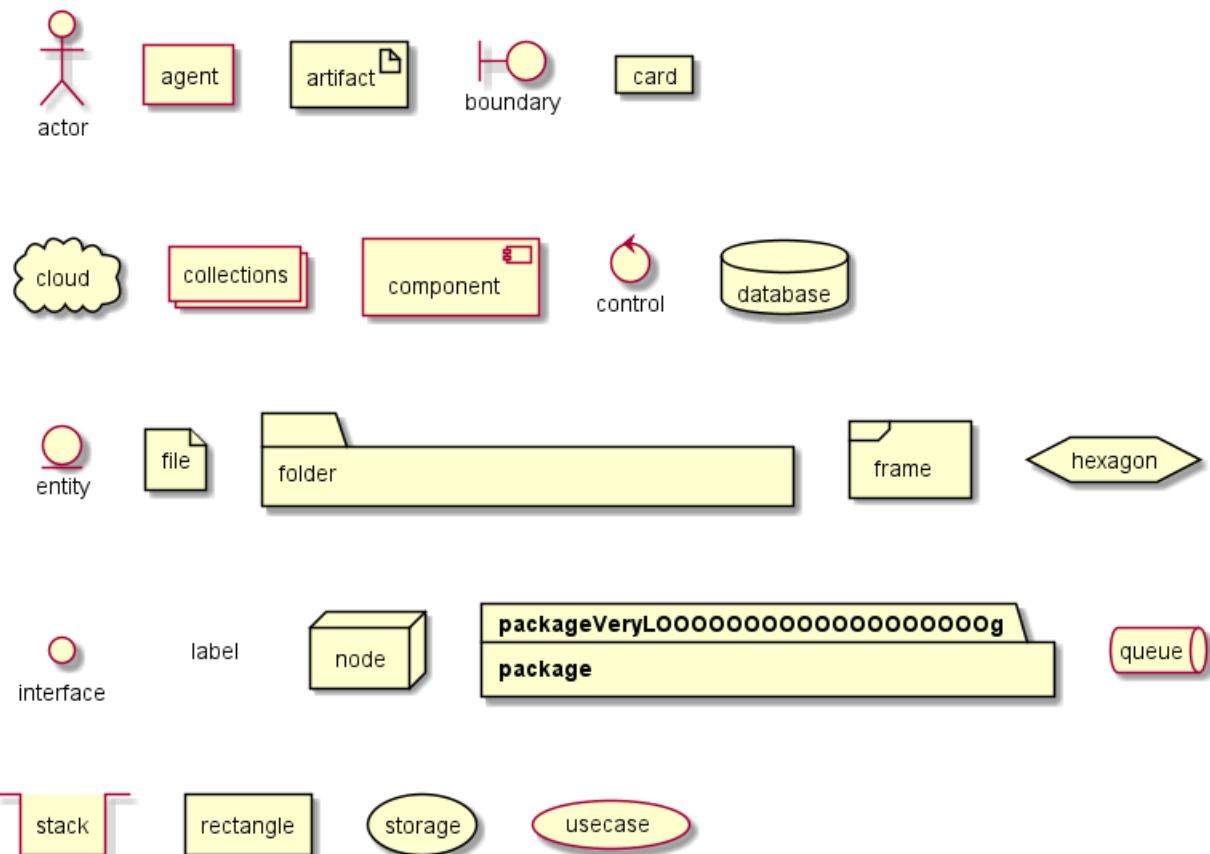


```

@startuml
actor      actorVeryLooooooooooooo0g          as "actor"
agent      agentVeryLooooooooooooo0g           as "agent"
artifact    artifactVeryLooooooooooooo0g        as "artifact"
boundary   boundaryVeryLooooooooooooo0g       as "boundary"
card       cardVeryLooooooooooooo0g            as "card"
cloud      cloudVeryLooooooooooooo0g           as "cloud"
collections collectionsVeryLooooooooooooo0g   as "collections"
component   componentVeryLooooooooooooo0g      as "component"
control     controlVeryLooooooooooooo0g         as "control"
database   databaseVeryLooooooooooooo0g        as "database"
entity      entityVeryLooooooooooooo0g          as "entity"
file        fileVeryLooooooooooooo0g            as "file"
folder      folderVeryLooooooooooooo0g          as "folder"
frame       frameVeryLooooooooooooo0g           as "frame"
hexagon     hexagonVeryLooooooooooooo0g         as "hexagon"
interface   interfaceVeryLooooooooooooo0g       as "interface"
label       labelVeryLooooooooooooo0g           as "label"
node        nodeVeryLooooooooooooo0g            as "node"
package     packageVeryLooooooooooooo0g          as "package"
queue       queueVeryLooooooooooooo0g           as "queue"
stack       stackVeryLooooooooooooo0g            as "stack"
rectangle   rectangleVeryLooooooooooooo0g        as "rectangle"
storage     storageVeryLooooooooooooo0g          as "storage"
usecase    usecaseVeryLooooooooooooo0g           as "usecase"
@enduml

```



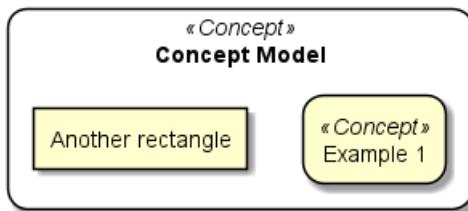


[Ref. QA-12082]

## 8.10 Round corner

```
@startuml
skinparam rectangle {
    roundCorner<<Concept>> 25
}

rectangle "Concept Model" <<Concept>> {
    rectangle "Example 1" <<Concept>> as ex1
    rectangle "Another rectangle"
}
@enduml
```



## 8.11 Specific SkinParameter

### 8.11.1 roundCorner

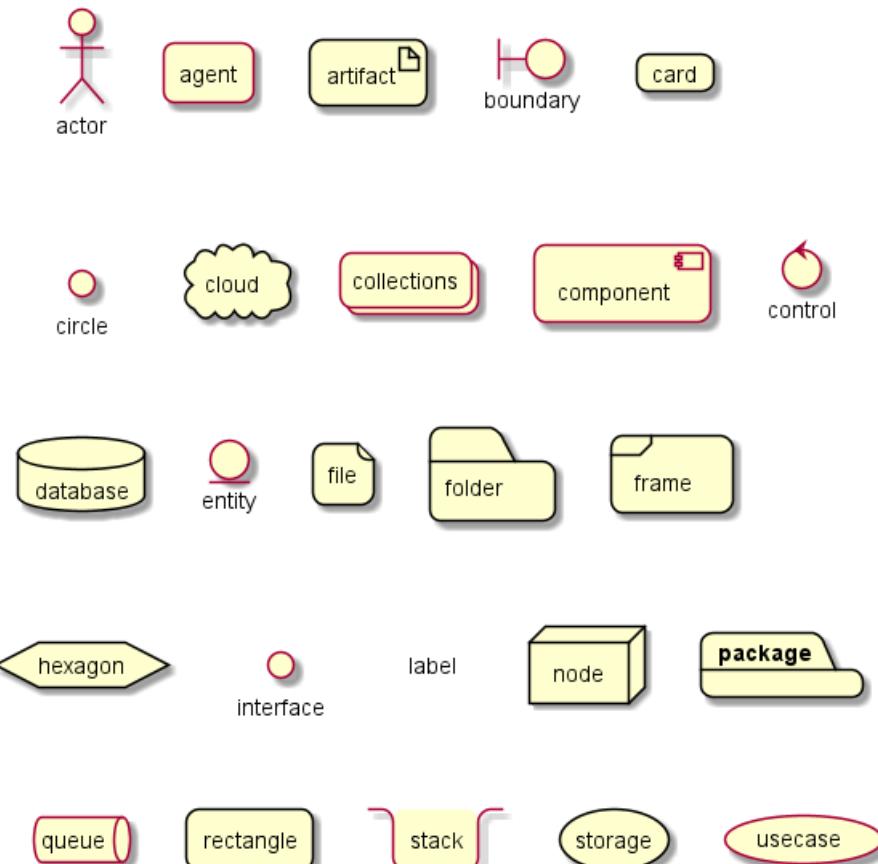
```
@startuml
skinparam roundCorner 15
actor actor
agent agent
```



```

artifact artifact
boundary boundary
card card
circle circle
cloud cloud
collections collections
component component
control control
database database
entity entity
file file
folder folder
frame frame
hexagon hexagon
interface interface
label label
node node
package package
queue queue
rectangle rectangle
stack stack
storage storage
usecase usecase
@enduml

```



[Ref. QA-5299, QA-6915, QA-11943]

## 8.12 Appendix: All type of arrow line

```

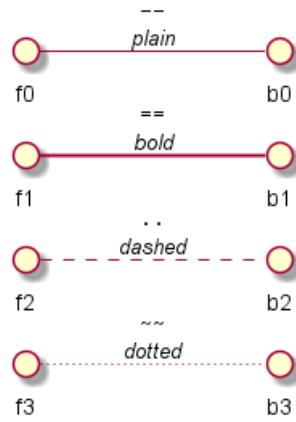
@startuml
left to right direction

```



```
skinparam nodesep 5
```

```
f3 ~~ b3 : ""~~"\n//dotted//  
f2 .. b2 : "".."\n//dashed//  
f1 == b1 : ""=="\n//bold//  
f0 -- b0 : ""--"\n//plain//  
@enduml
```



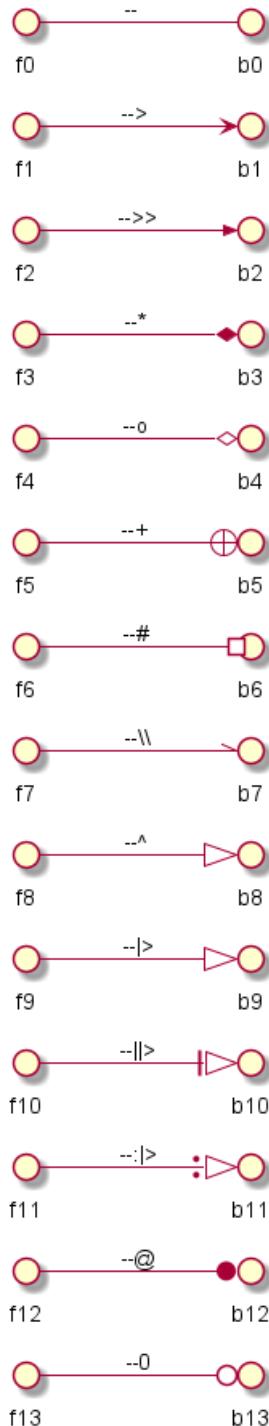
## 8.13 Appendix: All type of arrow head or '0' arrow

### 8.13.1 Type of arrow head

```
@startuml  
left to right direction  
skinparam nodesep 5
```

```
f13 --0 b13 : ""--0""  
f12 --@ b12 : ""--@""  
f11 --:> b11 : ""--:>""  
f10 --||> b10 : ""--||>""  
f9 --|> b9 : ""--|>""  
f8 --^ b8 : ""--^ ""  
f7 --\\ b7 : ""--\\\\\\ ""  
f6 --# b6 : ""--# ""  
f5 --+ b5 : ""--+ ""  
f4 --o b4 : ""--o ""  
f3 --* b3 : ""--* ""  
f2 -->> b2 : ""-->>""  
f1 --> b1 : ""--> ""  
f0 -- b0 : ""-- ""  
@enduml
```





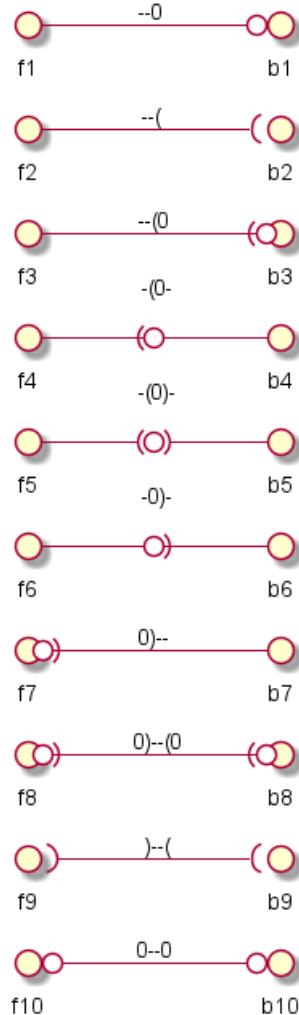
### 8.13.2 Type of '0' arrow or circle arrow

```
@startuml
left to right direction
skinparam nodesep 5

f10 0--0 b10 : "" 0--0 ""
f9 )--( b9 : "" )--(""
f8 0)--(0 b8 : "" 0)--(0"""
f7 0)-- b7 : "" 0)-- ""
f6 -0)- b6 : "" -0)-\n """
f5 -(0)- b5 : "" -(0)-\n""
```



```
f4 -(0- b4 : "" -(0-\n """
f3 --(0 b3 : "" --(0 """
f2 --( b2 : "" --(  """
f1 --0 b1 : "" --0  """
@enduml
```



## 8.14 Appendix: Test of inline style on all element

### 8.14.1 Simple element

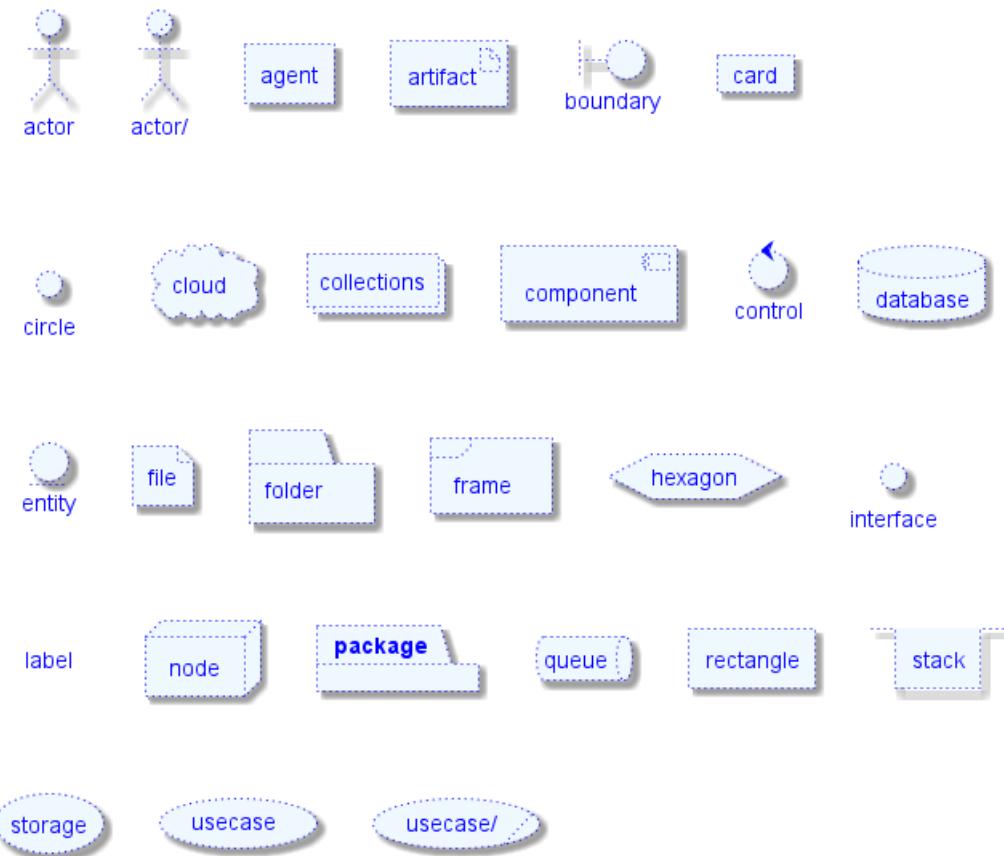
```
@startuml
actor actor #aliceblue;line:blue;line.dotted;text:blue
actor/ "actor/" #aliceblue;line:blue;line.dotted;text:blue
agent agent #aliceblue;line:blue;line.dotted;text:blue
artifact artifact #aliceblue;line:blue;line.dotted;text:blue
boundary boundary #aliceblue;line:blue;line.dotted;text:blue
card card #aliceblue;line:blue;line.dotted;text:blue
circle circle #aliceblue;line:blue;line.dotted;text:blue
cloud cloud #aliceblue;line:blue;line.dotted;text:blue
collections collections #aliceblue;line:blue;line.dotted;text:blue
component component #aliceblue;line:blue;line.dotted;text:blue
control control #aliceblue;line:blue;line.dotted;text:blue
database database #aliceblue;line:blue;line.dotted;text:blue
entity entity #aliceblue;line:blue;line.dotted;text:blue
file file #aliceblue;line:blue;line.dotted;text:blue
folder folder #aliceblue;line:blue;line.dotted;text:blue
```



```

frame frame #aliceblue;line:blue;line.dotted;text:blue
hexagon hexagon #aliceblue;line:blue;line.dotted;text:blue
interface interface #aliceblue;line:blue;line.dotted;text:blue
label label #aliceblue;line:blue;line.dotted;text:blue
node node #aliceblue;line:blue;line.dotted;text:blue
package package #aliceblue;line:blue;line.dotted;text:blue
queue queue #aliceblue;line:blue;line.dotted;text:blue
rectangle rectangle #aliceblue;line:blue;line.dotted;text:blue
stack stack #aliceblue;line:blue;line.dotted;text:blue
storage storage #aliceblue;line:blue;line.dotted;text:blue
usecase usecase #aliceblue;line:blue;line.dotted;text:blue
usecase/ "usecase/" #aliceblue;line:blue;line.dotted;text:blue
@enduml

```



#### 8.14.2 Nested element

#### 8.14.3 Without sub-element

```

@startuml
artifact artifact #aliceblue;line:blue;line.dotted;text:blue {
}
card card #aliceblue;line:blue;line.dotted;text:blue {
}
cloud cloud #aliceblue;line:blue;line.dotted;text:blue {
}
component component #aliceblue;line:blue;line.dotted;text:blue {
}
database database #aliceblue;line:blue;line.dotted;text:blue {
}
file file #aliceblue;line:blue;line.dotted;text:blue {
}

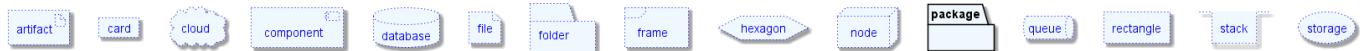
```



```

folder folder #aliceblue;line:blue;line.dotted;text:blue {
}
frame frame #aliceblue;line:blue;line.dotted;text:blue {
}
hexagon hexagon #aliceblue;line:blue;line.dotted;text:blue {
}
node node #aliceblue;line:blue;line.dotted;text:blue {
}
package package #aliceblue;line:blue;line.dotted;text:blue {
}
queue queue #aliceblue;line:blue;line.dotted;text:blue {
}
rectangle rectangle #aliceblue;line:blue;line.dotted;text:blue {
}
stack stack #aliceblue;line:blue;line.dotted;text:blue {
}
storage storage #aliceblue;line:blue;line.dotted;text:blue {
}
@enduml

```



#### 8.14.4 With sub-element

```

@startuml
artifact      artifactVeryL00000000000000000000g      as "artifact" #aliceblue;line:blue;line.dotted;text:blue
file f1
}
card         cardVeryL00000000000000000000g      as "card" #aliceblue;line:blue;line.dotted;text:blue
file f2
}
cloud        cloudVeryL00000000000000000000g      as "cloud" #aliceblue;line:blue;line.dotted;text:blue
file f3
}
component    componentVeryL00000000000000000000g   as "component" #aliceblue;line:blue;line.dotted;text:blue
file f4
}
database     databaseVeryL00000000000000000000g    as "database" #aliceblue;line:blue;line.dotted;text:blue
file f5
}
file         fileVeryL00000000000000000000g      as "file" #aliceblue;line:blue;line.dotted;text:blue
file f6
}
folder        folderVeryL00000000000000000000g     as "folder" #aliceblue;line:blue;line.dotted;text:blue
file f7
}
frame         frameVeryL00000000000000000000g      as "frame" #aliceblue;line:blue;line.dotted;text:blue
file f8
}
hexagon       hexagonVeryL00000000000000000000g     as "hexagon" #aliceblue;line:blue;line.dotted;text:blue
file f9
}
node          nodeVeryL00000000000000000000g      as "node" #aliceblue;line:blue;line.dotted;text:blue
file f10
}
package       packageVeryL00000000000000000000g     as "package" #aliceblue;line:blue;line.dotted;text:blue
file f11
}

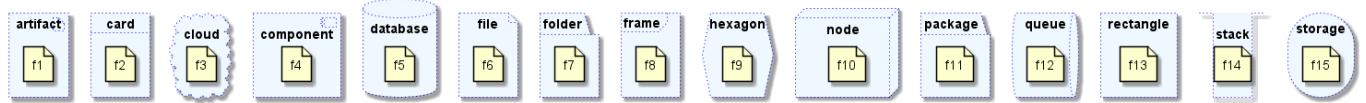
```



```

}
queue      queueVeryL0000000000000000000g      as "queue" #aliceblue;line:blue;line.dotted;text:bl
file f12
}
rectangle  rectangleVeryL0000000000000000000g   as "rectangle" #aliceblue;line:blue;line.dotted;text:
file f13
}
stack      stackVeryL0000000000000000000g      as "stack" #aliceblue;line:blue;line.dotted;text:bl
file f14
}
storage    storageVeryL0000000000000000000g     as "storage" #aliceblue;line:blue;line.dotted;text:bl
file f15
}
@enduml

```



## 8.15 Appendix: Test of style on all element

### 8.15.1 Simple element

### 8.15.2 Global style (on componentDiagram)

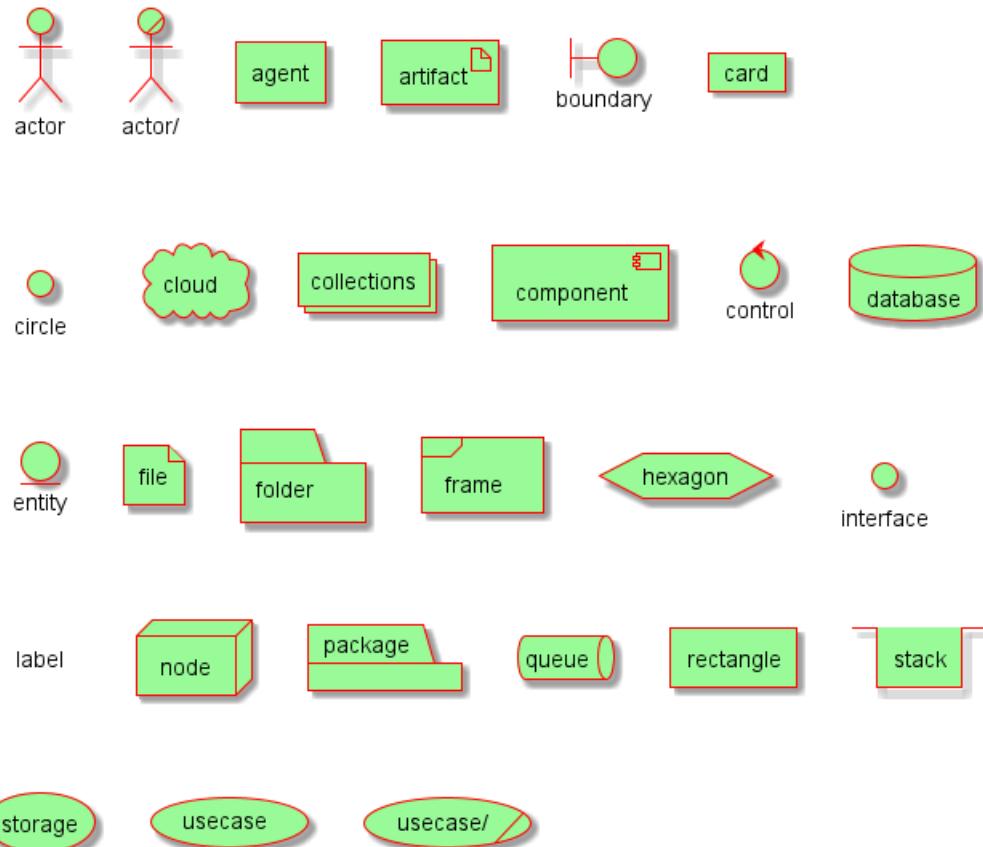
```

@startuml
<style>
componentDiagram {
    BackGroundColor palegreen
    LineThickness 1
    LineColor red
}
</style>
actor actor
actor/ "actor/"
agent agent
artifact artifact
boundary boundary
card card
circle circle
cloud cloud
collections collections
component component
control control
database database
entity entity
file file
folder folder
frame frame
hexagon hexagon
interface interface
label label
node node
package package
queue queue
rectangle rectangle
stack stack
storage storage

```



```
usecase usecase
usecase/ "usecase/"
@enduml
```



### 8.15.3 Style for each element

```
@startuml
<style>
actor {
    BackGroundColor #f80c12
    LineThickness 1
    LineColor black
}
agent {
    BackGroundColor #f80c12
    LineThickness 1
    LineColor black
}
artifact {
    BackGroundColor #ee1100
    LineThickness 1
    LineColor black
}
boundary {
    BackGroundColor #ee1100
    LineThickness 1
    LineColor black
}
card {
    BackGroundColor #ff3311
    LineThickness 1
```



```
LineColor black
}
circle {
    BackGroundColor #ff3311
    LineThickness 1
    LineColor black
}
cloud {
    BackGroundColor #ff4422
    LineThickness 1
    LineColor black
}
collections {
    BackGroundColor #ff4422
    LineThickness 1
    LineColor black
}
component {
    BackGroundColor #ff6644
    LineThickness 1
    LineColor black
}
control {
    BackGroundColor #ff6644
    LineThickness 1
    LineColor black
}
database {
    BackGroundColor #ff9933
    LineThickness 1
    LineColor black
}
entity {
    BackGroundColor #feae2d
    LineThickness 1
    LineColor black
}
file {
    BackGroundColor #feae2d
    LineThickness 1
    LineColor black
}
folder {
    BackGroundColor #ccbb33
    LineThickness 1
    LineColor black
}
frame {
    BackGroundColor #d0c310
    LineThickness 1
    LineColor black
}
hexagon {
    BackGroundColor #aacc22
    LineThickness 1
    LineColor black
}
interface {
```



```

BackGroundColor #69d025
LineThickness 1
LineColor black
}
label {
    BackGroundColor black
    LineThickness 1
    LineColor black
}
node {
    BackGroundColor #22ccaa
    LineThickness 1
    LineColor black
}
package {
    BackGroundColor #12bdb9
    LineThickness 1
    LineColor black
}
queue {
    BackGroundColor #11aabb
    LineThickness 1
    LineColor black
}
rectangle {
    BackGroundColor #4444dd
    LineThickness 1
    LineColor black
}
stack {
    BackGroundColor #3311bb
    LineThickness 1
    LineColor black
}
storage {
    BackGroundColor #3b0cbd
    LineThickness 1
    LineColor black
}
usecase {
    BackGroundColor #442299
    LineThickness 1
    LineColor black
}
</style>
actor actor
actor/ "actor/"
agent agent
artifact artifact
boundary boundary
card card
circle circle
cloud cloud
collections collections
component component
control control
database database
entity entity

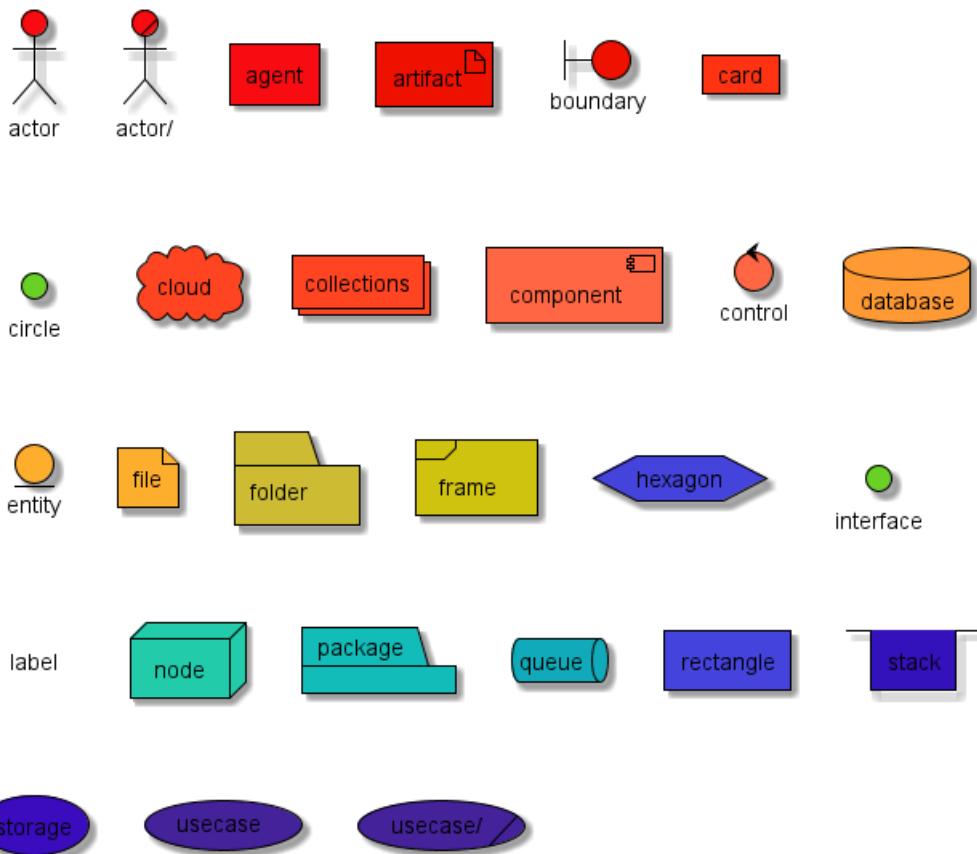
```



```

file file
folder folder
frame frame
hexagon hexagon
interface interface
label label
node node
package package
queue queue
rectangle rectangle
stack stack
storage storage
usecase usecase
usecase/ "usecase/"
@enduml

```



#### 8.15.4 Nested element (without level)

#### 8.15.5 Global style (on componentDiagram)

```

@startuml
<style>
componentDiagram {
    BackGroundColor palegreen
    LineThickness 2
    LineColor red
}
</style>
artifact artifact {
}

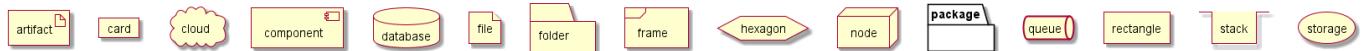
```

```

card card {
}
cloud cloud {
}
component component {
}
database database {
}
file file {
}
folder folder {
}
frame frame {
}
hexagon hexagon {
}
node node {
}
package package {
}
queue queue {
}
rectangle rectangle {
}
stack stack {
}
storage storage {
}

```

@enduml



### 8.15.6 Style for each nested element

```

@startuml
<style>
artifact {
    BackGroundColor #ee1100
    LineThickness 1
    LineColor black
}
card {
    BackGroundColor #ff3311
    LineThickness 1
    LineColor black
}
cloud {
    BackGroundColor #ff4422
    LineThickness 1
    LineColor black
}
component {
    BackGroundColor #ff6644
    LineThickness 1
    LineColor black
}
database {

```



```

BackGroundColor #ff9933
LineThickness 1
LineColor black
}
file {
    BackGroundColor #feae2d
    LineThickness 1
    LineColor black
}
folder {
    BackGroundColor #ccbb33
    LineThickness 1
    LineColor black
}
frame {
    BackGroundColor #d0c310
    LineThickness 1
    LineColor black
}
hexagon {
    BackGroundColor #aacc22
    LineThickness 1
    LineColor black
}
node {
    BackGroundColor #22ccaa
    LineThickness 1
    LineColor black
}
package {
    BackGroundColor #12bdb9
    LineThickness 1
    LineColor black
}
queue {
    BackGroundColor #11aabb
    LineThickness 1
    LineColor black
}
rectangle {
    BackGroundColor #4444dd
    LineThickness 1
    LineColor black
}
stack {
    BackGroundColor #3311bb
    LineThickness 1
    LineColor black
}
storage {
    BackGroundColor #3b0cbd
    LineThickness 1
    LineColor black
}

</style>
artifact artifact {
}

```



```

card card {
}
cloud cloud {
}
component component {
}
database database {
}
file file {
}
folder folder {
}
frame frame {
}
hexagon hexagon {
}
node node {
}
package package {
}
queue queue {
}
rectangle rectangle {
}
stack stack {
}
storage storage {
}

```

@enduml



### 8.15.7 Nested element (with one level)

### 8.15.8 Global style (on componentDiagram)

```

@startuml
<style>
componentDiagram {
    BackGroundColor palegreen
    LineThickness 1
    LineColor red
}
</style>
artifact e1 as "artifact" {
    file f1
}
card e2 as "card" {
    file f2
}
cloud e3 as "cloud" {
    file f3
}
component e4 as "component" {
    file f4
}
database e5 as "database" {
    file f5
}

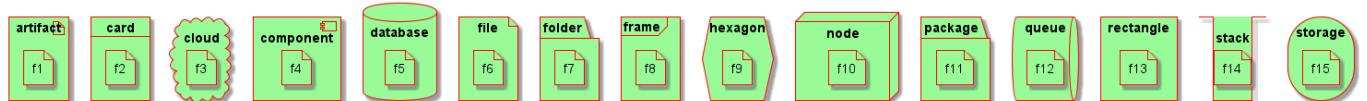
```



```

}
file e6 as "file" {
file f6
}
folder e7 as "folder" {
file f7
}
frame e8 as "frame" {
file f8
}
hexagon e9 as "hexagon" {
file f9
}
node e10 as "node" {
file f10
}
package e11 as "package" {
file f11
}
queue e12 as "queue" {
file f12
}
rectangle e13 as "rectangle" {
file f13
}
stack e14 as "stack" {
file f14
}
storage e15 as "storage" {
file f15
}
}
@enduml

```



### 8.15.9 Style for each nested element

```

@startuml
<style>
artifact {
    BackGroundColor #ee1100
    LineThickness 1
    LineColor black
}
card {
    BackGroundColor #ff3311
    LineThickness 1
    LineColor black
}
cloud {
    BackGroundColor #ff4422
    LineThickness 1
    LineColor black
}
component {

```



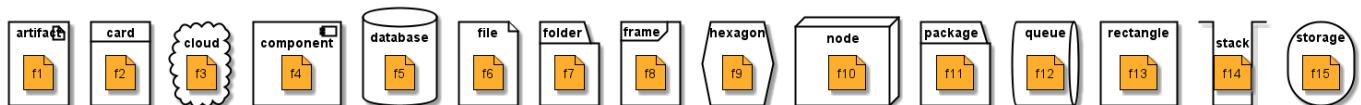
```
BackGroundColor #ff6644
LineThickness 1
LineColor black
}
database {
    BackGroundColor #ff9933
    LineThickness 1
    LineColor black
}
file {
    BackGroundColor #feae2d
    LineThickness 1
    LineColor black
}
folder {
    BackGroundColor #ccbb33
    LineThickness 1
    LineColor black
}
frame {
    BackGroundColor #d0c310
    LineThickness 1
    LineColor black
}
hexagon {
    BackGroundColor #aacc22
    LineThickness 1
    LineColor black
}
node {
    BackGroundColor #22ccaa
    LineThickness 1
    LineColor black
}
package {
    BackGroundColor #12bdb9
    LineThickness 1
    LineColor black
}
queue {
    BackGroundColor #11aabb
    LineThickness 1
    LineColor black
}
rectangle {
    BackGroundColor #4444dd
    LineThickness 1
    LineColor black
}
stack {
    BackGroundColor #3311bb
    LineThickness 1
    LineColor black
}
storage {
    BackGroundColor #3b0cbd
    LineThickness 1
    LineColor black
}
```



```

}
</style>
artifact e1 as "artifact" {
file f1
}
card e2 as "card" {
file f2
}
cloud e3 as "cloud" {
file f3
}
component e4 as "component" {
file f4
}
database e5 as "database" {
file f5
}
file e6 as "file" {
file f6
}
folder e7 as "folder" {
file f7
}
frame e8 as "frame" {
file f8
}
hexagon e9 as "hexagon" {
file f9
}
node e10 as "node" {
file f10
}
package e11 as "package" {
file f11
}
queue e12 as "queue" {
file f12
}
rectangle e13 as "rectangle" {
file f13
}
stack e14 as "stack" {
file f14
}
storage e15 as "storage" {
file f15
}
@enduml

```



## 9 State Diagram

State diagrams are used to give an abstract description of the behavior of a system. This behavior is represented as a series of events that can occur in one or more possible states.

### 9.1 Simple State

You can use [\*] for the starting point and ending point of the state diagram.

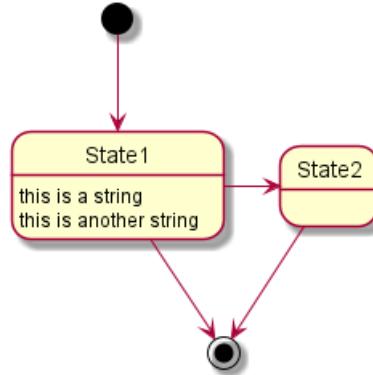
Use --> for arrows.

```
@startuml
```

```
[*] --> State1
State1 --> [*]
State1 : this is a string
State1 : this is another string

State1 -> State2
State2 --> [*]
```

```
@enduml
```



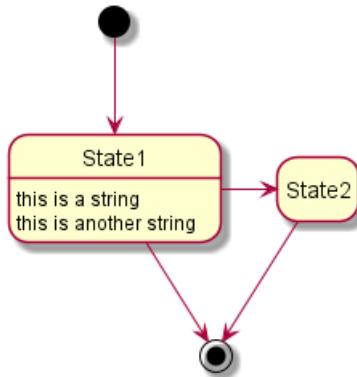
### 9.2 Change state rendering

You can use `hide empty description` to render state as simple box.

```
@startuml
hide empty description
[*] --> State1
State1 --> [*]
State1 : this is a string
State1 : this is another string

State1 -> State2
State2 --> [*]
@enduml
```





## 9.3 Composite state

A state can also be composite. You have to define it using the `state` keywords and brackets.

### 9.3.1 Internal sub-state

```
@startuml
scale 350 width
[*] --> NotShooting

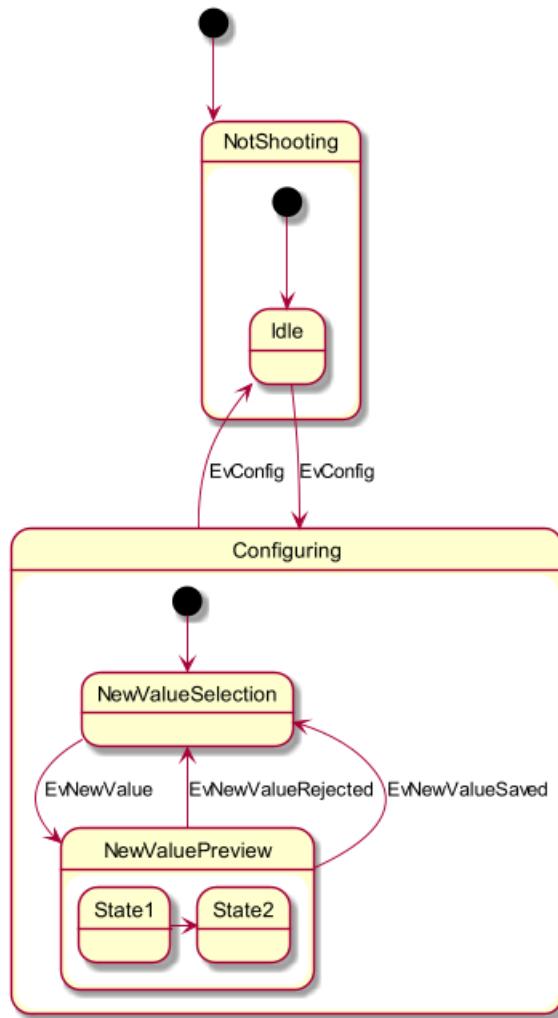
state NotShooting {
    [*] --> Idle
    Idle --> Configuring : EvConfig
    Configuring --> Idle : EvConfig
}

state Configuring {
    [*] --> NewValueSelection
    NewValueSelection --> NewValuePreview : EvnewValue
    NewValuePreview --> NewValueSelection : EvnewValueRejected
    NewValuePreview --> NewValueSelection : EvnewValueSaved

    state NewValuePreview {
        State1 -> State2
    }
}

@enduml
```





### 9.3.2 Sub-state to sub-state

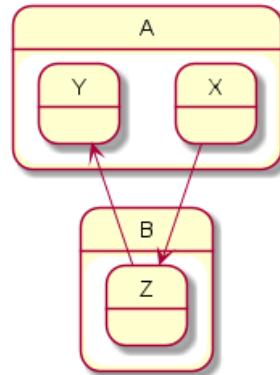
```

@startuml
state A {
  state X {
  }
  state Y {
  }
}

state B {
  state Z {
  }
}

X --> Z
Z --> Y
@enduml
  
```





[Ref. QA-3300]

## 9.4 Long name

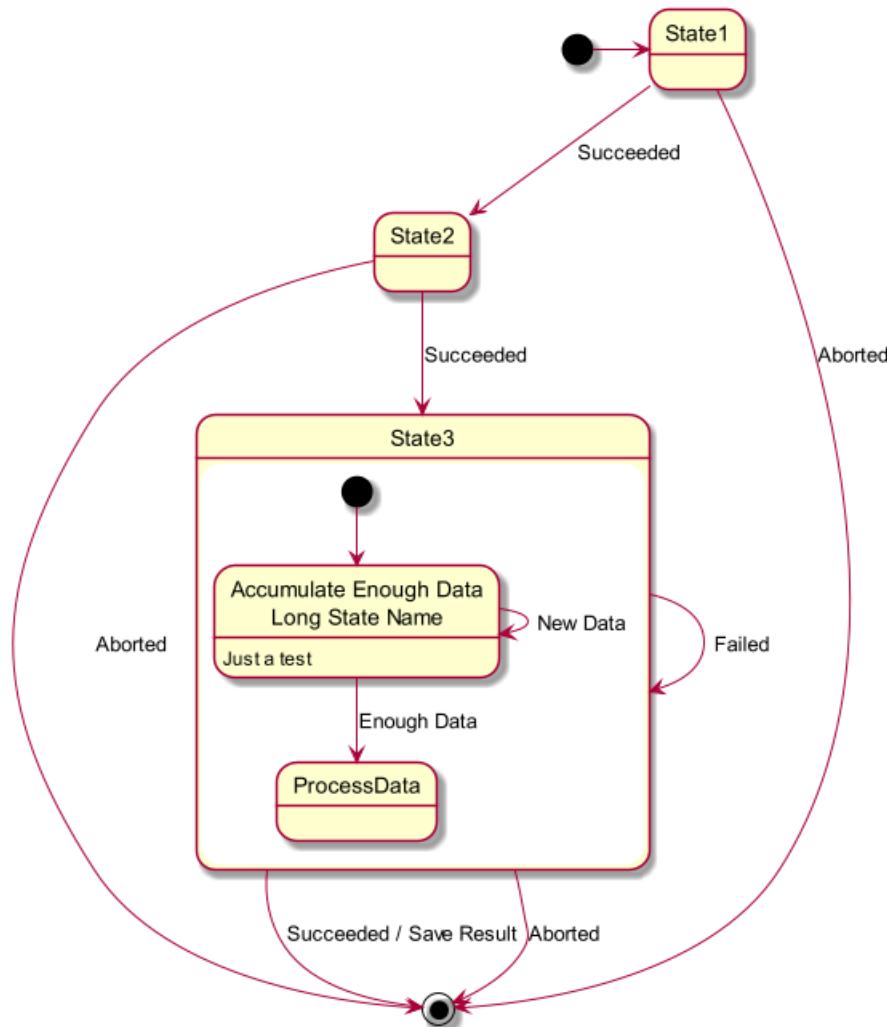
You can also use the `state` keyword to use long description for states.

```
@startuml
scale 600 width

[*] -> State1
State1 --> State2 : Succeeded
State1 --> [*] : Aborted
State2 --> State3 : Succeeded
State2 --> [*] : Aborted
state State3 {
    state "Accumulate Enough Data\nLong State Name" as long1
    long1 : Just a test
    [*] --> long1
    long1 --> long1 : New Data
    long1 --> ProcessData : Enough Data
}
State3 --> State3 : Failed
State3 --> [*] : Succeeded / Save Result
State3 --> [*] : Aborted

@enduml
```





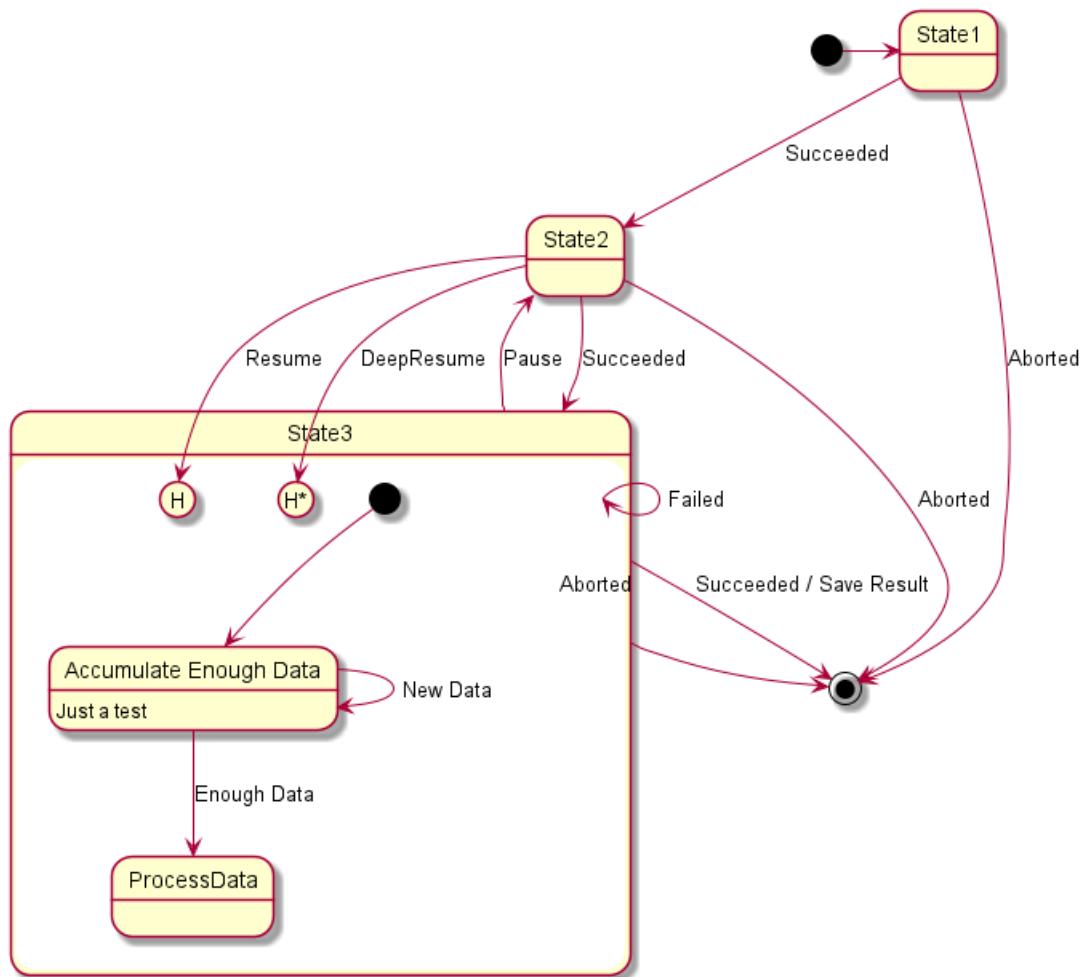
## 9.5 History [[H], [H\*]]

You can use `[H]` for the history and `[H*]` for the deep history of a substate.

```

@startuml
[*] --> State1
State1 --> State2 : Succeeded
State1 --> [*] : Aborted
State2 --> State3 : Succeeded
State2 --> [*] : Aborted
state State3 {
    state "Accumulate Enough Data" as long1
    long1 : Just a test
    [*] --> long1
    long1 --> long1 : New Data
    long1 --> ProcessData : Enough Data
    State2 --> [H] : Resume
}
State3 --> State2 : Pause
State2 --> State3[H*]: DeepResume
State3 --> State3 : Failed
State3 --> [*] : Succeeded / Save Result
State3 --> [*] : Aborted
@enduml
  
```





## 9.6 Fork [fork, join]

You can also fork and join using the <<fork>> and <<join>> stereotypes.

@startuml

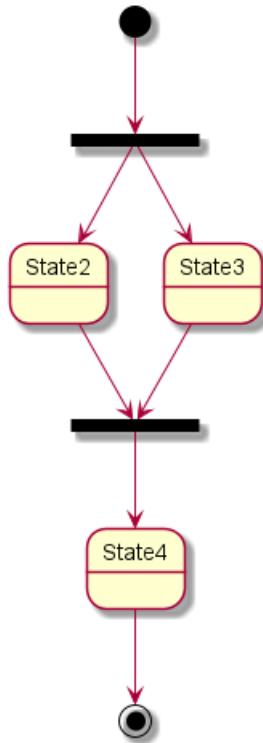
```

state fork_state <<fork>>
[*] --> fork_state
fork_state --> State2
fork_state --> State3

state join_state <<join>>
State2 --> join_state
State3 --> join_state
join_state --> State4
State4 --> [*]
    
```

@enduml





## 9.7 Concurrent state [-, ||]

You can define concurrent state into a composite state using either -- or || symbol as separator.

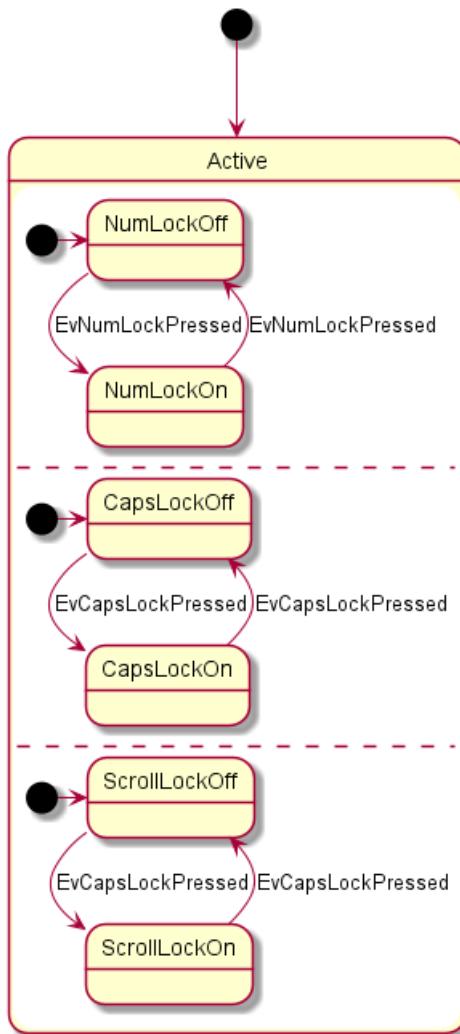
### 9.7.1 Horizontal separator --

```

@startuml
[*] --> Active

state Active {
    [*] -> NumLockOff
    NumLockOff --> NumLockOn : EvNumLockPressed
    NumLockOn --> NumLockOff : EvNumLockPressed
    --
    [*] -> CapsLockOff
    CapsLockOff --> CapsLockOn : EvCapsLockPressed
    CapsLockOn --> CapsLockOff : EvCapsLockPressed
    --
    [*] -> ScrollLockOff
    ScrollLockOff --> ScrollLockOn : EvCapsLockPressed
    ScrollLockOn --> ScrollLockOff : EvCapsLockPressed
}
@enduml
  
```





### 9.7.2 Vertical separator ||

```

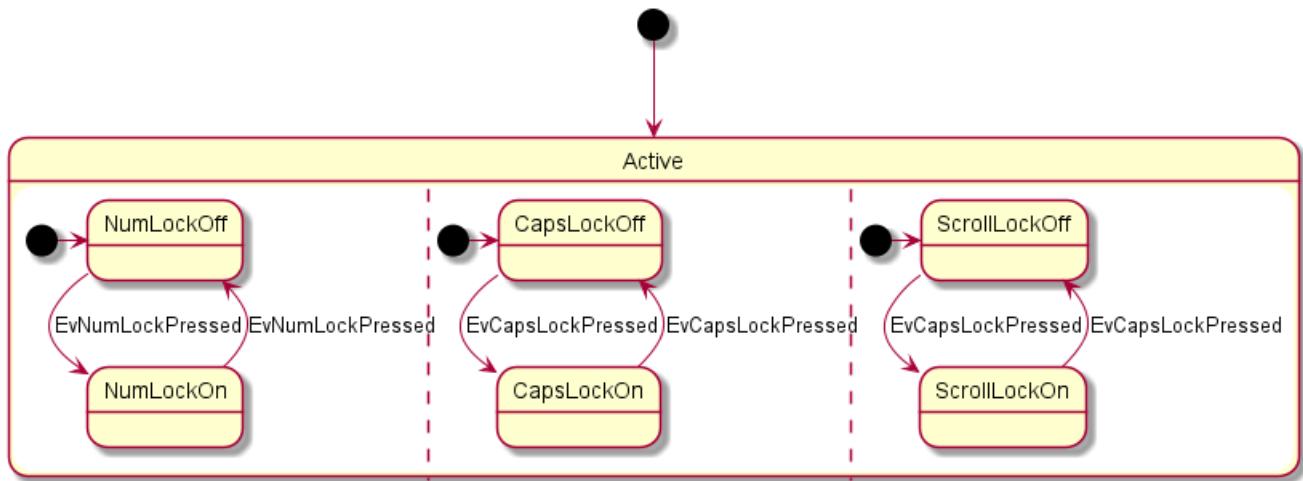
@startuml
[*] --> Active

state Active {
    [*] --> NumLockOff
    NumLockOff --> NumLockOn : EvNumLockPressed
    NumLockOn --> NumLockOff : EvNumLockPressed
    ||
    [*] --> CapsLockOff
    CapsLockOff --> CapsLockOn : EvCapsLockPressed
    CapsLockOn --> CapsLockOff : EvCapsLockPressed
    ||
    [*] --> ScrollLockOff
    ScrollLockOff --> ScrollLockOn : EvCapsLockPressed
    ScrollLockOn --> ScrollLockOff : EvCapsLockPressed
}

```

```
@enduml
```





## 9.8 Conditional [choice]

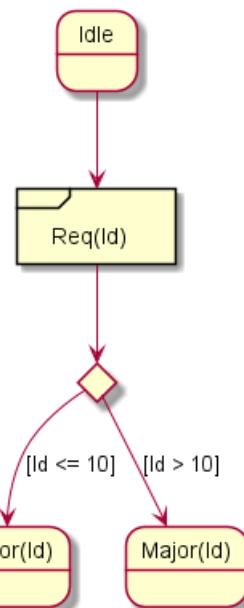
The stereotype <<choice>> can be used to use conditional state.

```

@startuml
state "Req(Id)" as ReqId <<sdlreceive>>
state "Minor(Id)" as MinorId
state "Major(Id)" as MajorId

state c <<choice>>

Idle --> ReqId
ReqId --> c
c --> MinorId : [Id <= 10]
c --> MajorId : [Id > 10]
@enduml
  
```



## 9.9 Stereotypes full example [choice, fork, join, end]

```

@startuml
state choice1 <<choice>>
state fork1    <<fork>>
  
```



```

state join2    <<join>>
state end3    <<end>>

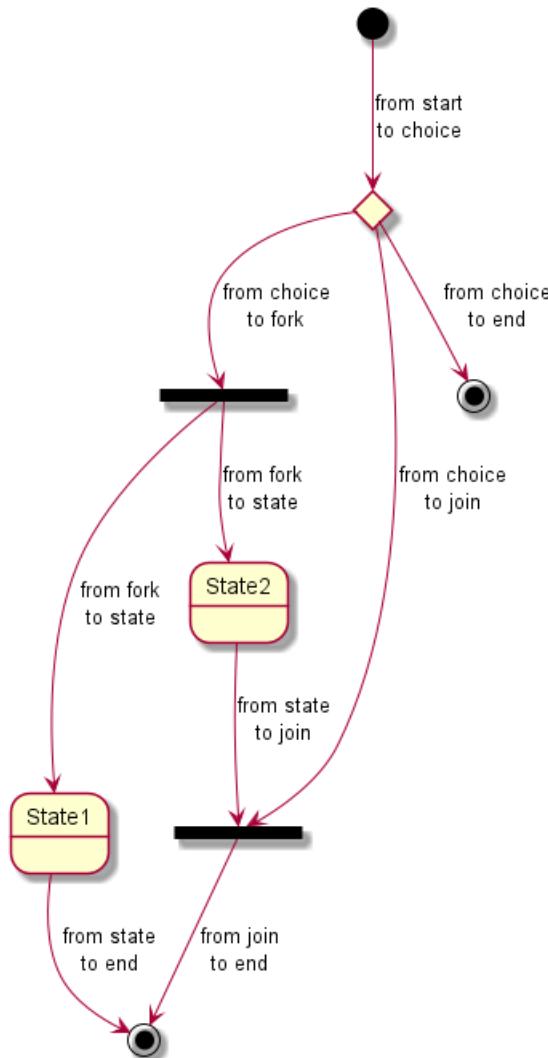
[*]      --> choice1 : from start\nto choice
choice1 --> fork1   : from choice\nto fork
choice1 --> join2   : from choice\nto join
choice1 --> end3    : from choice\nto end

fork1    ---> State1 : from fork\nto state
fork1    --> State2 : from fork\nto state

State2 --> join2   : from state\nto join
State1 --> [*]     : from state\nto end

join2 --> [*]     : from join\nto end
@enduml

```



[Ref. QA-404 and QA-1159]

## 9.10 Point [entryPoint, exitPoint]

You can add **point** with `<<entryPoint>>` and `<<exitPoint>>` stereotypes:

```

@startuml
state Somp {

```

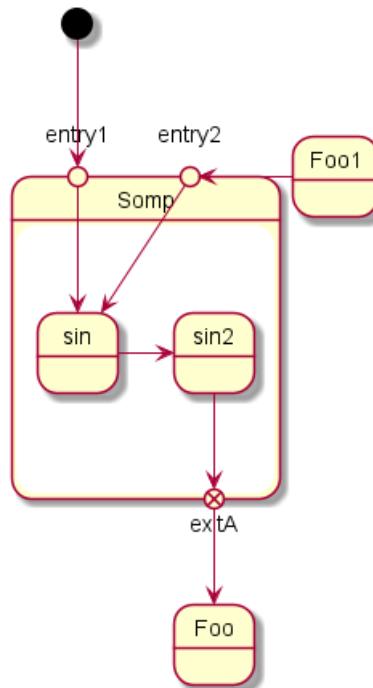


```

state entry1 <<entryPoint>>
state entry2 <<entryPoint>>
state sin
entry1 --> sin
entry2 -> sin
sin -> sin2
sin2 --> exitA <<exitPoint>>
}

[*] --> entry1
exitA --> Foo
Foo1 -> entry2
@enduml

```



## 9.11 Pin [inputPin, outputPin]

You can add **pin** with `<<inputPin>>` and `<<outputPin>>` stereotypes:

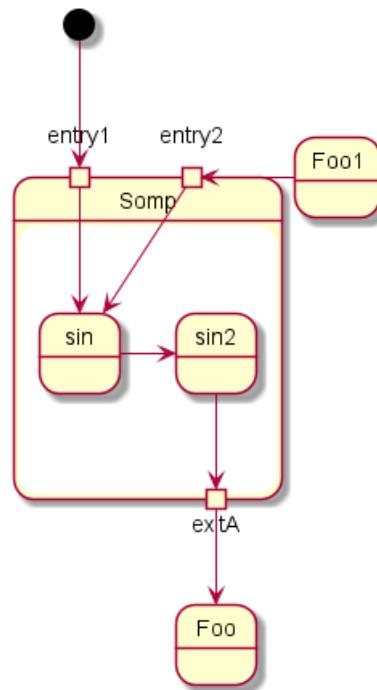
```

@startuml
state Somp {
    state entry1 <<inputPin>>
    state entry2 <<inputPin>>
    state sin
    entry1 --> sin
    entry2 -> sin
    sin -> sin2
    sin2 --> exitA <<outputPin>>
}

[*] --> entry1
exitA --> Foo
Foo1 -> entry2
@enduml

```





[Ref. QA-4309]

## 9.12 Expansion [expansionInput, expansionOutput]

You can add **expansion** with <<expansionInput>> and <<expansionOutput>> stereotypes:

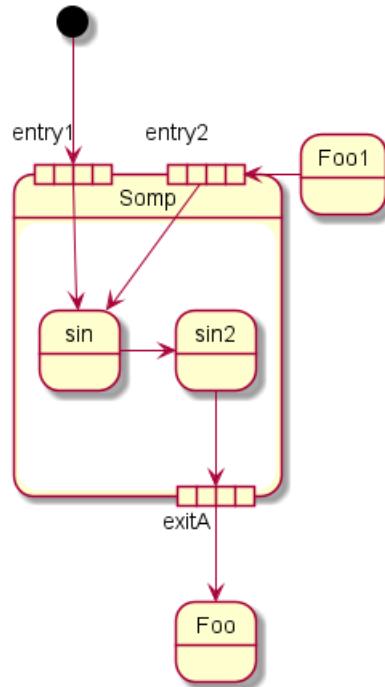
```

@startuml
state Somp {
    state entry1 <<expansionInput>>
    state entry2 <<expansionInput>>
    state sin
    entry1 --> sin
    entry2 -> sin
    sin -> sin2
    sin2 --> exitA <<expansionOutput>>
}

[*] --> entry1
exitA --> Foo
Foo1 -> entry2
@enduml

```





[Ref. QA-4309]

### 9.13 Arrow direction

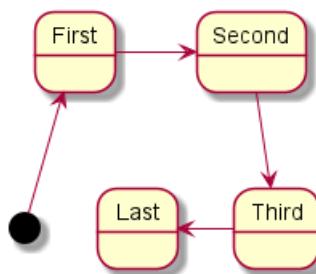
You can use `->` for horizontal arrows. It is possible to force arrow's direction using the following syntax:

- `-down->` or `-->`
- `-right->` or `->` (*default arrow*)
- `-left->`
- `-up->`

`@startuml`

```
[*] -up-> First
First -right-> Second
Second --> Third
Third -left-> Last
```

`@enduml`



You can shorten the arrow definition by using only the first character of the direction (for example, `-d-` instead of `-down-`) or the two first characters (`-do-`).

Please note that you should not abuse this functionality : *Graphviz* gives usually good results without tweaking.

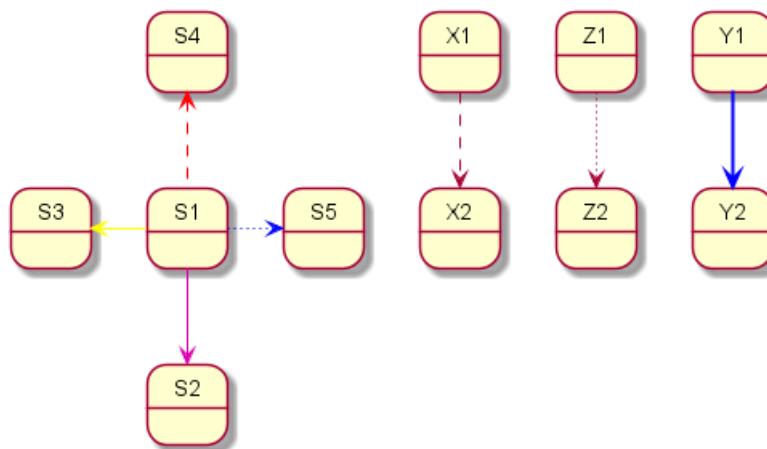


## 9.14 Change line color and style

You can change line color and/or line style.

```
@startuml
State S1
State S2
State S3
S1 -[#DD00AA]-> S2
S1 -left[#yellow]-> S3
S1 -up[#red,dashed]-> S4
S1 -right[dotted,#blue]-> S5
```

```
X1 -[dashed]-> X2
Z1 -[dotted]-> Z2
Y1 -[#blue,bold]-> Y2
@enduml
```



[Ref. Incubation: Change line color in state diagrams]

## 9.15 Note

You can also define notes using `note left of`, `note right of`, `note top of`, `note bottom of` keywords.

You can also define notes on several lines.

```
@startuml
```

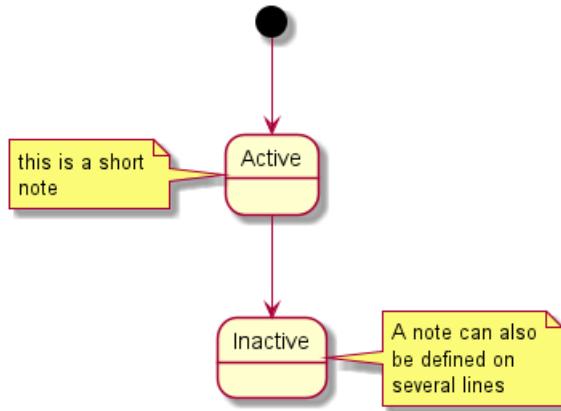
```
[*] --> Active
Active --> Inactive

note left of Active : this is a short\nnote

note right of Inactive
A note can also
be defined on
several lines
end note

@enduml
```





You can also have floating notes.

```
@startuml
```

```
state foo
note "This is a floating note" as N1
```

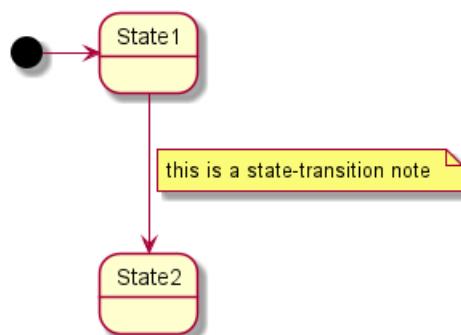
```
@enduml
```



## 9.16 Note on link

You can put notes on state-transition or link, with `note on link` keyword.

```
@startuml
[*] --> State1
State1 --> State2
note on link
  this is a state-transition note
end note
@enduml
```



## 9.17 More in notes

You can put notes on composite states.

```
@startuml
```

```
[*] --> NotShooting

state "Not Shooting State" as NotShooting {
```



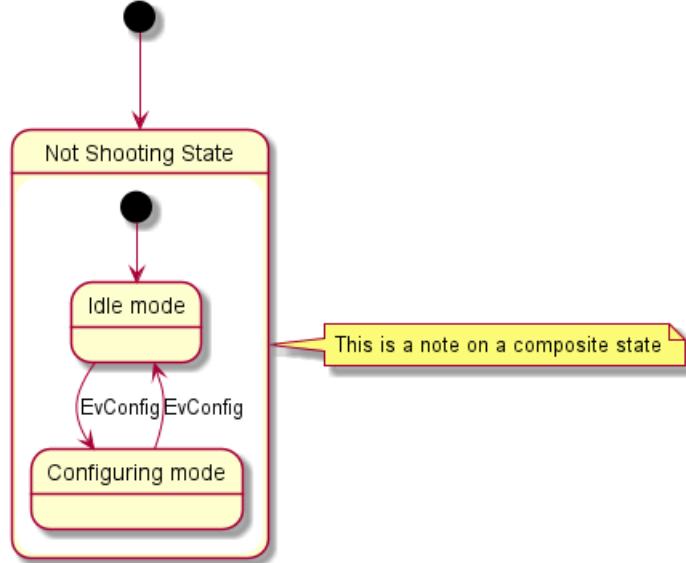
```

state "Idle mode" as Idle
state "Configuring mode" as Configuring
[*] --> Idle
Idle --> Configuring : EvConfig
Configuring --> Idle : EvConfig
}

note right of NotShooting : This is a note on a composite state

@enduml

```



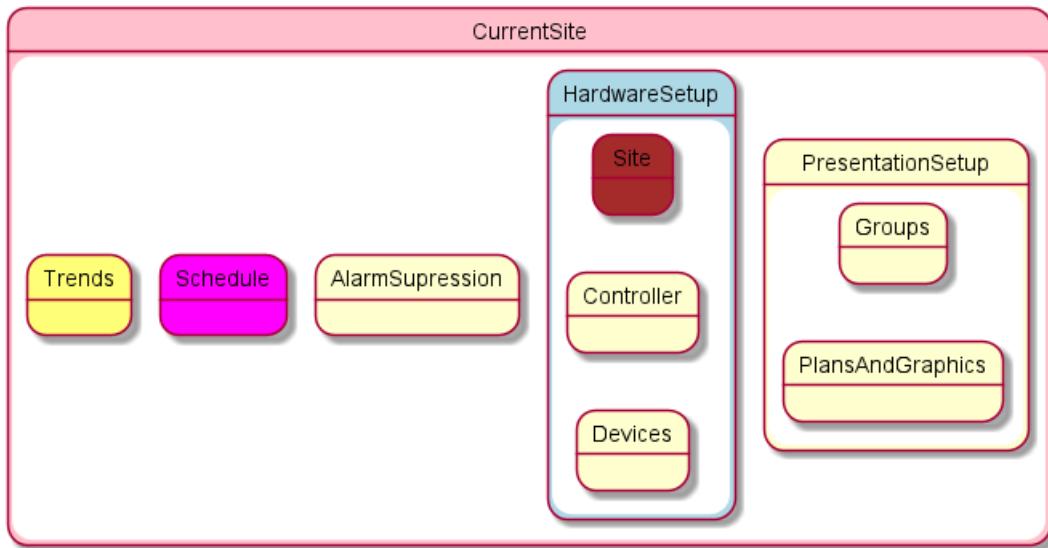
## 9.18 Inline color

```

@startuml
state CurrentSite #pink {
    state HardwareSetup #lightblue {
        state Site #brown
        Site -[hidden]-> Controller
        Controller -[hidden]-> Devices
    }
    state PresentationSetup{
        Groups -[hidden]-> PlansAndGraphics
    }
    state Trends #FFFF77
    state Schedule #magenta
    state AlarmSupression
}
@enduml

```





[Ref. QA-1812]

## 9.19 Skinparam

You can use the skinparam command to change colors and fonts for the drawing.

You can use this command :

- In the diagram definition, like any other commands,
- In an included file,
- In a configuration file, provided in the command line or the ANT task.

You can define specific color and fonts for stereotyped states.

```

@startuml
skinparam backgroundColor LightYellow
skinparam state {
    StartColor MediumBlue
    EndColor Red
    BackgroundColor Peru
    BackgroundColor<<Warning>> Olive
    BorderColor Gray
    FontName Impact
}

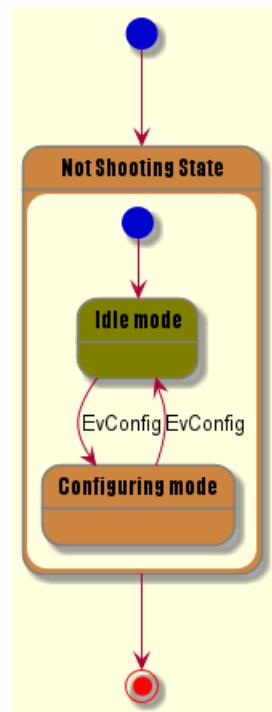
[*] --> NotShooting

state "Not Shooting State" as NotShooting {
    state "Idle mode" as Idle <<Warning>>
    state "Configuring mode" as Configuring
    [*] --> Idle
    Idle --> Configuring : EvConfig
    Configuring --> Idle : EvConfig
}

NotShooting --> [*]
@enduml

```





## 9.20 Changing style

You can change style.

@startuml

```

<style>
stateDiagram {
    BackgroundColor Peru
    'LineColor Gray
    FontName Impact
    FontColor Red
    arrow {
        FontSize 13
        LineColor Blue
    }
}
</style>
  
```

[\*] --> NotShooting

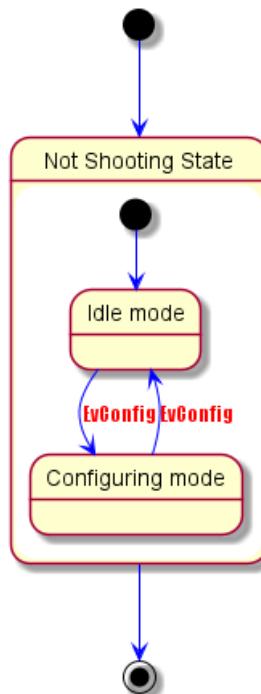
```

state "Not Shooting State" as NotShooting {
    state "Idle mode" as Idle <<Warning>>
    state "Configuring mode" as Configuring
    [*] --> Idle
    Idle --> Configuring : EvConfig
    Configuring --> Idle : EvConfig
}
  
```

NotShooting --> [\*]

@enduml





## 9.21 Change state color and style (inline style)

You can change the color or style of individual state using the following notation:

- `#color ##[style]color`

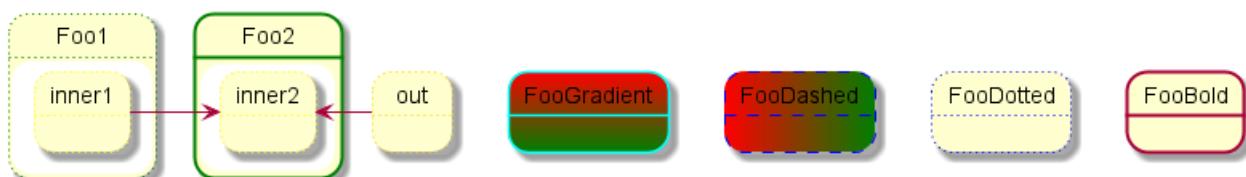
With background color first (`#color`), then line style and line color (`##[style]color`).

```

@startuml
state FooGradient #red-green ##00FFFF
state FooDashed #red|green ##[dashed]blue {
}
state FooDotted ##[dotted]blue {
}
state FooBold ##[bold] {
}
state Foo1 ##[dotted]green {
state inner1 ##[dotted]yellow
}

state out ##[dotted]gold

state Foo2 ##[bold]green {
state inner2 ##[dotted]yellow
}
inner1 -> inner2
out -> inner2
@enduml
    
```



[Ref. QA-1487]

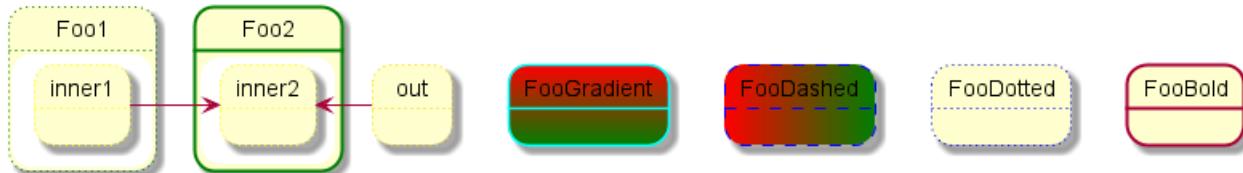
- #color;line:color;line.[bold|dashed|dotted];text:color

**TODO:**FIXME text:color seems not to be taken into account **TODO:**FIXME

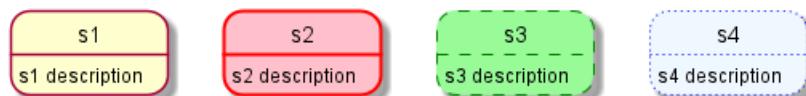
```
@startuml
@startuml
state FooGradient #red-green;line:00FFFF
state FooDashed #red|green;line.dashed;line:blue {
}
state FooDotted #line.dotted;line:blue {
}
state FooBold #line.bold {
}
state Foo1 #line.dotted;line:green {
state inner1 #line.dotted;line:yellow
}

state out #line.dotted;line:gold

state Foo2 #line.bold;line:green {
state inner2 #line.dotted;line:yellow
}
inner1 -> inner2
out -> inner2
@enduml
@enduml
```



```
@startuml
state s1 : s1 description
state s2 #pink;line:red;line.bold;text:red : s2 description
state s3 #palegreen;line:green;line.dashed;text:green : s3 description
state s4 #aliceblue;line:blue;line.dotted;text:blue : s4 description
@enduml
```



[Adapted from QA-3770]



## 10 Timing Diagram

This is still under construction. You can propose new features if you need some.

### 10.1 Declaring participant

You declare participant using the following keywords, depending on how you want them to be drawn.

- **concise**: A simplified signal designed to show the movement of data (great for messages).
- **robust**: A complex line signal designed to show the transition from one state to another (can have many states).
- **clock**: A 'clocked' signal that repeatedly transitions from high to low
- **binary**: A specific signal restricted to only 2 states (binary).

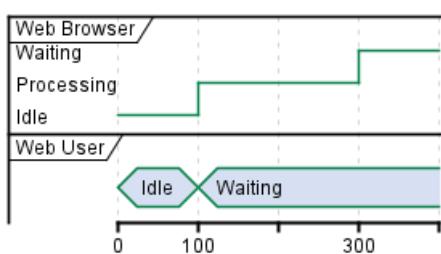
You define state change using the @ notation, and the **is** verb.

```
@startuml
robust "Web Browser" as WB
concise "Web User" as WU
```

```
@0
WU is Idle
WB is Idle
```

```
@100
WU is Waiting
WB is Processing
```

```
@300
WB is Waiting
@enduml
```



### 10.2 Binary and Clock

It's also possible to have binary and clock signal, using the following keywords:

- **binary**
- **clock**

```
@startuml
clock clk with period 1
binary "Enable" as EN
```

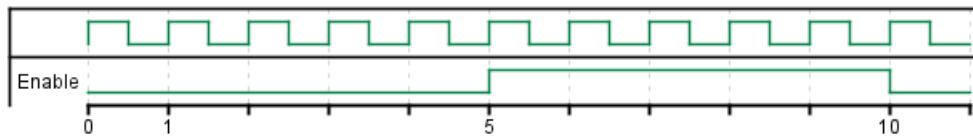
```
@0
EN is low
```

```
@5
EN is high
```

```
@10
```



EN is low  
@enduml



### 10.3 Adding message

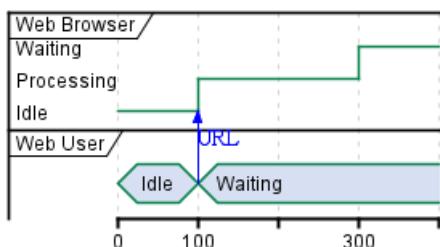
You can add message using the following syntax.

```
@startuml
robust "Web Browser" as WB
concise "Web User" as WU
```

```
@0
WU is Idle
WB is Idle
```

```
@100
WU -> WB : URL
WU is Waiting
WB is Processing
```

```
@300
WB is Waiting
@enduml
```



### 10.4 Relative time

It is possible to use relative time with @.

```
@startuml
robust "DNS Resolver" as DNS
robust "Web Browser" as WB
concise "Web User" as WU
```

```
@0
WU is Idle
WB is Idle
DNS is Idle
```

```
@+100
WU -> WB : URL
WU is Waiting
WB is Processing
```

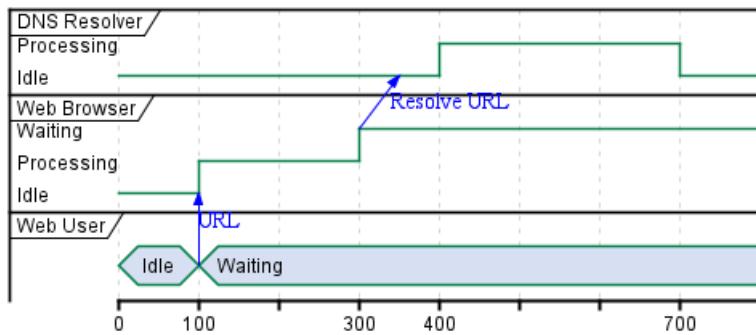
```
@+200
WB is Waiting
```



WB → DNS@+50 : Resolve URL

@+100  
DNS is Processing

@+300  
DNS is Idle  
@enduml



## 10.5 Anchor Points

Instead of using absolute or relative time on an absolute time you can define a time as an anchor point by using the `as` keyword and starting the name with a `:`.

`@XX as :<anchor point name>`

```
@startuml
clock clk with period 1
binary "enable" as EN
concise "dataBus" as db
```

```
@0 as :start
@5 as :en_high
@10 as :en_low
```

`@:start`  
EN is low  
db is "0x0000"

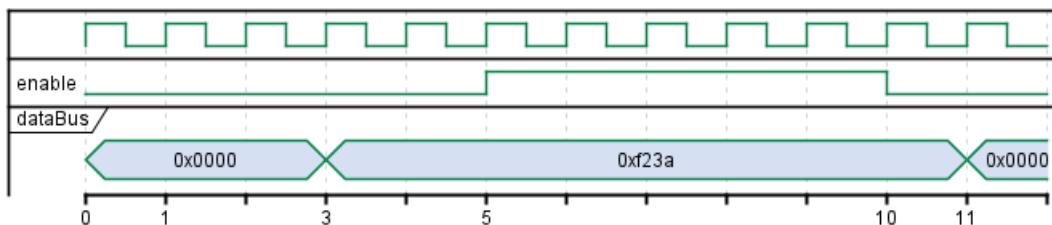
`@:en_high`  
EN is high

`@:en_low`  
EN is low

`@:en_high-2`  
db is "0xf23a"

`@:en_high+6`  
db is "0x0000"  
@enduml





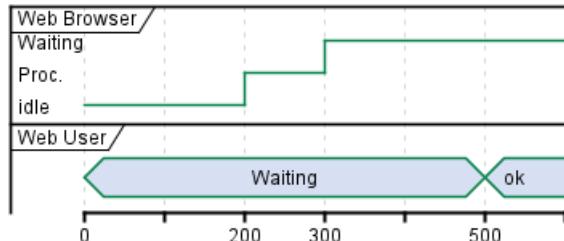
## 10.6 Participant oriented

Rather than declare the diagram in chronological order, you can define it by participant.

```
@startuml
robust "Web Browser" as WB
concise "Web User" as WU

@WB
0 is idle
+200 is Proc.
+100 is Waiting

@WU
0 is Waiting
+500 is ok
@enduml
```

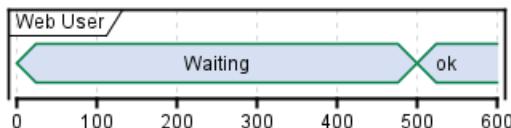


## 10.7 Setting scale

You can also set a specific scale.

```
@startuml
concise "Web User" as WU
scale 100 as 50 pixels

@WU
0 is Waiting
+500 is ok
@enduml
```



## 10.8 Initial state

You can also define an initial state.

```
@startuml
robust "Web Browser" as WB
```

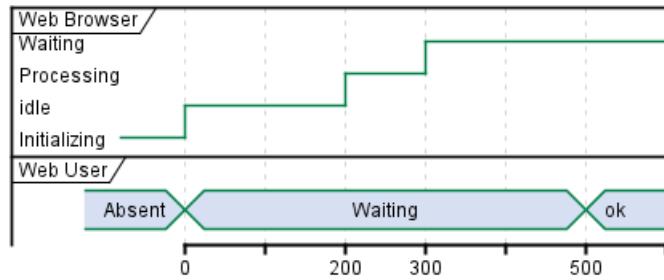


concise "Web User" as WU

WB is Initializing  
WU is Absent

@WB  
0 is idle  
+200 is Processing  
+100 is Waiting

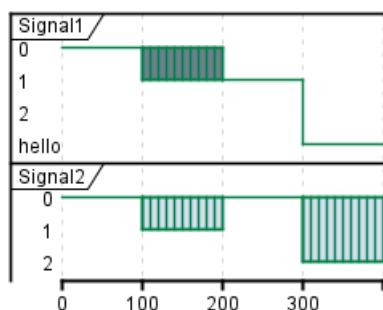
@WU  
0 is Waiting  
+500 is ok  
@enduml



## 10.9 Intricated state

A signal could be in some undefined state.

```
@startuml
robust "Signal1" as S1
robust "Signal2" as S2
S1 has 0,1,2,hello
S2 has 0,1,2
@0
S1 is 0
S2 is 0
@100
S1 is {0,1} #SlateGrey
S2 is {0,1}
@200
S1 is 1
S2 is 0
@300
S1 is hello
S2 is {0,2}
@enduml
```



## 10.10 Hidden state

It is also possible to hide some state.

```
@startuml
concise "Web User" as WU

@0
WU is {-}

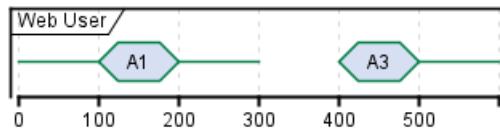
@100
WU is A1

@200
WU is {-}

@300
WU is {hidden}

@400
WU is A3

@500
WU is {-}
@enduml
```



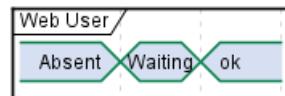
## 10.11 Hide time axis

It is possible to hide time axis.

```
@startuml
hide time-axis
concise "Web User" as WU

WU is Absent

@WU
0 is Waiting
+500 is ok
@enduml
```



## 10.12 Using Time and Date

It is possible to use time or date.

```
@startuml
robust "Web Browser" as WB
concise "Web User" as WU

@2019/07/02
WU is Idle
```



WB is Idle

@2019/07/04

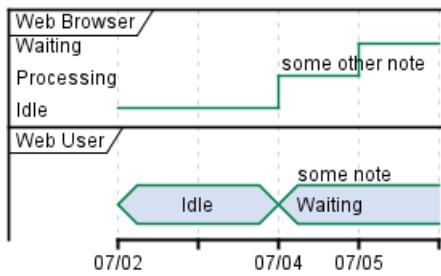
WU is Waiting : some note

WB is Processing : some other note

@2019/07/05

WB is Waiting

@enduml



@startuml

robust "Web Browser" as WB

concise "Web User" as WU

@1:15:00

WU is Idle

WB is Idle

@1:16:30

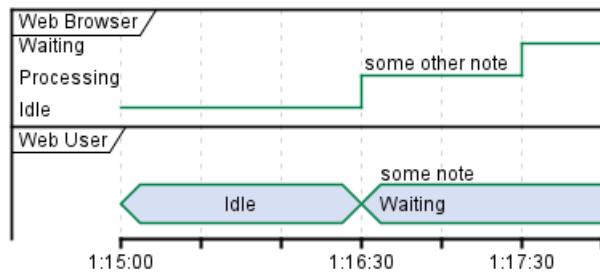
WU is Waiting : some note

WB is Processing : some other note

@1:17:30

WB is Waiting

@enduml



## 10.13 Adding constraint

It is possible to display time constraints on the diagrams.

@startuml

robust "Web Browser" as WB

concise "Web User" as WU

WB is Initializing

WU is Absent

@WB

0 is idle

+200 is Processing



```
+100 is Waiting
WB@0 <-> @50 : {50 ms lag}
```

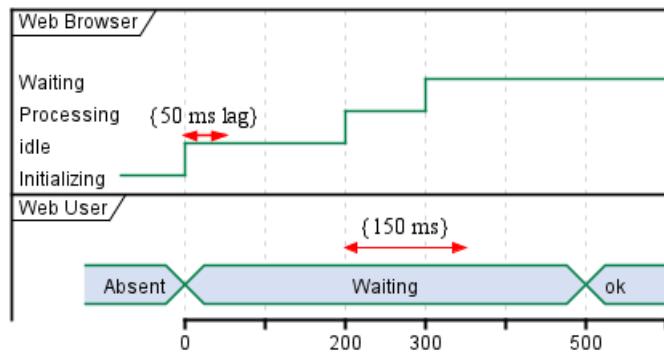
@WU

0 is Waiting

+500 is ok

```
@200 <-> @+150 : {150 ms}
```

@enduml



## 10.14 Highlighted period

You can highlight a part of diagram.

```
@startuml
robust "Web Browser" as WB
concise "Web User" as WU
```

```
@0
WU is Idle
WB is Idle
```

```
@100
WU -> WB : URL
WU is Waiting #LightCyan;line:Aqua
```

```
@200
WB is Proc.
```

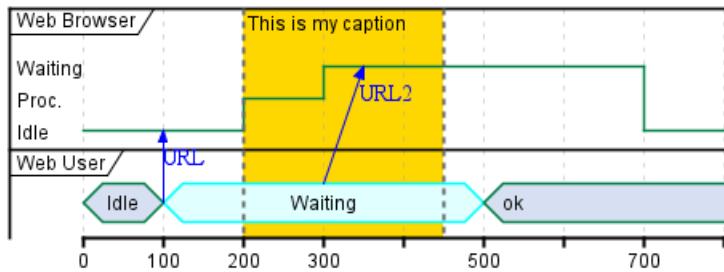
```
@300
WU -> WB@350 : URL2
WB is Waiting
```

```
@+200
WU is ok
```

```
@+200
WB is Idle
```

```
highlight 200 to 450 #Gold;line:DimGrey : This is my caption
@enduml
```





## 10.15 Adding texts

You can optionally add a title, a header, a footer, a legend and a caption:

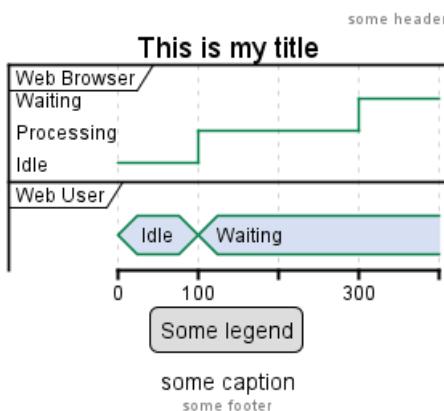
```
@startuml
Title This is my title
header: some header
footer: some footer
legend
Some legend
end legend
caption some caption

robust "Web Browser" as WB
concise "Web User" as WU

@0
WU is Idle
WB is Idle

@100
WU is Waiting
WB is Processing

@300
WB is Waiting
@enduml
```



## 10.16 Complete example

Thanks to Adam Rosien for this example.

```
@startuml
concise "Client" as Client
concise "Server" as Server
```



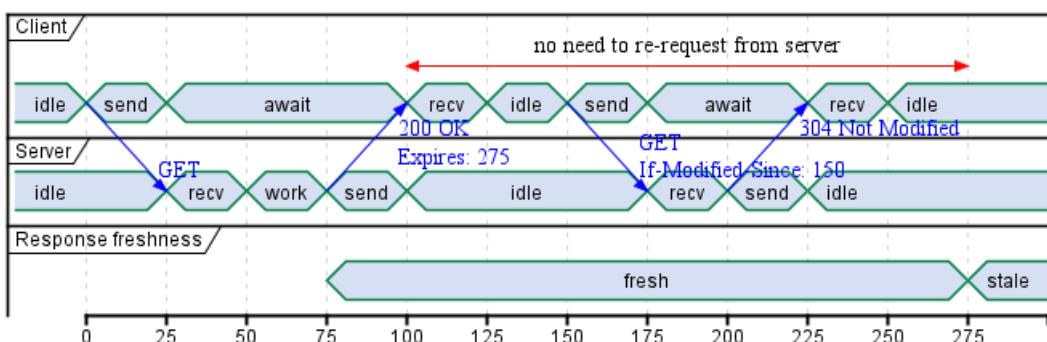
concise "Response freshness" as Cache

Server is idle  
Client is idle

```
@Client
0 is send
Client -> Server@+25 : GET
+25 is await
+75 is recv
+25 is idle
+25 is send
Client -> Server@+25 : GET\nIf-MODIFIED-Since: 150
+25 is await
+50 is recv
+25 is idle
@100 <-> @275 : no need to re-request from server
```

```
@Server
25 is recv
+25 is work
+25 is send
Server -> Client@+25 : 200 OK\nExpires: 275
+25 is idle
+75 is recv
+25 is send
Server -> Client@+25 : 304 Not Modified
+25 is idle
```

```
@Cache
75 is fresh
+200 is stale
@enduml
```



## 10.17 Digital Example

```
@startuml
scale 5 as 150 pixels
```

```
clock clk with period 1
binary "enable" as en
binary "R/W" as rw
binary "data Valid" as dv
concise "dataBus" as db
concise "address bus" as addr
```



```

@6 as :write_beg
@10 as :write_end

@15 as :read_beg
@19 as :read_end

@0
en is low
db is "0x0"
addr is "0x03f"
rw is low
dv is 0

@:write_beg-3
en is high
@:write_beg-2
db is "0xDEADBEEF"
@:write_beg-1
dv is 1
@:write_beg
rw is high

@:write_end
rw is low
dv is low
@:write_end+1
rw is low
db is "0x0"
addr is "0x23"

@12
dv is high
@13
db is "0xFFFF"

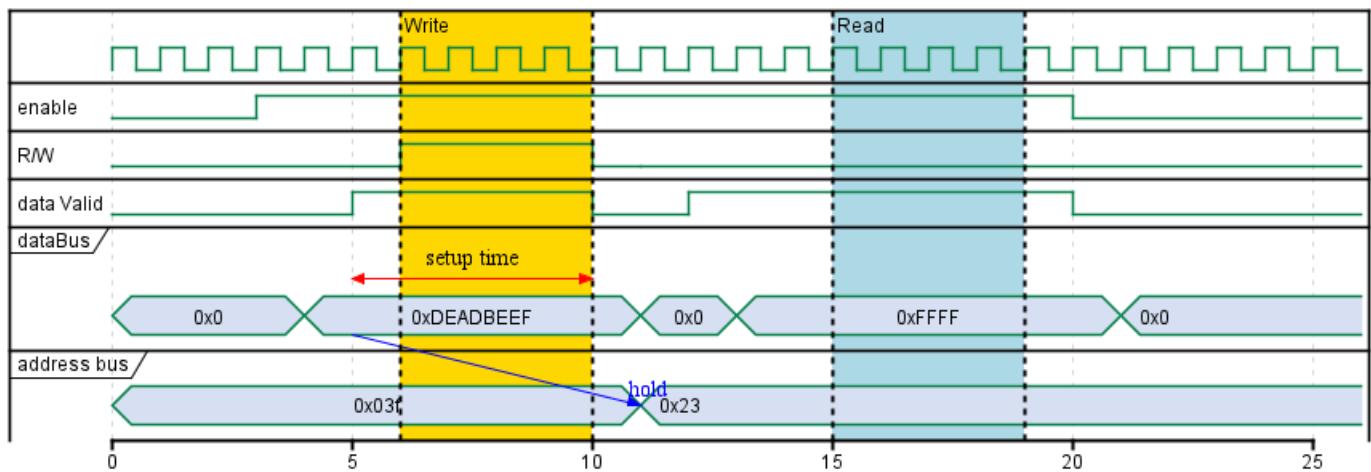
@20
en is low
dv is low
@21
db is "0x0"

highlight :write_beg to :write_end #Gold:Write
highlight :read_beg to :read_end #lightBlue:Read

db@:write_beg-1 <-> @:write_end : setup time
db@:write_beg-1 -> addr@:write_end+1 : hold
@enduml

```





## 10.18 Adding color

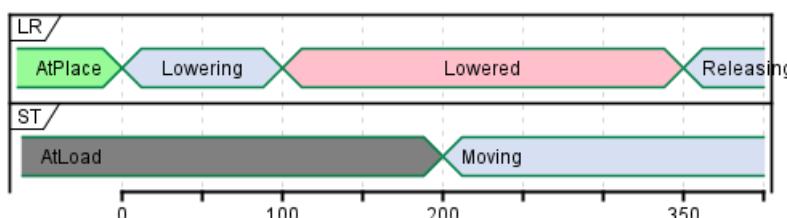
You can add color.

```
@startuml
concise "LR" as LR
concise "ST" as ST
```

LR is AtPlace #palegreen  
ST is AtLoad #gray

```
@LR
0 is Lowering
100 is Lowered #pink
350 is Releasing
```

```
@ST
200 is Moving
@enduml
```



[Ref. QA-5776]



## 11 Display JSON Data

JSON format is widely used in software.

You can use PlantUML to visualize your data.

To activate this feature, the diagram must:

- begin with `@startjson` keyword
- end with `@endjson` keyword.

```
@startjson
{
    "fruit": "Apple",
    "size": "Large",
    "color": "Red"
}
@endjson
```

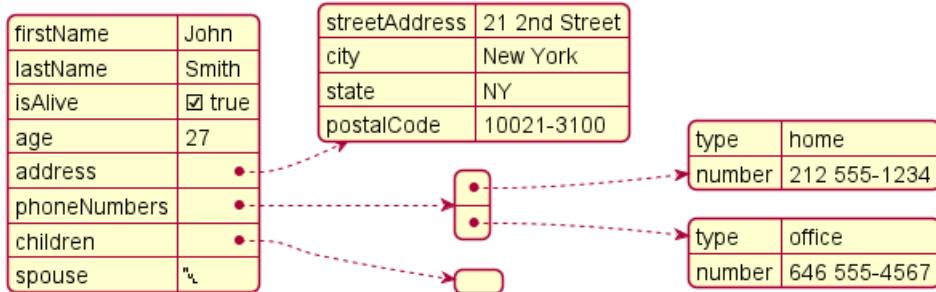
fruit	Apple
size	Large
color	Red

### 11.1 Complex example

You can use complex JSON structure.

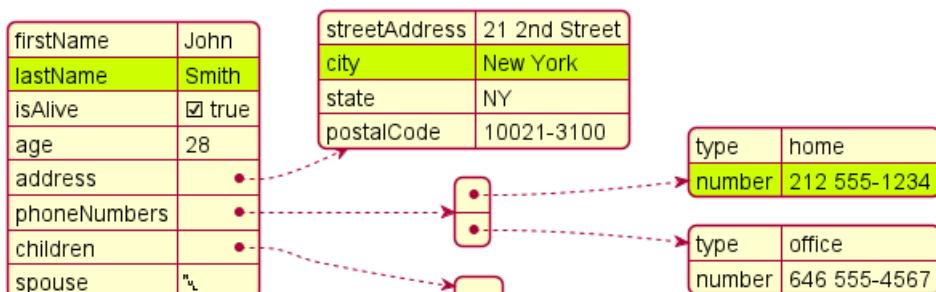
```
@startjson
{
    "firstName": "John",
    "lastName": "Smith",
    "isAlive": true,
    "age": 27,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3100"
    },
    "phoneNumbers": [
        {
            "type": "home",
            "number": "212 555-1234"
        },
        {
            "type": "office",
            "number": "646 555-4567"
        }
    ],
    "children": [],
    "spouse": null
}
@endjson
```





## 11.2 Highlight parts

```
@startjson
#highlight "lastName"
#highlight "address" / "city"
#highlight "phoneNumbers" / "0" / "number"
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 28,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
@endjson
```



## 11.3 JSON basic element

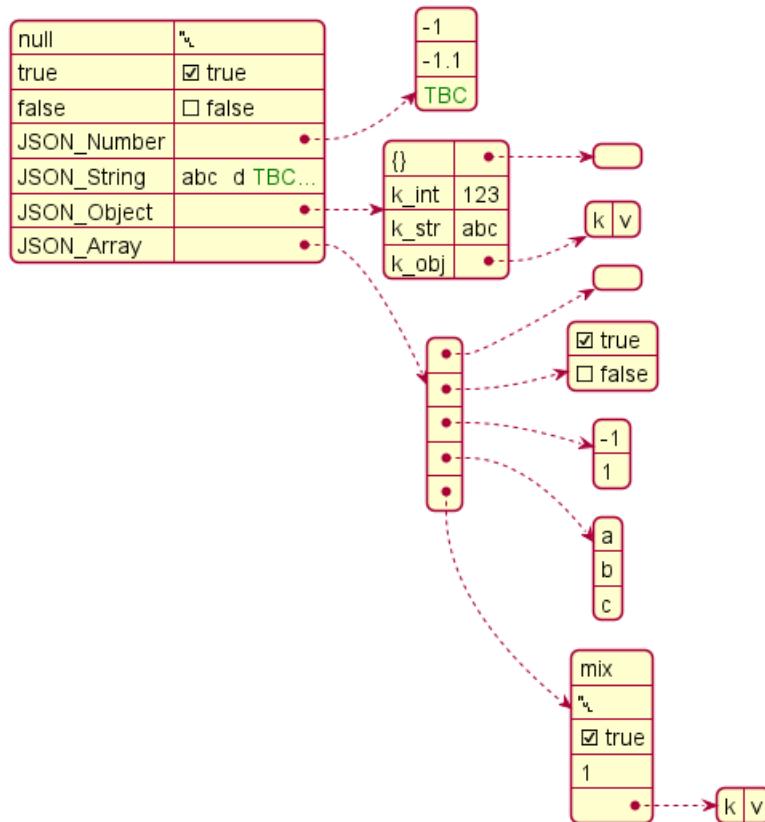
### 11.3.1 Synthesis of all JSON basic element

```
@startjson
```



```
{
  "null": null,
  "true": true,
  "false": false,
  "JSON_Number": [-1, -1.1, "<color:green>TBC"],
  "JSON_String": "a\nb\rc\td <color:green>TBC...",
  "JSON_Object": {
    "{}": {},
    "k_int": 123,
    "k_str": "abc",
    "k_obj": {"k": "v"}
  },
  "JSON_Array" : [
    [],
    [true, false],
    [-1, 1],
    ["a", "b", "c"],
    ["mix", null, true, 1, {"k": "v"}]
  ]
}
}

@endjson
```



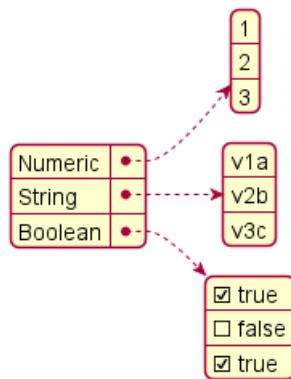
## 11.4 JSON array or table

### 11.4.1 Array type

```
@startjson
{
  "Numeric": [1, 2, 3],
  "String": ["v1a", "v2b", "v3c"],
  "Boolean": [true, false, true]
}
```



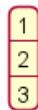
```
@endjson
```



#### 11.4.2 Minimal array or table

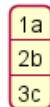
#### 11.4.3 Number array

```
@startjson
[1, 2, 3]
@endjson
```



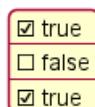
#### 11.4.4 String array

```
@startjson
["1a", "2b", "3c"]
@endjson
```



#### 11.4.5 Boolean array

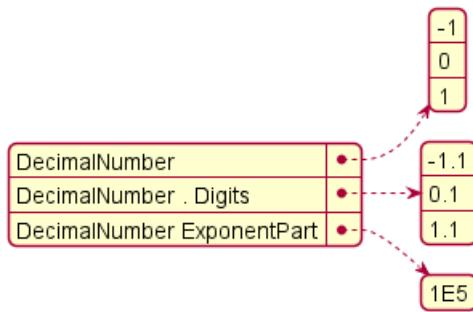
```
@startjson
[true, false, true]
@endjson
```



## 11.5 JSON numbers

```
@startjson
{
  "DecimalNumber": [-1, 0, 1],
  "DecimalNumber . Digits": [-1.1, 0.1, 1.1],
  "DecimalNumber ExponentPart": [1E5]
}
@endjson
```





## 11.6 JSON strings

### 11.6.1 JSON Unicode

On JSON you can use Unicode directly or by using escaped form like .

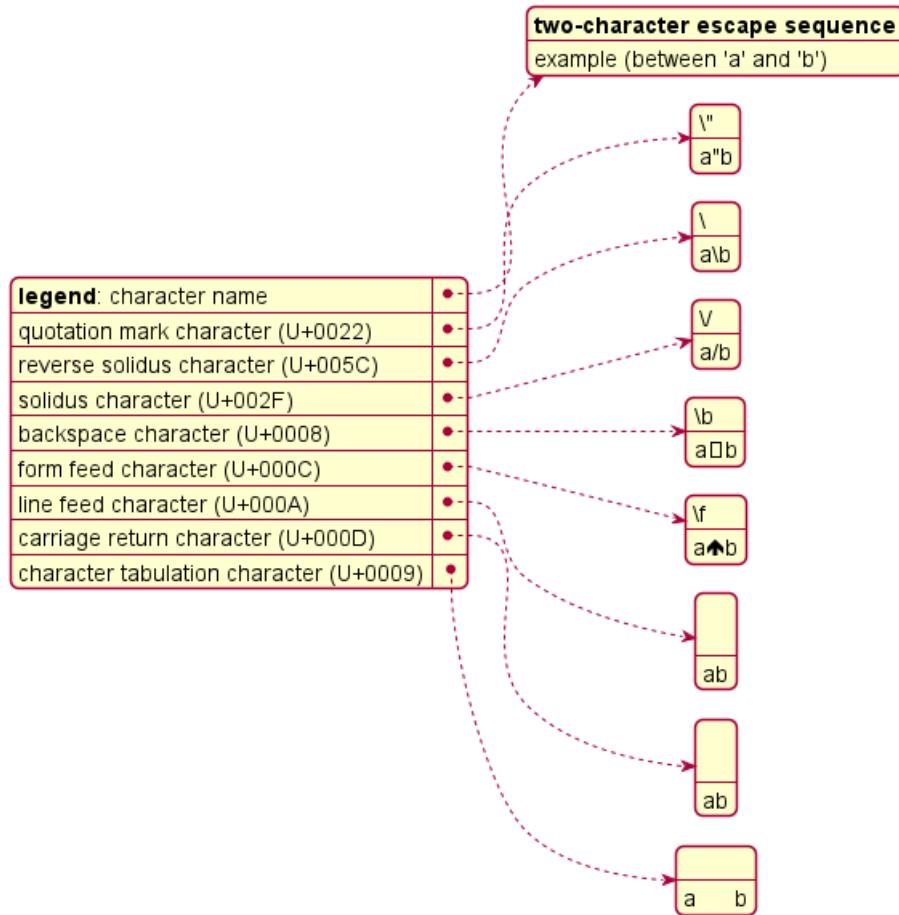
```
@startjson
{
  "<color:blue><b>code": "<color:blue><b>value",
  "a\\u005Cb": "a\u005Cb",
  "\\uD83D\\uDE10": "\uD83D\uDE10",
  " ":
}
@endjson
```

code	value
a\u005Cb	a\b
\uD83D\uDE10	😊
😊	😊

### 11.6.2 JSON two-character escape sequence

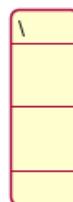
```
@startjson
{
  "**legend**: character name": [
    "quotation mark character (U+0022)": ["\\\"", "a\"b"],
    "reverse solidus character (U+005C)": ["\\\\\\\", "a\\\"b"],
    "solidus character (U+002F)": ["\\\\\\/", "a\\/b"],
    "backspace character (U+0008)": ["\\\\b", "a\\bb"],
    "form feed character (U+000C)": ["\\\\f", "a\\fb"],
    "line feed character (U+000A)": ["\\\\n", "a\\nb"],
    "carriage return character (U+000D)": ["\\\\r", "a\\rb"],
    "character tabulation character (U+0009)": ["\\\\t", "a\\tb"]
  ]
}
@endjson
```





**TODO:** FIXME FIXME or not , on the same item as management in PlantUML **TODO:** FIXME

```
@startjson
[
  "\\\\",
  "\\n",
  "\\r",
  "\\t"
]
@endjson
```



## 11.7 Minimal JSON examples

```
@startjson
"Hello world!"
@endjson
```

Hello world!

```
@startjson
```



```
42
@endjson
```

42

```
@startjson
true
@endjson
```

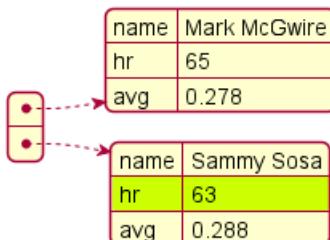
true

(Examples come from STD 90 - Examples)

## 11.8 Using (global) style

### 11.8.1 Without style (by default)

```
@startjson
#highlight "1" / "hr"
[
  {
    "name": "Mark McGwire",
    "hr": 65,
    "avg": 0.278
  },
  {
    "name": "Sammy Sosa",
    "hr": 63,
    "avg": 0.288
  }
]
@endjson
```



### 11.8.2 With style

You can use style to change rendering of elements.

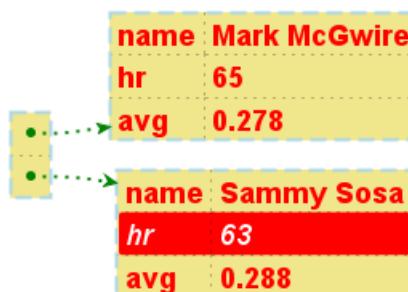
```
@startjson
<style>
jsonDiagram {
  node {
    BackGroundColor Khaki
    LineColor lightblue
    FontName Helvetica
    FontColor red
    FontSize 18
    FontStyle bold
    RoundCorner 0
    LineThickness 2
    LineStyle 10;5
```



```

separator {
    LineThickness 0.5
    LineColor black
    LineStyle 1;5
}
arrow {
    BackGroundColor lightblue
    LineColor green
    LineThickness 2
    LineStyle 2;5
}
highlight {
    BackGroundColor red
    FontColor white
    FontStyle italic
}
}
</style>
#highlight "1" / "hr"
[
{
    "name": "Mark McGwire",
    "hr": 65,
    "avg": 0.278
},
{
    "name": "Sammy Sosa",
    "hr": 63,
    "avg": 0.288
}
]
@endjson

```



[Adapted from QA-13123 and QA-13288]



## 12 Display YAML Data

YAML format is widely used in software.

You can use PlantUML to visualize your data.

To activate this feature, the diagram must:

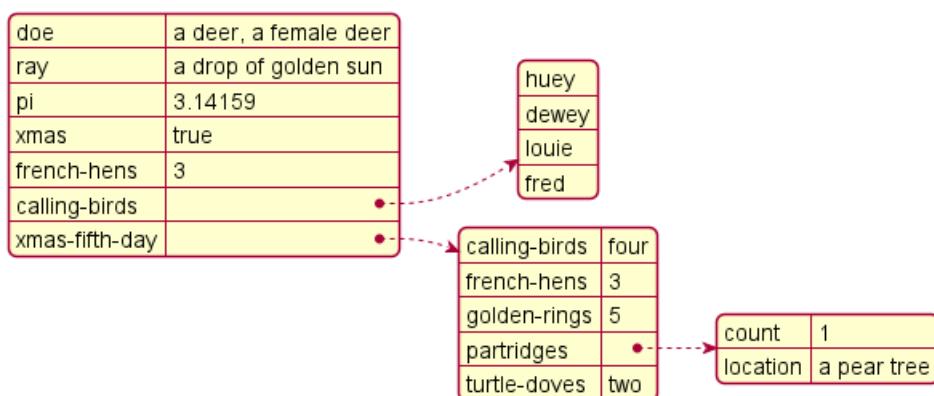
- begin with `@startyaml` keyword
- end with `@endyaml` keyword.

```
@startyaml
fruit: Apple
size: Large
color: Red
@endyaml
```

fruit	Apple
size	Large
color	Red

### 12.1 Complex example

```
@startyaml
doe: "a deer, a female deer"
ray: "a drop of golden sun"
pi: 3.14159
xmas: true
french-hens: 3
calling-birds:
- huey
- dewey
- louie
- fred
xmas-fifth-day:
calling-birds: four
french-hens: 3
golden-rings: 5
partridges:
count: 1
location: "a pear tree"
turtle-doves: two
@endyaml
```



## 12.2 Specific key (with symbols or unicode)

```
@startyaml
@fruit: Apple
$size: Large
&color: Red
: Heart
%: Per mille
@endyaml
```

@fruit	Apple
\$size	Large
&color	Red
♥	Heart
%	Per mille

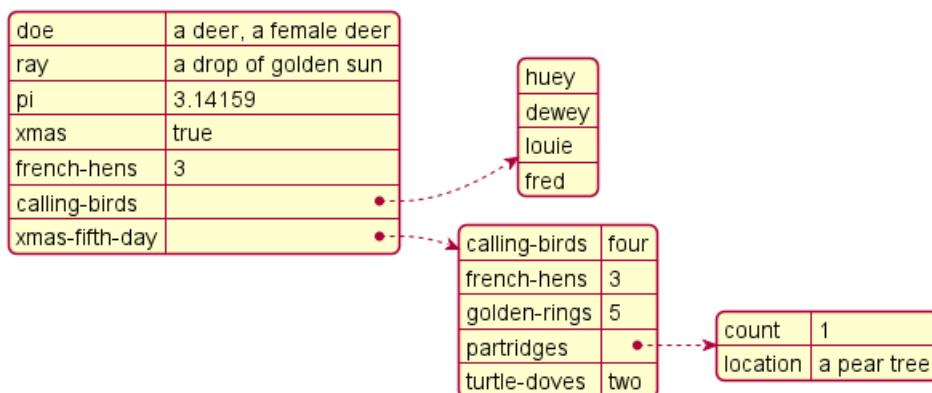
[Ref. QA-13376]

## 12.3 Highlight parts

### 12.3.1 Normal style

```
@startyaml
#highlight "french-hens"
#highlight "xmas-fifth-day" / "partridges"

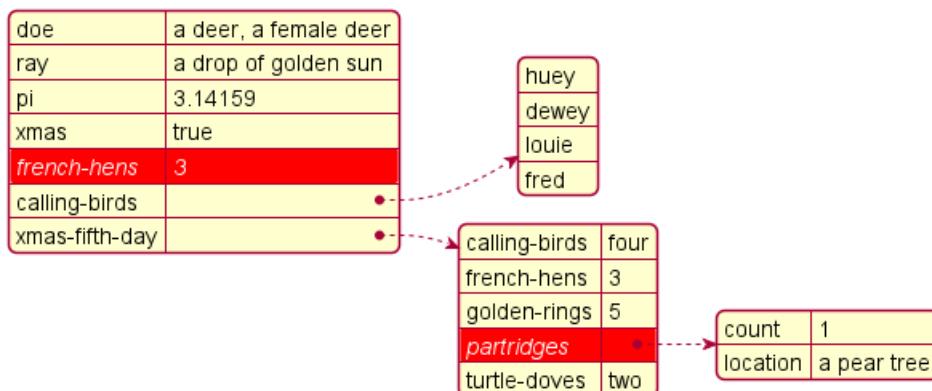
doe: "a deer, a female deer"
ray: "a drop of golden sun"
pi: 3.14159
xmas: true
french-hens: 3
calling-birds:
- huey
- dewey
- louie
- fred
xmas-fifth-day:
calling-birds: four
french-hens: 3
golden-rings: 5
partridges:
count: 1
location: "a pear tree"
turtle-doves: two
@endyaml
```



### 12.3.2 Customised style

```
@startyaml
<style>
yamlDiagram {
    highlight {
        BackGroundColor red
        FontColor white
        FontStyle italic
    }
}
</style>
#highlight "french-hens"
#highlight "xmas-fifth-day" / "partridges"

doe: "a deer, a female deer"
ray: "a drop of golden sun"
pi: 3.14159
xmas: true
french-hens: 3
calling-birds:
- huey
- dewey
- louie
- fred
xmas-fifth-day:
calling-birds: four
french-hens: 3
golden-rings: 5
partridges:
count: 1
location: "a pear tree"
turtle-doves: two
@endyaml
```



[Ref. QA-13288]

## 12.4 Using (global) style

### 12.4.1 Without style (*by default*)

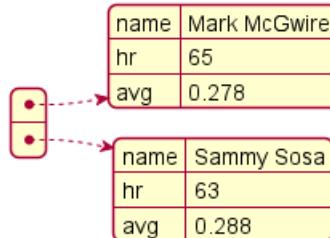
```
@startyaml
-
  name: Mark McGwire
  hr: 65
  avg: 0.278
```



```

-
  name: Sammy Sosa
  hr: 63
  avg: 0.288
@endyaml

```



### 12.4.2 With style

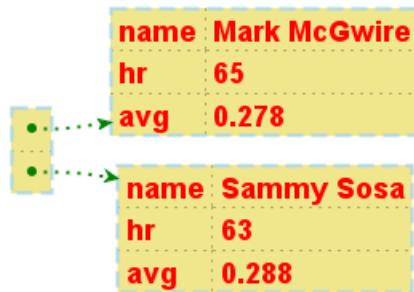
You can use style to change rendering of elements.

```

@startyaml
<style>
yamlDiagram {
  node {
    BackGroundColor lightblue
    LineColor lightblue
    FontName Helvetica
    FontColor red
    FontSize 18
    FontStyle bold
    BackGroundColor Khaki
    RoundCorner 0
    LineThickness 2
    LineStyle 10;5
    separator {
      LineThickness 0.5
      LineColor black
      LineStyle 1;5
    }
  }
  arrow {
    BackGroundColor lightblue
    LineColor green
    LineThickness 2
    LineStyle 2;5
  }
}
</style>
-
  name: Mark McGwire
  hr: 65
  avg: 0.278
-
  name: Sammy Sosa
  hr: 63
  avg: 0.288
@endyaml

```





[Ref. QA-13123]

## 13 Network diagram (nwdiag)

nwdiag has been created by Takeshi Komiya and allows to quickly draw network diagrams. So we thank him for his creation!

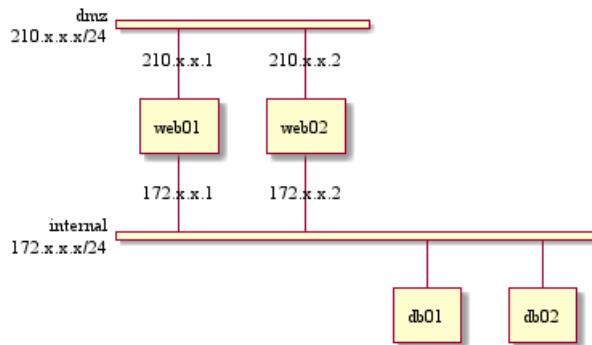
Since the syntax is clear and simple, this has been integrated within PlantUML. We reuse here the examples that Takeshi has documented.

### 13.1 Simple diagram

```
@startuml
nwdiag {
    network dmz {
        address = "210.x.x.x/24"

        web01 [address = "210.x.x.1"];
        web02 [address = "210.x.x.2"];
    }
    network internal {
        address = "172.x.x.x/24";

        web01 [address = "172.x.x.1"];
        web02 [address = "172.x.x.2"];
        db01;
        db02;
    }
}
@enduml
```



### 13.2 Define multiple addresses

```
@startuml
nwdiag {
    network dmz {
        address = "210.x.x.x/24

        // set multiple addresses (using comma)
        web01 [address = "210.x.x.1, 210.x.x.20"];
        web02 [address = "210.x.x.2"];
    }
    network internal {
        address = "172.x.x.x/24;

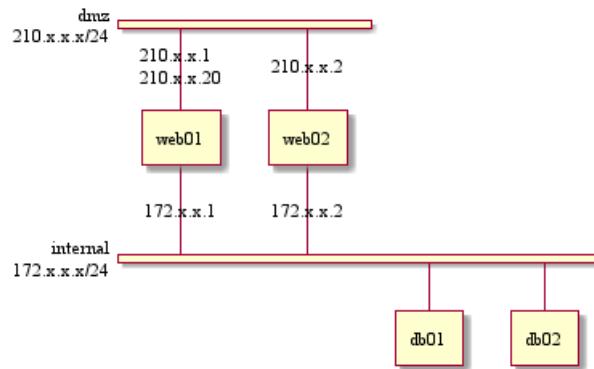
        web01 [address = "172.x.x.1"];
        web02 [address = "172.x.x.2"];
        db01;
    }
}
```



```

        db02;
    }
}
@enduml

```



## 13.3 Grouping nodes

### 13.3.1 Define group inside network definitions

```

@startuml
nwdiag {
    network Sample_front {
        address = "192.168.10.0/24";

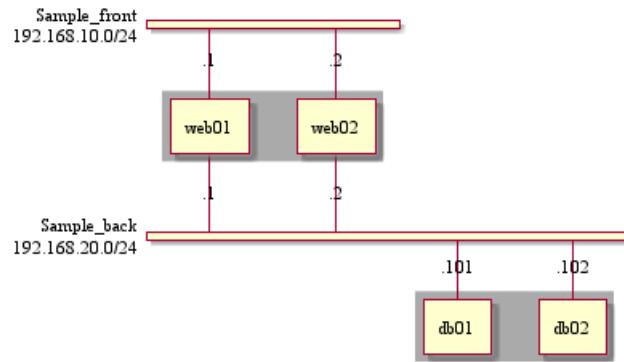
        // define group
        group web {
            web01 [address = ".1"];
            web02 [address = ".2"];
        }
    }

    network Sample_back {
        address = "192.168.20.0/24";
        web01 [address = ".1"];
        web02 [address = ".2"];
        db01 [address = ".101"];
        db02 [address = ".102"];

        // define network using defined nodes
        group db {
            db01;
            db02;
        }
    }
}
@enduml

```





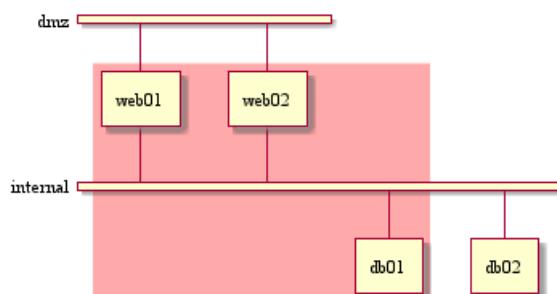
### 13.3.2 Define group outside of network definitions

```

@startuml
nwdiag {
    // define group outside of network definitions
    group {
        color = "#FFAAAA";
        web01;
        web02;
        db01;
    }

    network dmz {
        web01;
        web02;
    }
    network internal {
        web01;
        web02;
        db01;
        db02;
    }
}
@enduml

```



### 13.3.3 Define several groups on same network

#### 13.3.4 Example with 2 group

```

@startuml
nwdiag {
    group {

```



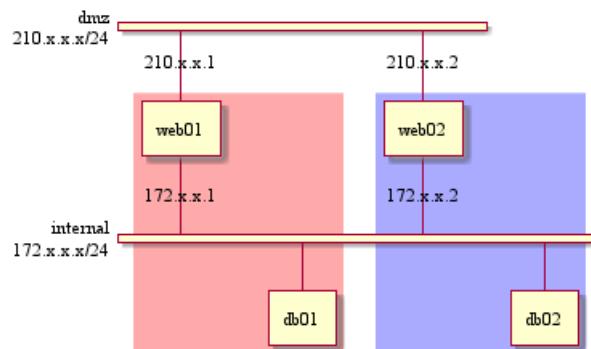
```

color = "#FFaaaa";
web01;
db01;
}
group {
color = "#aaaaFF";
web02;
db02;
}
network dmz {
address = "210.x.x.x/24"

web01 [address = "210.x.x.1"];
web02 [address = "210.x.x.2"];
}
network internal {
address = "172.x.x.x/24";

web01 [address = "172.x.x.1"];
web02 [address = "172.x.x.2"];
db01 ;
db02 ;
}
}
}
@enduml

```



[Ref. QA-12663]

### 13.3.5 Example with 3 groups

```

@startuml
nwdiag {
group {
color = "#FFaaaa";
web01;
db01;
}
group {
color = "#aaFFaa";
web02;
db02;
}
group {
color = "#aaaaFF";
web03;
db03;
}
}
}
@enduml

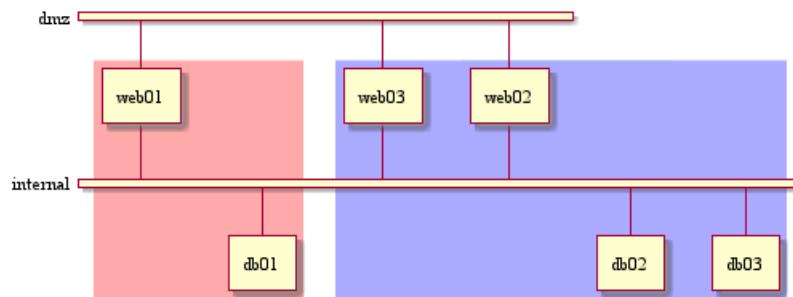
```



```

}
network dmz {
    web01;
    web02;
    web03;
}
network internal {
    web01;
    db01 ;
    web02;
    db02 ;
    web03;
    db03;
}
}
@enduml

```



[Ref. QA-13138]

## 13.4 Extended Syntax (for network or group)

### 13.4.1 Network

For network or network's component, you can add or change:

- addresses (*separated by comma ,*);
- color;
- description;
- shape.

```

@startuml
nwdiag {
    network Sample_front {
        address = "192.168.10.0/24"
        color = "red"

        // define group
        group web {
            web01 [address = ".1, .2", shape = "node"]
            web02 [address = ".2, .3"]
        }
    }
    network Sample_back {
        address = "192.168.20.0/24"
        color = "palegreen"
        web01 [address = ".1"]
    }
}

```



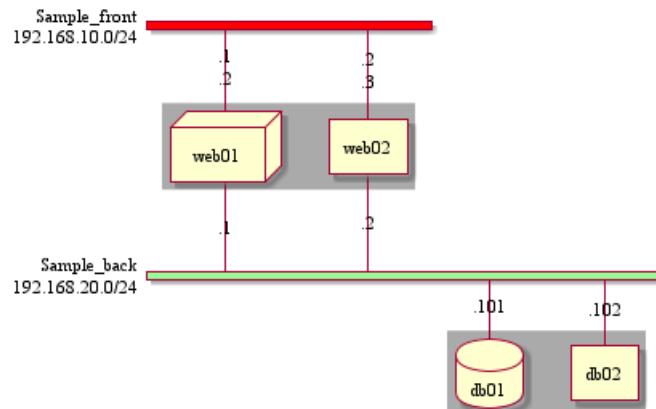
```

web02 [address = ".2"]
db01 [address = ".101", shape = database ]
db02 [address = ".102"]

// define network using defined nodes
group db {
    db01;
    db02;
}
}

@enduml

```



### 13.4.2 Group

For a group, you can add or change:

- color;
- description.

```

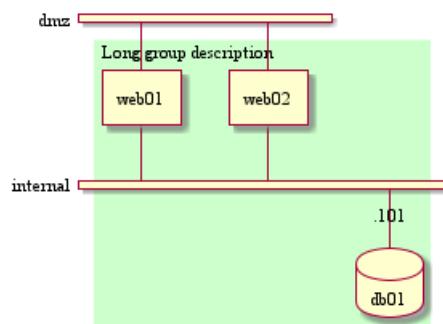
@startuml
nwdiag {
    group {
        color = "#CCFFCC";
        description = "Long group description";

        web01;
        web02;
        db01;
    }

    network dmz {
        web01;
        web02;
    }
    network internal {
        web01;
        web02;
        db01 [address = ".101", shape = database];
    }
}
@enduml

```





[Ref. QA-12056]

## 13.5 Using Sprites

You can use all sprites (icons) from the Standard Library or any other library.

Use the notation <\$sprite> to use a sprite, to make a new line, or any other Creole syntax.

```

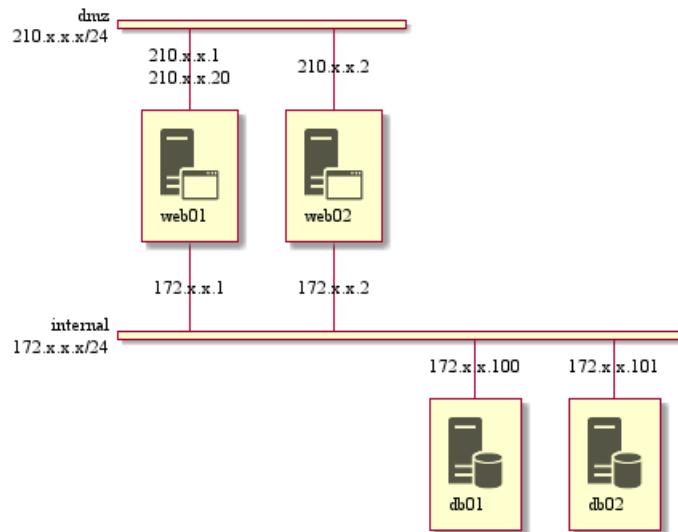
@startuml
!include <office/Servers/application_server>
!include <office/Servers/database_server>

nwdiag {
    network dmz {
        address = "210.x.x.x/24"

        // set multiple addresses (using comma)
        web01 [address = "210.x.x.1, 210.x.x.20", description = "<$application_server>\n web01"]
        web02 [address = "210.x.x.2", description = "<$application_server>\n web02"];
    }
    network internal {
        address = "172.x.x.x/24";

        web01 [address = "172.x.x.1"];
        web02 [address = "172.x.x.2"];
        db01 [address = "172.x.x.100", description = "<$database_server>\n db01"];
        db02 [address = "172.x.x.101", description = "<$database_server>\n db02"];
    }
}
@enduml

```



[Ref. QA-11862]

## 13.6 Using OpenIconic

You can also use the icons from OpenIconic in network or node descriptions.

Use the notation `<&icon>` to make an icon, `<&icon*n>` to multiply the size by a factor n, and `\n` to make a newline:

```

@startuml

nwdiag {
    group nightly {
        color = "#FFAAAA";
        description = "<&clock> Restarted nightly <&clock>";
        web02;
        db01;
    }
    network dmz {
        address = "210.x.x.x/24"

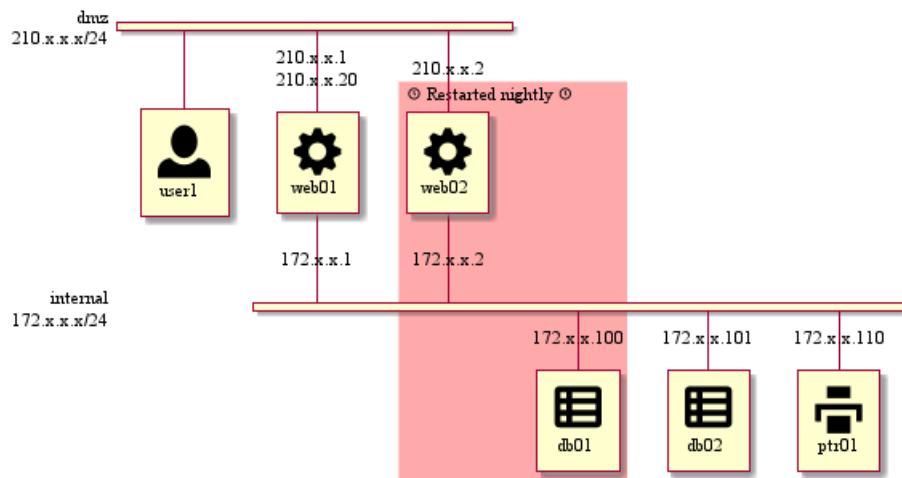
        user [description = "<&person*4.5>\n user1"];
        // set multiple addresses (using comma)
        web01 [address = "210.x.x.1, 210.x.x.20", description = "<&cog*4>\nweb01"]
        web02 [address = "210.x.x.2", description = "<&cog*4>\nweb02"];

    }
    network internal {
        address = "172.x.x.x/24";

        web01 [address = "172.x.x.1"];
        web02 [address = "172.x.x.2"];
        db01 [address = "172.x.x.100", description = "<&spreadsheet*4>\n db01"];
        db02 [address = "172.x.x.101", description = "<&spreadsheet*4>\n db02"];
        ptr [address = "172.x.x.110", description = "<&print*4>\n ptr01"];
    }
}
@enduml

```





### 13.7 Same nodes on more than two networks

You can use same nodes on different networks (more than two networks); *nwdiag* use in this case '*jump line*' over networks.

```

@startuml
nwdiag {
    // define group at outside network definitions
    group {
        color = "#7777FF";

        web01;
        web02;
        db01;
    }

    network dmz {
        color = "pink"

        web01;
        web02;
    }

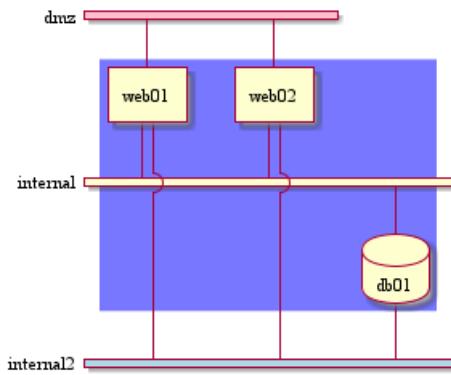
    network internal {
        web01;
        web02;
        db01 [shape = database ];
    }

    network internal2 {
        color = "LightBlue";

        web01;
        web02;
        db01;
    }
}
@enduml

```



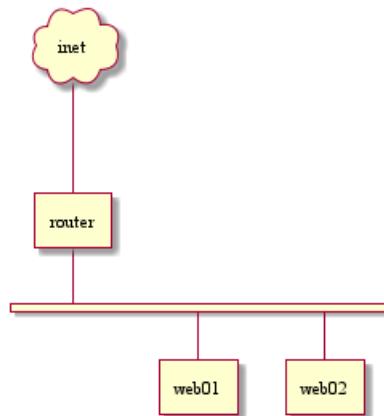


## 13.8 Peer networks

Peer networks are simple connections between two nodes, for which we don't use a horizontal "busbar" network

```
@startuml
nwdiag {
    inet [shape = cloud];
    inet -- router;

    network {
        router;
        web01;
        web02;
    }
}
@enduml
```



## 13.9 Peer networks and group

### 13.9.1 Without group

```
@startuml
nwdiag {
    internet [ shape = cloud];
    internet -- router;
    
```

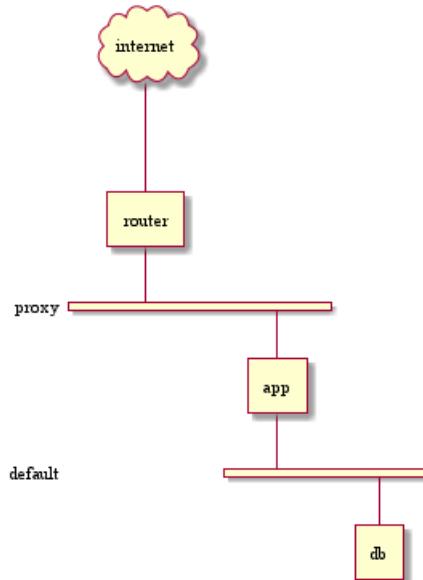


```

network proxy {
    router;
    app;
}
network default {
    app;
    db;
}
}

@enduml

```



### 13.9.2 Group on first

```

@startuml
nwdiag {
    internet [ shape = cloud];
    internet -- router;

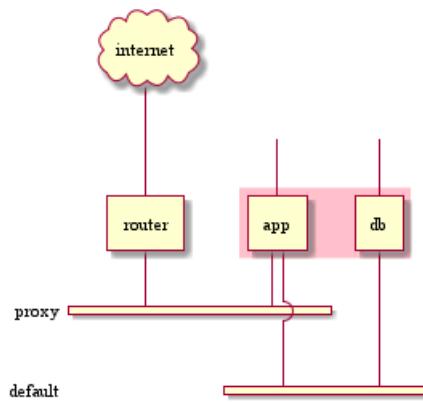
    group {
        color = "pink";
        app;
        db;
    }

    network proxy {
        router;
        app;
    }

    network default {
        app;
        db;
    }
}
@enduml

```

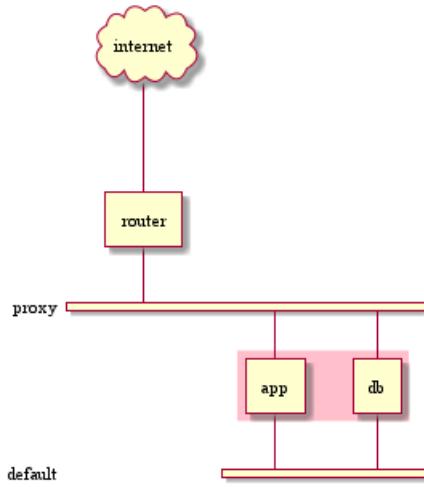




### 13.9.3 Group on second

```
@startuml  
nwdiag {  
    internet [ shape = cloud];  
    internet -- router;  
  
    network proxy {  
        router;  
        app;  
    }  
  
    group {  
        color = "pink";  
        app;  
        db;  
    }  
  
    network default {  
        app;  
        db;  
    }  
}  
@enduml
```





**TODO:** FIXME Why the line on proxy for 'db'? ('db' must be only on 'default network') [See example without group]

#### 13.9.4 Group on third

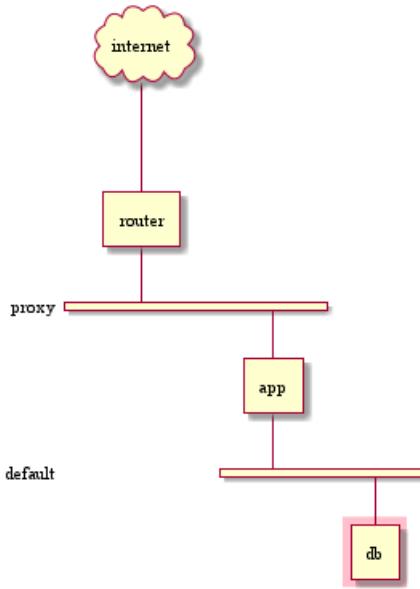
```

@startuml
nwdiag {
    internet [ shape = cloud ];
    internet -- router;

    network proxy {
        router;
        app;
    }
    network default {
        app;
        db;
    }
    group {
        color = "pink";
        app;
        db;
    }
}
@enduml

```





**TODO:** FIXME [Ref. Issue#408 and QA-12655] **TODO:** Not totally fixed

### 13.10 Add title, caption, header, footer or legend on network diagram

```
@startuml
```

```
header some header
```

```
footer some footer
```

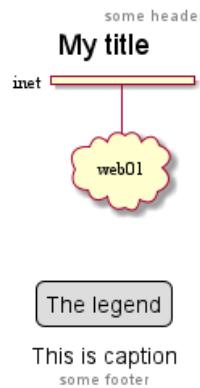
```
title My title
```

```
nwdiag {
    network inet {
        web01 [shape = cloud]
    }
}
```

```
legend
The legend
end legend
```

```
caption This is caption
@enduml
```





[Ref. QA-11303 and Common commands]

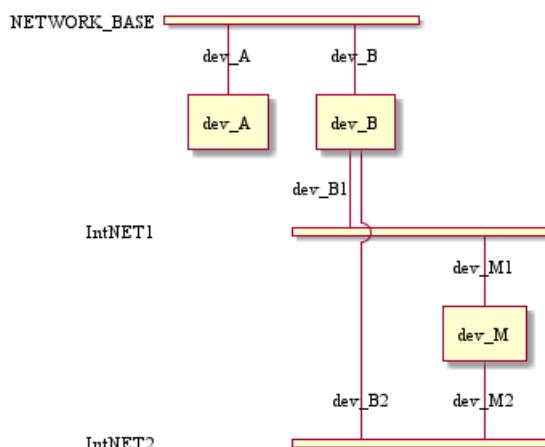
### 13.11 Change width of the networks

You can change the width of the networks, especially in order to have the same full width for only some or all networks.

Here are some examples, with all the possibilities:

- without

```
@startuml
nwdiag {
    network NETWORK_BASE {
        dev_A [address = "dev_A" ]
        dev_B [address = "dev_B" ]
    }
    network IntNET1 {
        dev_B [address = "dev_B1" ]
        dev_M [address = "dev_M1" ]
    }
    network IntNET2 {
        dev_B [address = "dev_B2" ]
        dev_M [address = "dev_M2" ]
    }
}
@enduml
```

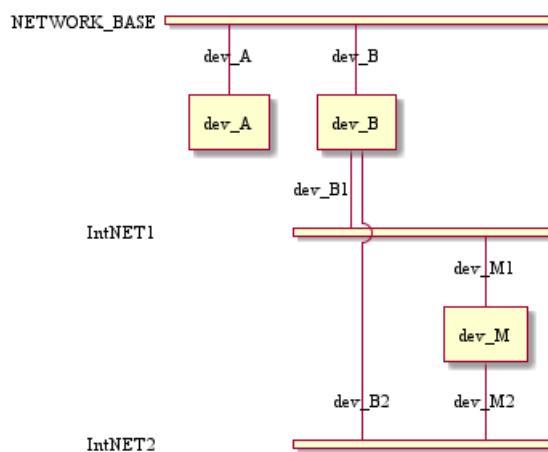


- only the first

```
@startuml
```



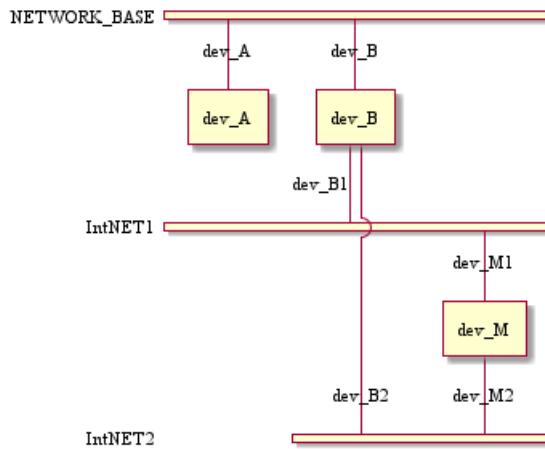
```
nwdiag {
    network NETWORK_BASE {
        width = full
        dev_A [address = "dev_A" ]
        dev_B [address = "dev_B" ]
    }
    network IntNET1 {
        dev_B [address = "dev_B1" ]
        dev_M [address = "dev_M1" ]
    }
    network IntNET2 {
        dev_B [address = "dev_B2" ]
        dev_M [address = "dev_M2" ]
    }
}
@enduml
```



- the first and the second

```
@startuml
nwdiag {
    network NETWORK_BASE {
        width = full
        dev_A [address = "dev_A" ]
        dev_B [address = "dev_B" ]
    }
    network IntNET1 {
        width = full
        dev_B [address = "dev_B1" ]
        dev_M [address = "dev_M1" ]
    }
    network IntNET2 {
        dev_B [address = "dev_B2" ]
        dev_M [address = "dev_M2" ]
    }
}
@enduml
```



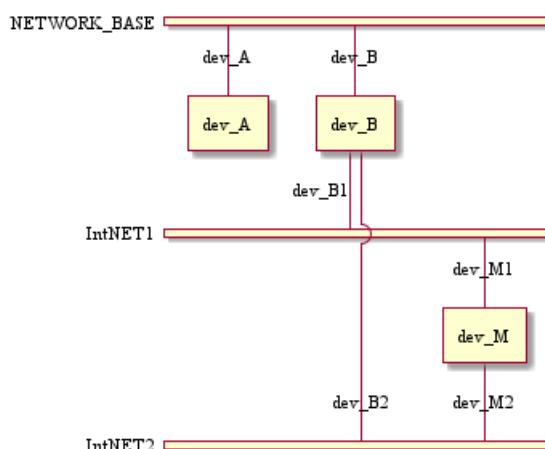


- all the network (with same full width)

```

@startuml
nwdiag {
    network NETWORK_BASE {
        width = full
        dev_A [address = "dev_A" ]
        dev_B [address = "dev_B" ]
    }
    network IntNET1 {
        width = full
        dev_B [address = "dev_B1" ]
        dev_M [address = "dev_M1" ]
    }
    network IntNET2 {
        width = full
        dev_B [address = "dev_B2" ]
        dev_M [address = "dev_M2" ]
    }
}
@enduml

```



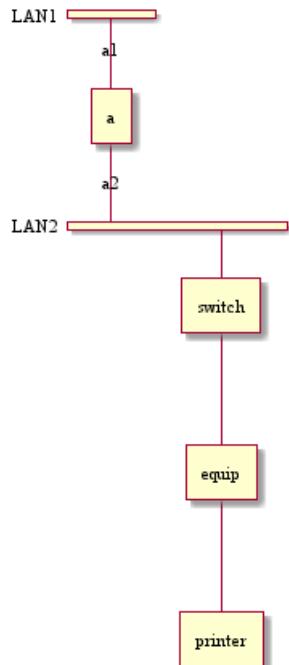
## 13.12 Other internal networks

You can define other internal networks (TCP/IP, USB, SERIAL,...).



- Without address or type

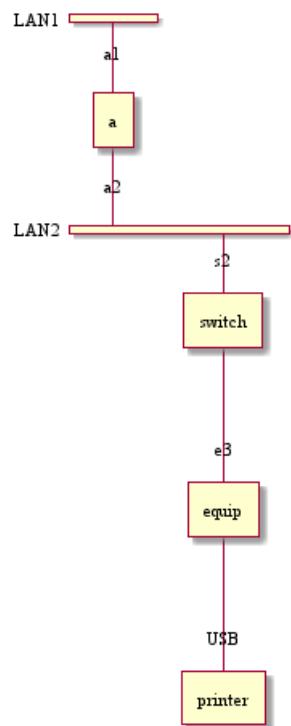
```
@startuml
nwdiag {
    network LAN1 {
        a [address = "a1"];
    }
    network LAN2 {
        a [address = "a2"];
        switch;
    }
    switch -- equip;
    equip -- printer;
}
@enduml
```



- With address or type

```
@startuml
nwdiag {
    network LAN1 {
        a [address = "a1"];
    }
    network LAN2 {
        a [address = "a2"];
        switch [address = "s2"];
    }
    switch -- equip;
    equip [address = "e3"];
    equip -- printer;
    printer [address = "USB"];
}
@enduml
```





[Ref. QA-12824]



## 14 Salt (Wireframe)

**Salt** is a subproject included in PlantUML that may help you to design graphical interface or *Website Wireframe or Page Schematic or Screen Blueprint*.

The goal of this tool is to discuss about simple and sample windows.

You can use either `@startsalt` keyword, or `@startuml` followed by a line with `salt` keyword.

### 14.1 Basic widgets

A window must start and end with brackets. You can then define:

- Button using [ and ].
- Radio button using ( and ).
- Checkbox using [ and ].
- User text area using ".
- Dropdown using ^.

```
@startsalt
{
    Just plain text
    [This is my button]
    () Unchecked radio
    (X) Checked radio
    [] Unchecked box
    [X] Checked box
    "Enter text here"
    ^This is a dropdown^
}
@endsalt
```



### 14.2 Using grid [[]]

A table is automatically created when you use an opening bracket {. And you have to use | to separate columns.

For example:

```
@startsalt
{
    Login | "MyName"
    Password | "****"
    [Cancel] | [ OK ]
}
@endsalt
```



Just after the opening bracket, you can use a character to define if you want to draw lines or columns of the grid :

Symbol	Result
#	To display all vertical and horizontal lines
!	To display all vertical lines
-	To display all horizontal lines
+	To display external lines

```
@startsalt
{+
    Login | "MyName"
    Password | "****"
    [Cancel] | [ OK ]
}
@endsalt
```

### 14.3 Group box [^]

```
@startsalt
{^"My group box"
    Login | "MyName"
    Password | "****"
    [Cancel] | [ OK ]
}
@endsalt
```

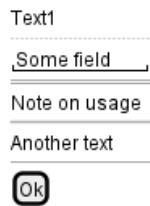
[Ref. QA-5840]

### 14.4 Using separator [.., ==, ~~, -]

You can use several horizontal lines as separator.

```
@startsalt
{
    Text1
    ..
    "Some field"
    ==
    Note on usage
    ~~
    Another text
    --
    [Ok]
}
@endsalt
```





## 14.5 Tree widget [T]

To have a Tree, you have to start with {T} and to use + to denote hierarchy.

```
@startsalt
{
{T
+ World
++ America
+++ Canada
+++ USA
++++ New York
++++ Boston
+++ Mexico
++ Europe
+++ Italy
+++ Germany
++++ Berlin
++ Africa
}
}
@endsalt
```



## 14.6 Tree table [T]

You can combine trees with tables.

```
@startsalt
{
{T
+Region | Population | Age
+ World | 7.13 billion | 30
++ America | 964 million | 30
+++ Canada | 35 million | 30
+++ USA | 319 million | 30
++++ NYC | 8 million | 30
++++ Boston | 617 thousand | 30
+++ Mexico | 117 million | 30
++ Europe | 601 million | 30
+++ Italy | 61 million | 30
+++ Germany | 82 million | 30
++++ Berlin | 3 million | 30
}
```



```

++ Africa      | 1 billion    | 30
}
}
}
@endsalt

```

Region	Population	Age
World	7.13 billion	30
America	964 million	30
Canada	35 million	30
USA	319 million	30
NYC	8 million	30
Boston	617 thousand	30
Mexico	117 million	30
Europe	601 million	30
Italy	61 million	30
Germany	82 million	30
Berlin	3 million	30
Africa	1 billion	30

And add lines.

```

@startsalt
{
..
== with T!
{T!
+Region      | Population    | Age
+ World      | 7.13 billion   | 30
++ America    | 964 million    | 30
}
..
== with T-
{T-
+Region      | Population    | Age
+ World      | 7.13 billion   | 30
++ America    | 964 million    | 30
}
..
== with T+
{T+
+Region      | Population    | Age
+ World      | 7.13 billion   | 30
++ America    | 964 million    | 30
}
..
== with T#
{T#
+Region      | Population    | Age
+ World      | 7.13 billion   | 30
++ America    | 964 million    | 30
}
..
}
@endsalt

```



**with T!**

Region	Population	Age
World	7.13 billion	30
America	964 million	30

---

**with T-**

Region	Population	Age
World	7.13 billion	30
America	964 million	30

---

**with T+**

Region	Population	Age
World	7.13 billion	30
America	964 million	30

---

**with T#**

Region	Population	Age
World	7.13 billion	30
America	964 million	30

[Ref. QA-1265]

## 14.7 Enclosing brackets [ , ]

You can define subelements by opening a new opening bracket.

```
@startsalt
{
Name      | "
Modifiers: | { (X) public | () default | () private | () protected
           | [] abstract | [] final   | [] static }
Superclass: | { "java.lang.Object" | [Browse...]
}
@endsalt
```

Name	<input type="text"/>
Modifiers:	<input checked="" type="radio"/> public <input type="radio"/> default <input type="radio"/> private <input type="radio"/> protected <input type="checkbox"/> abstract <input type="checkbox"/> final <input type="checkbox"/> static
Superclass:	<input type="text"/> <input type="button" value="Browse..."/>

## 14.8 Adding tabs [/]

You can add tabs using{/ notation. Note that you can use HTML code to have bold text.

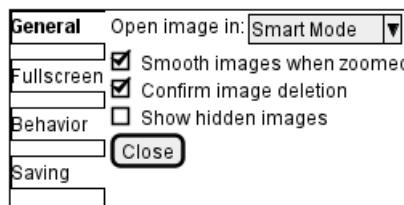
```
@startsalt
{+
{/ <b>General | Fullscreen | Behavior | Saving </b>
{
{ Open image in: | ^Smart Mode^ }
[X] Smooth images when zoomed
[X] Confirm image deletion
[ ] Show hidden images
}
[Close]
}
@endsalt
```

<b>General</b>	<input type="checkbox"/> Fullscreen	<input type="checkbox"/> Behavior	<input type="checkbox"/> Saving
Open image in:	<input type="text"/> Smart Mode <input type="button" value="▼"/>		
<input checked="" type="checkbox"/> Smooth images when zoomed			
<input checked="" type="checkbox"/> Confirm image deletion			
<input type="checkbox"/> Show hidden images			
<input type="button" value="Close"/>			



Tab could also be vertically oriented:

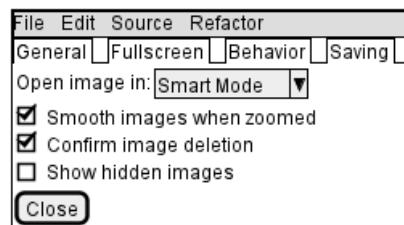
```
@startsalt
{+
{/ <b>General
Fullscreen
Behavior
Saving } |
{
{ Open image in: | ^Smart Mode^ }
[X] Smooth images when zoomed
[X] Confirm image deletion
[ ] Show hidden images
[Close]
}
}
@endsalt
```



## 14.9 Using menu [\*]

You can add a menu by using {\*} notation.

```
@startsalt
{+
{* File | Edit | Source | Refactor }
{/ General | Fullscreen | Behavior | Saving }
{
{ Open image in: | ^Smart Mode^ }
[X] Smooth images when zoomed
[X] Confirm image deletion
[ ] Show hidden images
}
[Close]
}
@endsalt
```

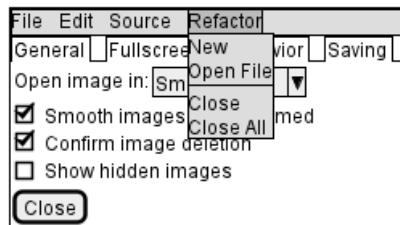


It is also possible to open a menu:

```
@startsalt
{+
{* File | Edit | Source | Refactor
Refactor | New | Open File | - | Close | Close All }
{/ General | Fullscreen | Behavior | Saving }
{
{ Open image in: | ^Smart Mode^ }
```



```
[X] Smooth images when zoomed
[X] Confirm image deletion
[ ] Show hidden images
}
[Close]
}
@endsalt
```



## 14.10 Advanced table

You can use two special notations for table :

- \* to indicate that a cell with span with left
- . to denote an empty cell

```
@startsalt
{#
. | Column 2 | Column 3
Row header 1 | value 1 | value 2
Row header 2 | A long cell | *
}
@endsalt
```

	Column 2	Column 3
Row header 1	value 1	value 2
Row header 2	A long cell	*

## 14.11 Scroll Bars [S, SI, S-]

You can use {S notation for scroll bar like in following examples:

- {S: for horizontal and vertical scrollbars

```
@startsalt
{S
Message
.
.
.
.
}
@endsalt
```



- {SI : for vertical scrollbar only

```
@startsalt
{SI
```



```
Message
```

```
.
.
.
}
@endsalt
```



- {S- : for horizontal scrollbar only

```
@startsalt
```

```
{S-
Message
```

```
.
.
.
}
@endsalt
```



## 14.12 Colors

It is possible to change text color of widget.

```
@startsalt
{
<color:Blue>Just plain text
[This is my default button]
[<color:green>This is my green button]
[<color:#9a9a9a>This is my disabled button]
[] <color:red>Unchecked box
[X] <color:green>Checked box
"Enter text here"
~This is a droplist~
^<color:#9a9a9a>This is a disabled droplist^
^<color:red>This is a red droplist^
}
```

```
@endsalt
```





[Ref. QA-12177]

### 14.13 Pseudo sprite [«, »]

Using << and >> you can define a pseudo-sprite or sprite-like drawing and reusing it latter.

```
@startsalt
{
[X] checkbox | [] checkbox
() radio | (X) radio
This is a text | [This is my button] | This is another text
"A field" | "Another long Field" | [A button]
<<folder
.....
.XXXX.....
.X...X.....
XXXXXXXXXX.
.X.....X.
.X.....X.
.X.....X.
.X.....X.
.XXXXXXXX.
.....
>>|<color:blue>other folder|<<folder>>
^DropList^
}
@endsalt
```



[Ref. QA-5849]

### 14.14 OpenIconic

OpenIconic is a very nice open source icon set. Those icons have been integrated into the creole parser, so you can use them out-of-the-box. You can use the following syntax: <&ICON\_NAME>.

```
@startsalt
{
Login<&person> | "MyName"
Password<&key> | "*****"
[Cancel <&circle-x>] | [OK <&account-login>]
}
```



@endsalt

Login	<input type="text" value="MyName"/>
Password	<input type="password" value="****"/>
<input type="button" value="Cancel"/>	<input type="button" value="OK"/>

The complete list is available on OpenIconic Website, or you can use the following special diagram:

```
@startuml
listopeniconic
@enduml
```

<b>List Open Iconic</b>	bell	cloud	excerpt	justify-right	musical-note	star
Credit to	bluetooth	cloudy	expand-down	key	paperclip	sun
<a href="https://useiconic.com/open">https://useiconic.com/open</a>	bold	code	expand-left	laptop	pencil	tablet
	bolt	cog	expand-right	layers	people	tag
- account-login	book	collapse-down	expand-up	lightbulb	person	tags
- account-logout	bookmark	collapse-left	external-link	link-broken	phone	target
- action-redo	box	collapse-right	eye	link-intact	pie-chart	task
- align-undo	briefcase	collapse-up	eyedropper	list-rich	pin	terminal
- align-center	british-pound	command	file	list	play-circle	text
- align-left	browser	comment-square	fire	location	plus	thumb-down
- align-right	brush	compass	flag	lock-locked	power-standby	thumb-up
+ aperture	bug	contrast	flash	lock-unlocked	print	timer
↓ arrow-bottom	bullhorn	copywriting	folder	loop-circular	project	transfer
↑ arrow-circle-bottom	calculator	credit-card	fork	loop-square	pulse	trash
○ arrow-circle-left	calendar	crop	fullscreen-enter	loop	puzzle-piece	underline
○ arrow-circle-right	camera-slr	dashboard	fullscreen-exit	magnifying-glass	question-mark	vertical-align-bottom
○ arrow-circle-top	caret-bottom	data-transfer-download	globe	map-marker	rain	vertical-align-center
↑ arrow-left	caret-left	data-transfer-upload	graph	map	random	vertical-align-top
→ arrow-right	caret-right	delete	grid-four-up	media-pause	reload	video
↓ arrow-thick-bottom	caret-top	dial	grid-three-up	media-play	resize-both	volume-high
← arrow-thick-left	cart	document	grid-two-up	media-record	resize-height	volume-low
→ arrow-thick-right	chat	dollar	hard-drive	media-skip-backward	resize-width	volume-off
↑ arrow-thick-top	check	double-quote-sans-left	header	media-skip-forward	rss-alt	warning
↑ arrow-top	chevron-bottom	double-quote-sans-right	headphones	media-step-backward	rss	wifi
↔ audio-spectrum	chevron-left	double-quote-serif-left	heart	media-step-forward	script	wrench
↔ audio	chevron-right	double-quote-serif-right	home	media-stop	share-boxed	x
● badge	chevron-top	droplet	image	medical-cross	share	yen
○ ban	circle-check	eject	inbox	menu	shield	zoom-in
■ bar-chart	circle-x	elevator	infinity	microphone	signal	zoom-out
▲ basket	clipboard	ellipses	info	minus	signpost	
□ battery-empty	clock	envelope-closed	italic	monitor	sort-ascending	
■ battery-full	cloud-download	envelope-open	justify-center	moon	sort-descending	
△ beaker	cloud-upload	euro	justify-left	move	spreadsheet	

## 14.15 Include Salt "on activity diagram"

You can read the following explanation.

```
@startuml
(*) --> "
{{ salt
{+
<b>an example
choose one option
()one
()two
[ok]
}
}}
" as choose

choose -right-> "
{{ salt
{+
<b>please wait
```

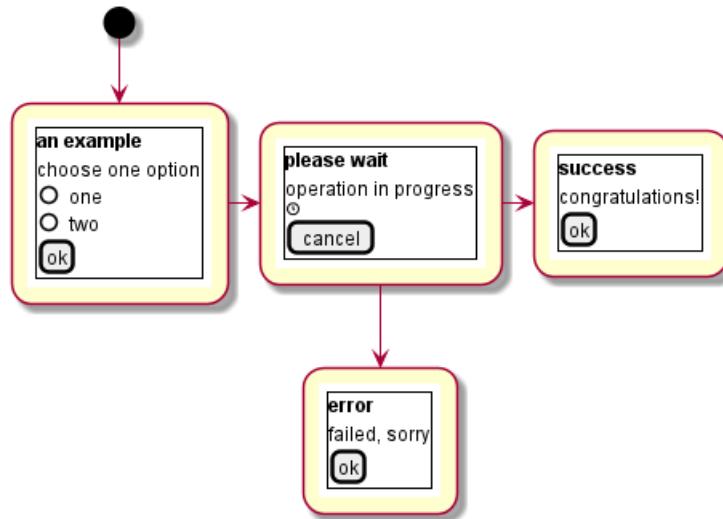


```

operation in progress
<&clock>
[cancel]
}
}}
" as wait
wait -right-> "
{{{
salt
{+
success
congratulations!
[ok]
}
}}
" as success

wait -down-> "
{{{
salt
{+
error
failed, sorry
[ok]
}
}}
"
@enduml

```



It can also be combined with define macro.

```

@startuml
!unquoted procedure SALT($x)
"{{{
salt
%invoke_procedure("_"+$x)
}}}" as $x
!endprocedure

!procedure _choose()
{+
<b>an example

```



```

choose one option
()one
()two
[ok]
}
!endprocedure

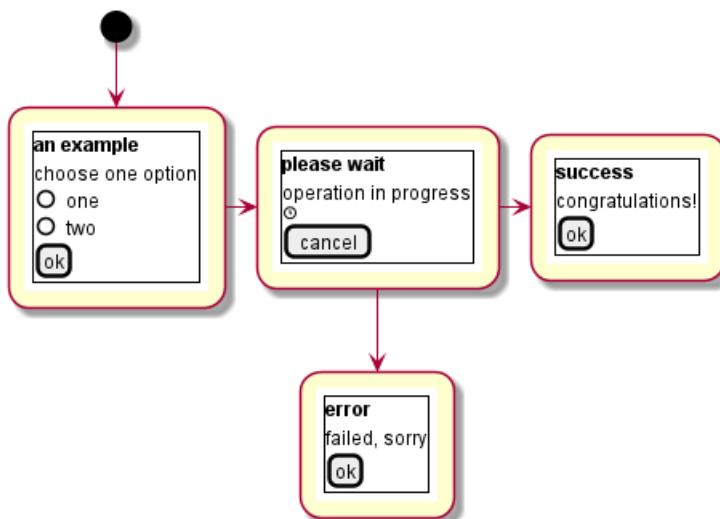
!procedure _wait()
{+
<b>please wait
operation in progress
<&clock>
[cancel]
}
!endprocedure

!procedure _success()
{+
<b>success
congratulations!
[ok]
}
!endprocedure

!procedure _error()
{+
<b>error
failed, sorry
[ok]
}
!endprocedure

(*) --> SALT(choose)
-right-> SALT(wait)
wait -right-> SALT(success)
wait -down-> SALT(error)
@enduml

```



## 14.16 Include salt "on while condition of activity diagram"

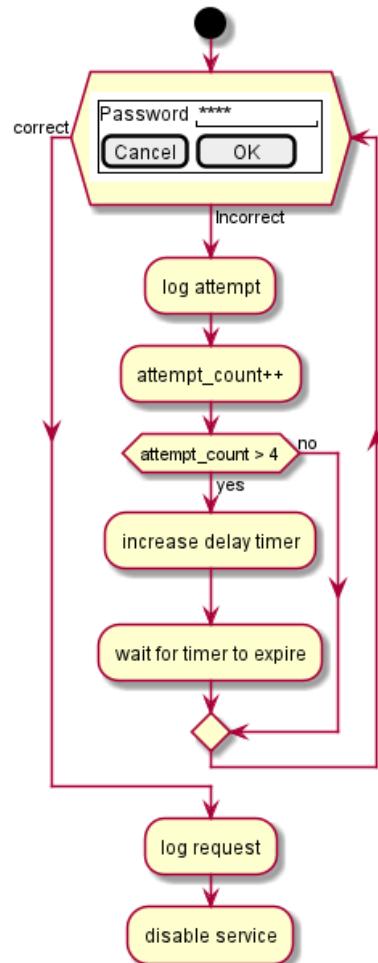
You can include salt on while condition of activity diagram.



```

@startuml
start
while (\n{\nsalt\n{+\nPassword | "****" "\n[Cancel] | [ OK ]}}\n) is (Incorrect)
    :log attempt;
    :attempt_count++;
    if (attempt_count > 4) then (yes)
        :increase delay timer;
        :wait for timer to expire;
    else (no)
        endif
    endwhile (correct)
    :log request;
    :disable service;
@enduml

```



[Ref. QA-8547]



## 15 Archimate Diagram

This is only a proposal and subject to change.

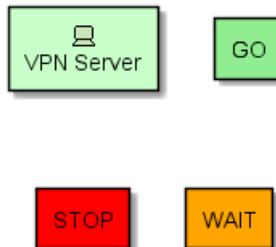
You are very welcome to create a new discussion on this future syntax. Your feedbacks, ideas and suggestions help us to find the right solution.

### 15.1 Archimate keyword

You can use the `archimate` keyword to define an element. Stereotype can optionally specify an additional icon. Some colors (Business, Application, Motivation, Strategy, Technology, Physical, Implementation) are also available.

```
@startuml
archimate #Technology "VPN Server" as vpnServerA <<technology-device>>

rectangle GO #lightgreen
rectangle STOP #red
rectangle WAIT #orange
@enduml
```



### 15.2 Defining Junctions

Using the `circle` keyword and the preprocessor, you can also create junctions.

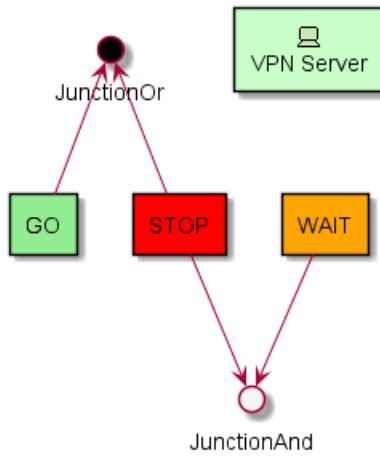
```
@startuml
!define Junction_Or circle #black
!define Junction_And circle #whitesmoke

Junction_And JunctionAnd
Junction_Or JunctionOr

archimate #Technology "VPN Server" as vpnServerA <<technology-device>>

rectangle GO #lightgreen
rectangle STOP #red
rectangle WAIT #orange
GO -up-> JunctionOr
STOP -up-> JunctionOr
STOP -down-> JunctionAnd
WAIT -down-> JunctionAnd
@enduml
```





### 15.3 Example 1

```

@startuml
skinparam rectangle<<behavior>> {
roundCorner 25
}
sprite $bProcess jar:archimate/business-process
sprite $aService jar:archimate/application-service
sprite $aComponent jar:archimate/application-component

rectangle "Handle claim" as HC <<$bProcess>><<behavior>> #Business
rectangle "Capture Information" as CI <<$bProcess>><<behavior>> #Business
rectangle "Notify\nAdditional Stakeholders" as NAS <<$bProcess>><<behavior>> #Business
rectangle "Validate" as V <<$bProcess>><<behavior>> #Business
rectangle "Investigate" as I <<$bProcess>><<behavior>> #Business
rectangle "Pay" as P <<$bProcess>><<behavior>> #Business

HC *--down- CI
HC *--down- NAS
HC *--down- V
HC *--down- I
HC *--down- P

CI -right->> NAS
NAS -right->> V
V -right->> I
I -right->> P

rectangle "Scanning" as scanning <<$aService>><<behavior>> #Application
rectangle "Customer admnistration" as customerAdministration <<$aService>><<behavior>> #Application
rectangle "Claims admnistration" as claimsAdministration <<$aService>><<behavior>> #Application
rectangle Printing <<$aService>><<behavior>> #Application
rectangle Payment <<$aService>><<behavior>> #Application

scanning -up-> CI
customerAdministration -up-> CI
claimsAdministration -up-> NAS
claimsAdministration -up-> V
claimsAdministration -up-> I
Payment -up-> P

Printing -up-> V
Printing -up-> P
  
```

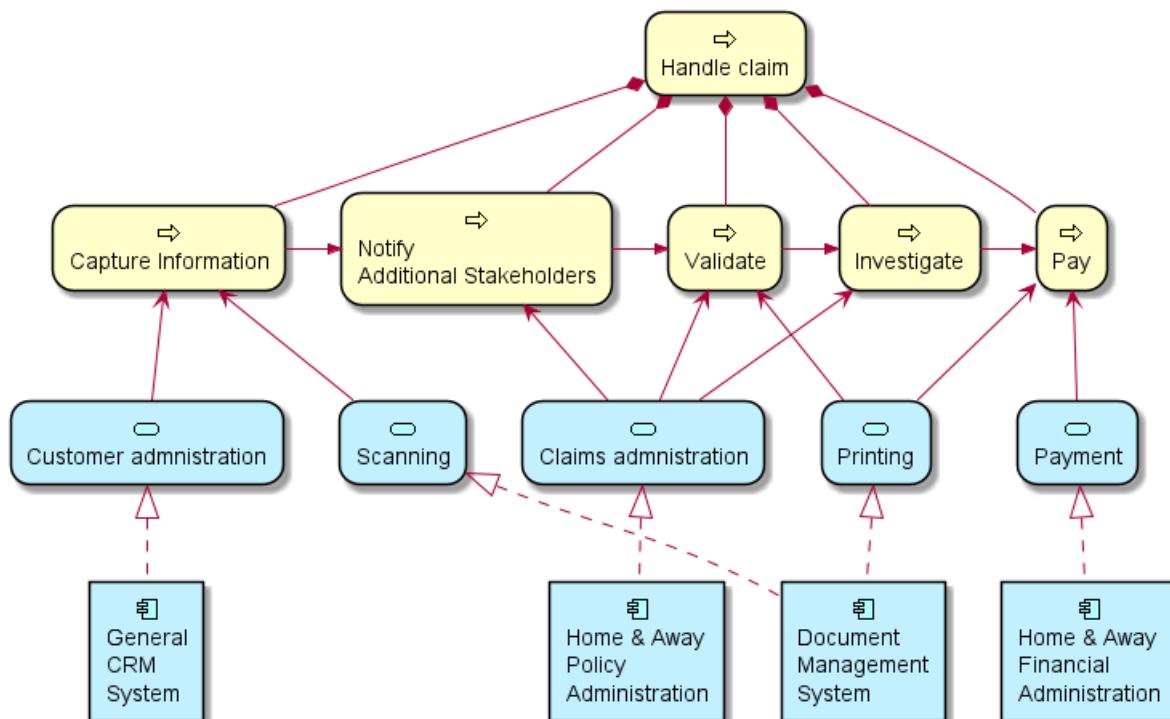
```

rectangle "Document\Management\System" as DMS <<$aComponent>> #Application
rectangle "General\CRM\System" as CRM <<$aComponent>> #Application
rectangle "Home & Away\Policy\Administration" as HAPA <<$aComponent>> #Application
rectangle "Home & Away\Financial\Administration" as HFPA <<$aComponent>> #Application

DMS .up.|> scanning
DMS .up.|> Printing
CRM .up.|> customerAdministration
HAPA .up.|> claimsAdministration
HFPA .up.|> Payment

legend left
Example from the "Archisurance case study" (OpenGroup).
See
=====
<$bProcess> :business process
=====
<$aService> : application service
=====
<$aComponent> : application component
endlegend
@enduml

```



Example from the "Archisurance case study" (OpenGroup).
See
⇒ :business process
□ : application service
■ : application component

## 15.4 Example 2

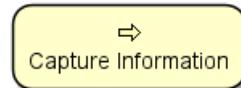
@startuml



```

skinparam roundcorner 25
rectangle "Capture Information" as CI <<$archimate/business-process>> #Business
@enduml

```



## 15.5 List possible sprites

You can list all possible sprites for Archimate using the following diagram:

```

@startuml
listsprite
@enduml

```

<b>List Current Sprites</b>	<b>business-object</b>	<b>interface-symmetric</b>	<b>service</b>
<i>Credit to</i> http://www.archimatetool.com	➡ business-process	○ interface	↗ serving
<b>archimate :</b>	■ business-product	► junction-and	↗ specialisation
	■ business-representation	► junction-or	↗ specialization
	○ business-role	● junction	● stakeholder-filled
	○ business-service	○ location	■ strategy-capability
	○ business-value	○ meaning	● strategy-course-of-action
	○ collaboration	○ motivation-assessment	■ strategy-resource
	○ communication-path	■ motivation-constraint	► strategy-value-stream
	○ component	● motivation-driver	○ system-software
	○ composition	● motivation-goal	■ technology-artifact
	■ constraint-filled	● motivation-meaning	● technology-collaboration
	■ constraint	● motivation-outcome	○ technology-communication-network
	■ deliverable-filled	● motivation-principle	► technology-communication-path
	■ deliverable	● motivation-requirement	■ technology-device
	○ device	● motivation-stakeholder	● technology-event
	○ driver-filled	● motivation-value	● technology-function
	○ driver	○ network	○ technology-infra-interface
	○ event	□ node	■ technology-infra-service
	○ flow	□ object	● technology-interaction
	○ function	↔ physical-distribution-network	● technology-interface
	○ gap-filled	○ physical-equipment	● technology-network
	○ gap	■ physical-facility	■ technology-node
	○ goal-filled	○ physical-material	► technology-path
	○ goal	─ plateau	○ technology-process
	○ implementation-deliverable	● principle-filled	■ technology-service
	○ implementation-event	● principle	● technology-system-software
	○ implementation-gap	► process	↗ triggering
	○ implementation-plateau	□ product	↗ used-by
	○ implementation-workpackage	○ realisation	○ value
	○ influence	○ representation	□ workpackage-filled
	○ interaction	○ requirement-filled	
	○ interface-required	○ requirement	
		○ role	

## 15.6 ArchiMate Macros

### 15.6.1 Archimate Macros and Library

A list of Archimate macros are defined Archimate-PlantUML here which simplifies the creation of ArchiMate diagrams, and Archimate is natively on the Standard Library of PlantUML.

### 15.6.2 Archimate elements

Using the macros, creation of ArchiMate elements are done using the following format: `Category_ElementName(nameOfThe "description")`

For example:

- To define a *Stakeholder* element, which is part of Motivation category, the syntax will be `Motivation_Stakeholder("Stakeholder Description")`:



```
@startuml
!include <archimate/Archimate>
Motivation_Stakeholder(StakeholderElement, "Stakeholder Description")
@enduml
```



- To define a *Business Service* element, `Business_Service(BService, "Business Service")`:

```
@startuml
!include <archimate/Archimate>
Business_Service(BService, "Business Service")
@enduml
```



### 15.6.3 Archimate relationships

The ArchiMate relationships are defined with the following pattern: `Rel_RelationType(fromElement, toElement, "description")` and to define the direction/orientation of the two elements: `Rel_RelationType_Direction(toElement, "description")`

The `RelationTypes` supported are:

- Access
- Aggregation
- Assignment
- Association
- Composition
- Flow
- Influence
- Realization
- Serving
- Specialization
- Triggering

The `Directions` supported are:

- Up
- Down
- Left
- Right

For example:

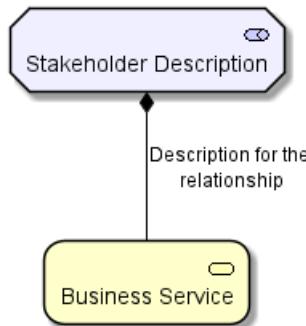
- To denote a composition relationship between the *Stakeholder* and *Business Service* defined above, the syntax will be

```
Rel_Composition(StakeholderElement, BService, "Description for the relationship")
```

```
@startuml
!include <archimate/Archimate>
Motivation_Stakeholder(StakeholderElement, "Stakeholder Description")
```

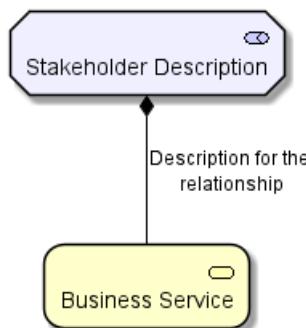


```
Business_Service(BService, "Business Service")
Rel_Composition(StakeholderElement, BService, "Description for the relationship")
@enduml
```



- Unordered List Item To orient the two elements in top - down position, the syntax will be

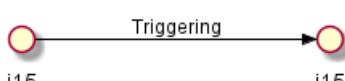
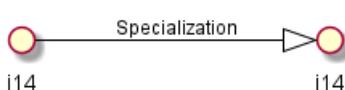
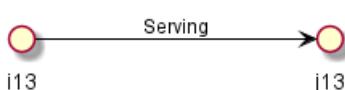
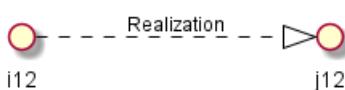
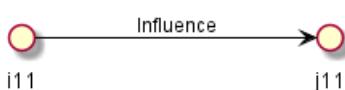
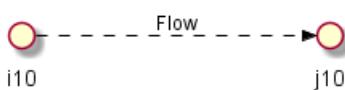
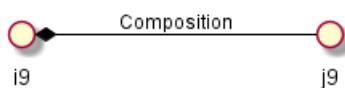
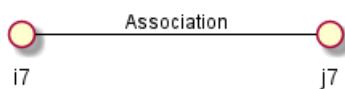
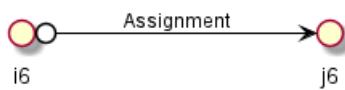
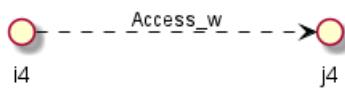
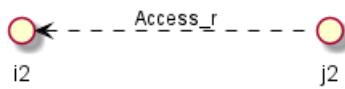
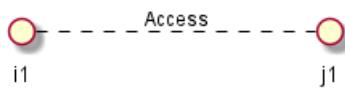
```
Rel_Composition_Down(StakeholderElement, BService, "Description for the relationship")
@startuml
!include <archimate/Archimate>
Motivation_Stakeholder(StakeholderElement, "Stakeholder Description")
Business_Service(BService, "Business Service")
Rel_Composition_Down(StakeholderElement, BService, "Description for the relationship")
@enduml
```



#### 15.6.4 Appendix: Examples of all Archimate RelationTypes

```
@startuml
left to right direction
!include <archimate/Archimate>
Rel_Triggering(i15, j15, Triggering)
Rel_Specialization(i14, j14, Specialization)
Rel_Serving(i13, j13, Serving)
Rel_Realization(i12, j12, Realization)
Rel_Influence(i11, j11, Influence)
Rel_Flow(i10, j10, Flow)
Rel_Composition(i9, j9, Composition)
'Rel_Association_dir(i8, j8, Association_dir)
Rel_Association(i7, j7, Association)
Rel_Assignment(i6, j6, Assignment)
Rel_Aggregation(i5, j5, Aggregation)
Rel_Access_w(i4, j4, Access_w)
Rel_Access_rw(i3, j3, Access_rw)
Rel_Access_r(i2, j2, Access_r)
Rel_Access(i1, j1, Access)
@enduml
```





## 16 Gantt Diagram

The Gantt is described in *natural* language, using very simple sentences (subject-verb-complement).

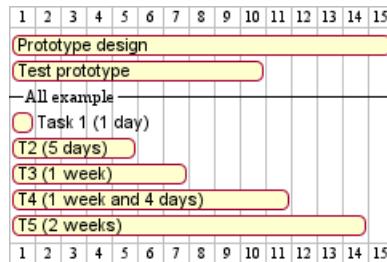
### 16.1 Declaring tasks

Tasks defined using square bracket.

#### 16.1.1 Duration

Their durations are defined using the `lasts` verb:

```
@startgantt
[Prototype design] lasts 15 days
[Test prototype] lasts 10 days
-- All example --
[Task 1 (1 day)] lasts 1 day
[T2 (5 days)] lasts 5 days
[T3 (1 week)] lasts 1 week
[T4 (1 week and 4 days)] lasts 1 week and 4 days
[T5 (2 weeks)] lasts 2 weeks
@endgantt
```

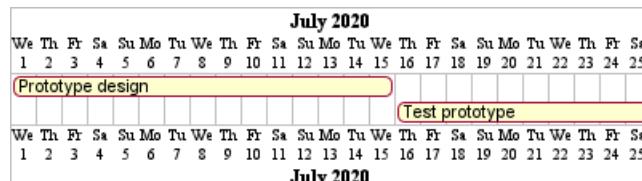


#### 16.1.2 Start

Their beginning are defined using the `start` verb:

```
@startuml
[Prototype design] lasts 15 days
[Test prototype] lasts 10 days
```

```
Project starts 2020-07-01
[Prototype design] starts 2020-07-01
[Test prototype] starts 2020-07-16
@enduml
```



#### 16.1.3 End

Their ending are defined using the `end` verb:

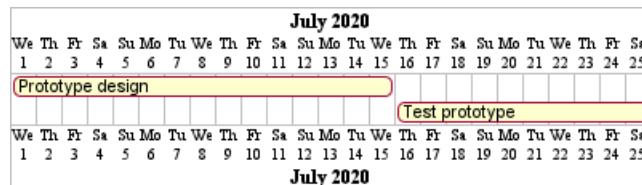
```
@startuml
[Prototype design] lasts 15 days
[Test prototype] lasts 10 days
```

```
Project starts 2020-07-01
[Prototype design] ends 2020-07-15
```



[Test prototype] ends 2020-07-25

@enduml



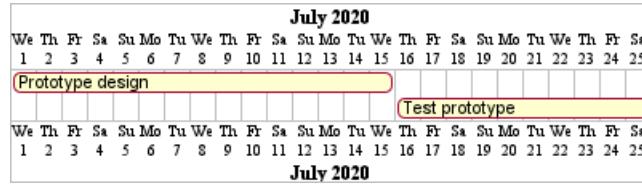
#### 16.1.4 Start/End

It is possible to define both absolutely, by specifying dates:

@startuml

```
Project starts 2020-07-01
[Prototype design] starts 2020-07-01
[Test prototype] starts 2020-07-16
[Prototype design] ends 2020-07-15
[Test prototype] ends 2020-07-25
```

@enduml

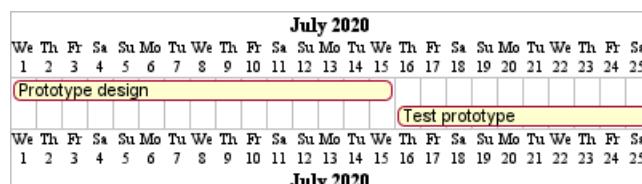


## 16.2 One-line declaration (with the and conjunction)

It is possible to combine declaration on one line with the `and` conjunction.

@startuml

```
Project starts 2020-07-01
[Prototype design] starts 2020-07-01 and ends 2020-07-15
[Test prototype] starts 2020-07-16 and lasts 10 days
@enduml
```



## 16.3 Adding constraints

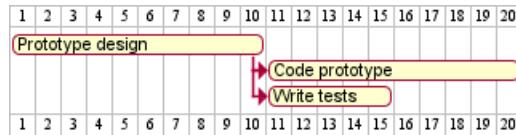
It is possible to add constraints between tasks.

@startgantt

```
[Prototype design] lasts 15 days
[Test prototype] lasts 10 days
[Test prototype] starts at [Prototype design]'s end
@endgantt
```



```
@startgantt
[Prototype design] lasts 10 days
[Code prototype] lasts 10 days
[Write tests] lasts 5 days
[Code prototype] starts at [Prototype design]'s end
[Write tests] starts at [Code prototype]'s start
@endgantt
```



## 16.4 Short names

It is possible to define short name for tasks with the `as` keyword.

```
@startgantt
[Prototype design] as [D] lasts 15 days
[Test prototype] as [T] lasts 10 days
[T] starts at [D]'s end
@endgantt
```



## 16.5 Customize colors

It is also possible to customize colors with `is colored in`.

```
@startgantt
[Prototype design] lasts 13 days
[Test prototype] lasts 4 days
[Test prototype] starts at [Prototype design]'s end
[Prototype design] is colored in Fuchsia/FireBrick
[Test prototype] is colored in GreenYellow/Green
@endgantt
```



## 16.6 Completion status

You can set the completion status of a task.

```
@startgantt
[foo] lasts 21 days
[foo] is 40% completed
[bar] lasts 30 days and is 10% complete
@endgantt
```



## 16.7 Milestone

You can define Milestones using the `happen` verb.



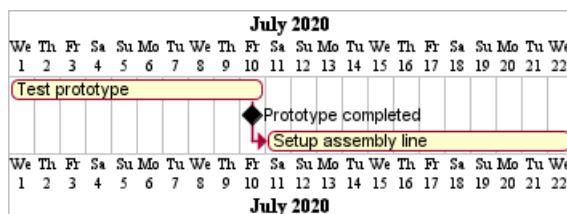
### 16.7.1 Relative milestone (use of constraints)

```
@startgantt
[Test prototype] lasts 10 days
[Prototype completed] happens at [Test prototype]'s end
[Setup assembly line] lasts 12 days
[Setup assembly line] starts at [Test prototype]'s end
@endgantt
```



### 16.7.2 Absolute milestone (use of fixed date)

```
@startgantt
Project starts 2020-07-01
[Test prototype] lasts 10 days
[Prototype completed] happens 2020-07-10
[Setup assembly line] lasts 12 days
[Setup assembly line] starts at [Test prototype]'s end
@endgantt
```



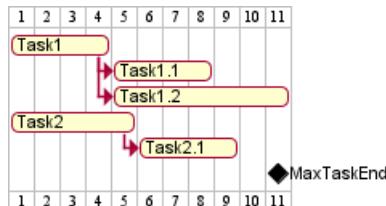
### 16.7.3 Milestone of maximum end of tasks

```
@startgantt
[Task1] lasts 4 days
then [Task1.1] lasts 4 days
[Task1.2] starts at [Task1]'s end and lasts 7 days

[Task2] lasts 5 days
then [Task2.1] lasts 4 days

[MaxTaskEnd] happens at [Task1.1]'s end
[MaxTaskEnd] happens at [Task1.2]'s end
[MaxTaskEnd] happens at [Task2.1]'s end

@endgantt
```



[Ref. QA-10764]

## 16.8 Hyperlinks

You can add hyperlinks to tasks.



```
@startgantt
[task1] lasts 10 days
[task1] links to [[http://plantuml.com]]
@endgantt
```



## 16.9 Calendar

You can specify a starting date for the whole project. By default, the first task starts at this date.

```
@startgantt
Project starts the 20th of september 2017
[Prototype design] as [TASK1] lasts 13 days
[TASK1] is colored in Lavender/LightBlue
@endgantt
```



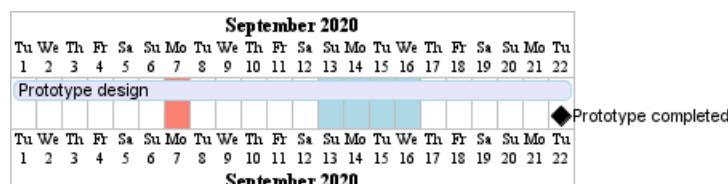
## 16.10 Coloring days

It is possible to add colors to some days.

```
@startgantt
Project starts the 2020/09/01
```

2020/09/07 is colored in salmon  
2020/09/13 to 2020/09/16 are colored in lightblue

```
[Prototype design] as [TASK1] lasts 22 days
[TASK1] is colored in Lavender/LightBlue
[Prototype completed] happens at [TASK1]'s end
@endgantt
```



## 16.11 Changing scale

You can change scale for very long project, with one of those parameters:

- printscale
- ganttscale
- projectscale

and one of the values:

- daily (*by default*)
- weekly
- monthly

(See QA-11272, QA-9041 and QA-10948)



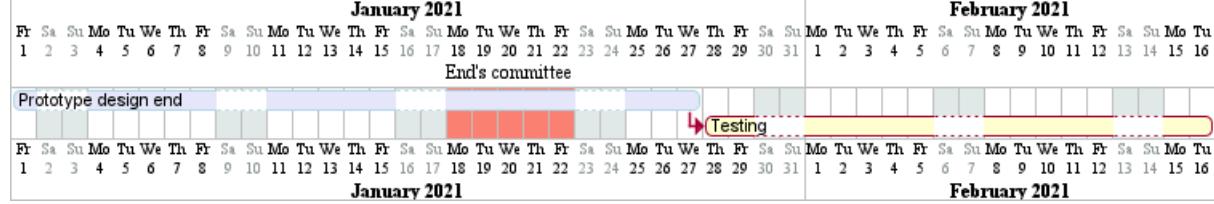
### 16.11.1 Daily (by default)

```
@startuml
saturday are closed
sunday are closed
```

Project starts the 1st of january 2021  
 [Prototype design end] as [TASK1] lasts 19 days  
 [TASK1] is colored in Lavender/LightBlue  
 [Testing] lasts 14 days  
 [TASK1]->[Testing]

2021-01-18 to 2021-01-22 are named [End's committee]  
 2021-01-18 to 2021-01-22 are colored in salmon

```
@enduml
```



### 16.11.2 Weekly

```
@startuml
printscale weekly
saturday are closed
sunday are closed
```

Project starts the 1st of january 2021  
 [Prototype design end] as [TASK1] lasts 19 days  
 [TASK1] is colored in Lavender/LightBlue  
 [Testing] lasts 14 days  
 [TASK1]->[Testing]

2021-01-18 to 2021-01-22 are named [End's committee]  
 2021-01-18 to 2021-01-22 are colored in salmon

```
@enduml
```

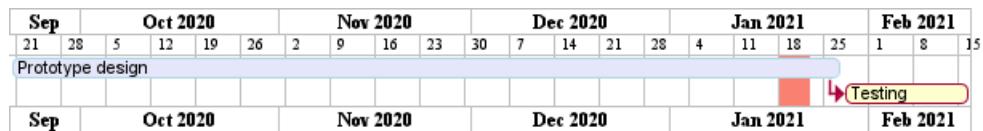


```
@startgantt
printscale weekly
Project starts the 20th of september 2020
[Prototype design] as [TASK1] lasts 130 days
[TASK1] is colored in Lavender/LightBlue
[Testing] lasts 20 days
[TASK1]->[Testing]
```

2021-01-18 to 2021-01-22 are named [End's committee]  
 2021-01-18 to 2021-01-22 are colored in salmon

```
@endgantt
```





### 16.11.3 Monthly

```
@startgantt
projectscale monthly
Project starts the 20th of september 2020
[Prototype design] as [TASK1] lasts 130 days
[TASK1] is colored in Lavender/LightBlue
[Testing] lasts 20 days
[TASK1]->[Testing]
```

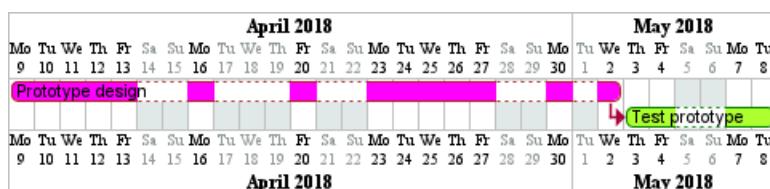
2021-01-18 to 2021-01-22 are named [End's committee]  
 2021-01-18 to 2021-01-22 are colored in salmon  
 @endgantt



### 16.12 Close day

It is possible to close some day.

```
@startgantt
project starts the 2018/04/09
saturday are closed
sunday are closed
2018/05/01 is closed
2018/04/17 to 2018/04/19 is closed
[Prototype design] lasts 14 days
[Test prototype] lasts 4 days
[Test prototype] starts at [Prototype design]'s end
[Prototype design] is colored in Fuchsia/FireBrick
[Test prototype] is colored in GreenYellow/Green
@endgantt
```

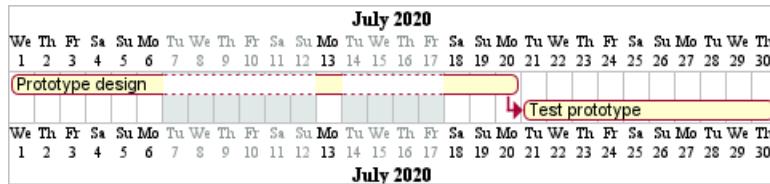


Then it is possible to open some closed day.

```
@startgantt
2020-07-07 to 2020-07-17 is closed
2020-07-13 is open
```

Project starts the 2020-07-01  
 [Prototype design] lasts 10 days  
 Then [Test prototype] lasts 10 days  
 @endgantt

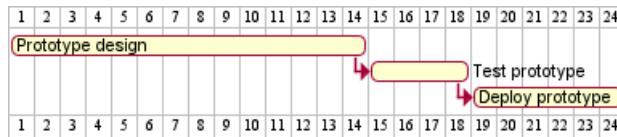




### 16.13 Simplified task succession

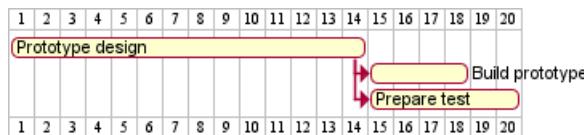
It's possible to use the `then` keyword to denote consecutive tasks.

```
@startgantt
[Prototype design] lasts 14 days
then [Test prototype] lasts 4 days
then [Deploy prototype] lasts 6 days
@endgantt
```



You can also use arrow `->`

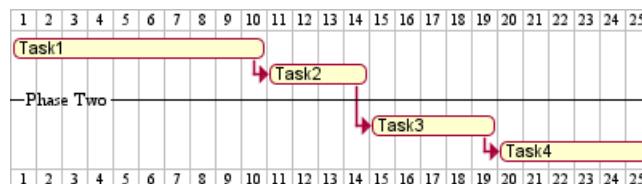
```
@startgantt
[Prototype design] lasts 14 days
[Build prototype] lasts 4 days
[Prepare test] lasts 6 days
[Prototype design] -> [Build prototype]
[Prototype design] -> [Prepare test]
@endgantt
```



### 16.14 Separator

You can use `--` to separate sets of tasks.

```
@startgantt
[Task1] lasts 10 days
then [Task2] lasts 4 days
-- Phase Two --
then [Task3] lasts 5 days
then [Task4] lasts 6 days
@endgantt
```



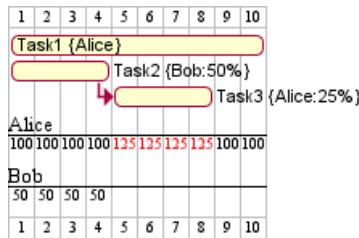
### 16.15 Working with resources

You can affect tasks on resources using the `on` keyword and brackets for resource name.

```
@startgantt
[Task1] on {Alice} lasts 10 days
```

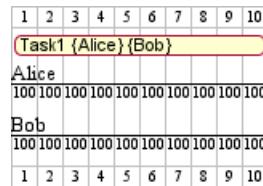


```
[Task2] on {Bob:50%} lasts 2 days
then [Task3] on {Alice:25%} lasts 1 days
@endgantt
```



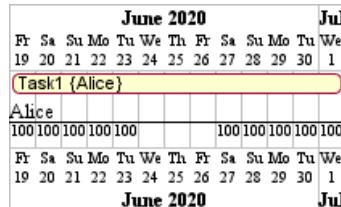
Multiple resources can be assigned to a task:

```
@startgantt
[Task1] on {Alice} {Bob} lasts 20 days
@endgantt
```



Resources can be marked as off on specific days:

```
@startgantt
project starts on 2020-06-19
[Task1] on {Alice} lasts 10 days
{Alice} is off on 2020-06-24 to 2020-06-26
@endgantt
```

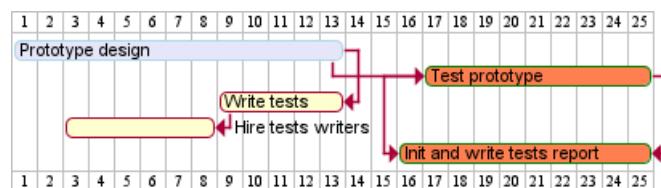


## 16.16 Complex example

It is also possible to use the `and` conjunction.

You can also add delays in constraints.

```
@startgantt
[Prototype design] lasts 13 days and is colored in Lavender/LightBlue
[Test prototype] lasts 9 days and is colored in Coral/Green and starts 3 days after [Prototype design]
[Write tests] lasts 5 days and ends at [Prototype design]'s end
[Hire tests writers] lasts 6 days and ends at [Write tests]'s start
[Init and write tests report] is colored in Coral/Green
[Init and write tests report] starts 1 day before [Test prototype]'s start and ends at [Test prototype]
@endgantt
```



## 16.17 Comments

As is mentioned on Common Commands page: blockquote Everything that starts with simple quote ' is a comment.

You can also put comments on several lines using '/' to start and '/ to end. blockquote (*i.e.: the first character (except space character) of a comment line must be a simple quote '*)

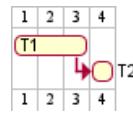
```
@startgantt
' This is a comment
```

```
[T1] lasts 3 days
```

```
/' this comment
is on several lines '/
```

```
[T2] starts at [T1]'s end and lasts 1 day
```

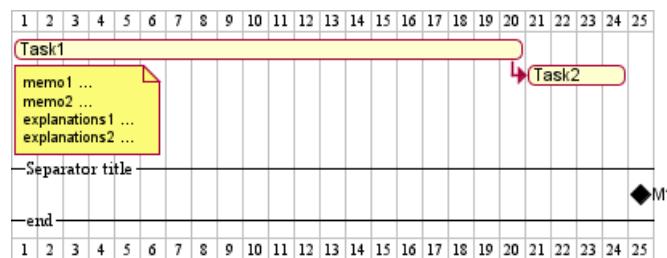
```
@endgantt
```



## 16.18 Using style

### 16.18.1 Without style (by default)

```
@startuml
[Task1] lasts 20 days
note bottom
    memo1 ...
    memo2 ...
    explanations1 ...
    explanations2 ...
end note
[Task2] lasts 4 days
[Task1] -> [Task2]
-- Separator title --
[M1] happens on 5 days after [Task1]'s end
-- end --
@enduml
```



### 16.18.2 With style

You can use style to change rendering of elements.

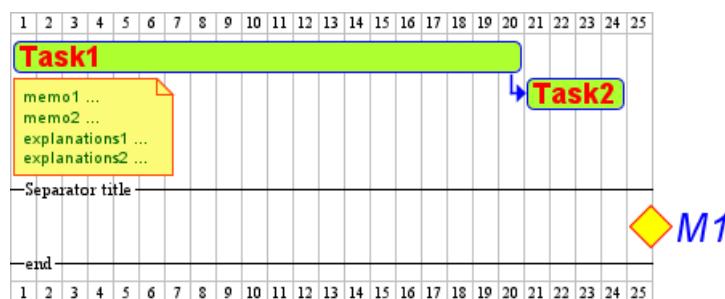
```
@startuml
<style>
ganttDiagram {
task {
FontName Helvetica
FontColor red
FontSize 18
}}
```



```

FontStyle bold
BackGroundColor GreenYellow
LineColor blue
}
}
 milestone {
FontColor blue
FontSize 25
FontStyle italic
BackGroundColor yellow
LineColor red
}
note {
FontColor DarkGreen
FontSize 10
LineColor OrangeRed
}
arrow {
FontName Helvetica
FontColor red
FontSize 18
FontStyle bold
BackGroundColor GreenYellow
LineColor blue
}
separator {
LineColor red
BackGroundColor green
FontSize 16
FontStyle bold
FontColor purple
}
}
</style>
[Task1] lasts 20 days
note bottom
memo1 ...
memo2 ...
explanations1 ...
explanations2 ...
end note
[Task2] lasts 4 days
[Task1] -> [Task2]
-- Separator title --
[M1] happens on 5 days after [Task1]'s end
-- end --
@enduml

```



[Ref. QA-10835, QA-12045, QA-11877 and PR-438]

**TODO:** TODO Awaiting style for Separator and all style for Arrow (thickness...)

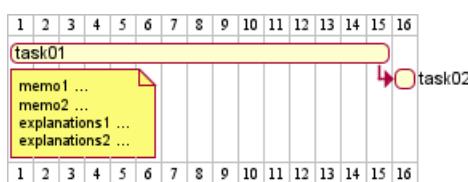


## 16.19 Add notes

```
@startgantt
[task01] lasts 15 days
note bottom
    memo1 ...
    memo2 ...
    explanations1 ...
    explanations2 ...
end note
```

[task01] -> [task02]

@endgantt

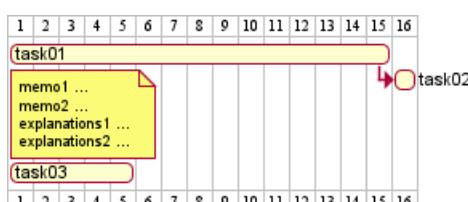


Example with overlap.

```
@startgantt
[task01] lasts 15 days
note bottom
    memo1 ...
    memo2 ...
    explanations1 ...
    explanations2 ...
end note
```

[task01] -> [task02]
[task03] lasts 5 days

@endgantt



@startgantt

-- test01 --

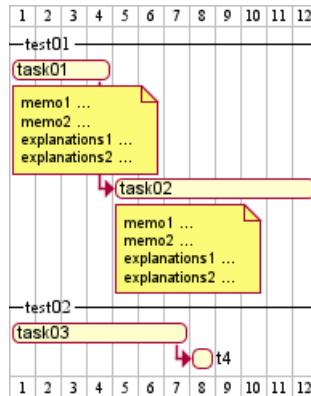
```
[task01] lasts 4 days
note bottom
'note left
memo1 ...
memo2 ...
explanations1 ...
explanations2 ...
end note
```

```
[task02] lasts 8 days
[task01] -> [task02]
note bottom
```



```
'note left
memo1 ...
memo2 ...
explanations1 ...
explanations2 ...
end note
-- test02 --

[task03] as [t3] lasts 7 days
[t3] -> [t4]
@endgantt
```



**TODO:** DONE *Thanks for correction (of #386 on v1.2020.18) when overlapping*

@startgantt

Project starts 2020-09-01

```
[taskA] starts 2020-09-01 and lasts 3 days
[taskB] starts 2020-09-10 and lasts 3 days
[taskB] displays on same row as [taskA]
```

[task01] starts 2020-09-05 and lasts 4 days

```
then [task02] lasts 8 days
note bottom
    note for task02
    more notes
end note
```

```
then [task03] lasts 7 days
note bottom
    note for task03
    more notes
end note
```

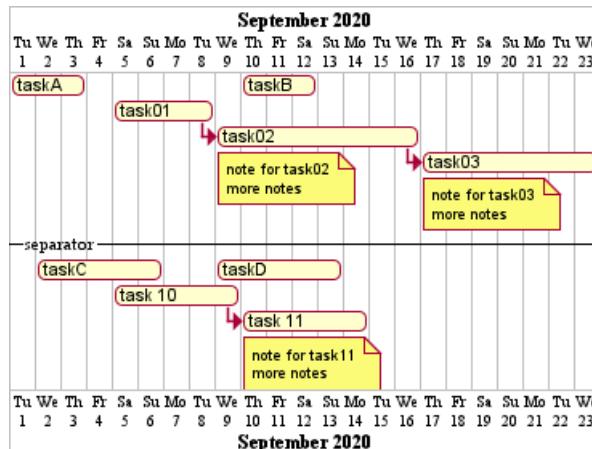
-- separator --

```
[taskC] starts 2020-09-02 and lasts 5 days
[taskD] starts 2020-09-09 and lasts 5 days
[taskD] displays on same row as [taskC]
```

```
[task 10] starts 2020-09-05 and lasts 5 days
then [task 11] lasts 5 days
note bottom
    note for task11
```

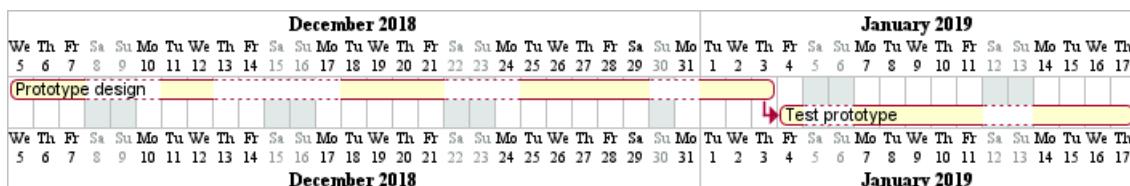


```
more notes
end note
@endgantt
```



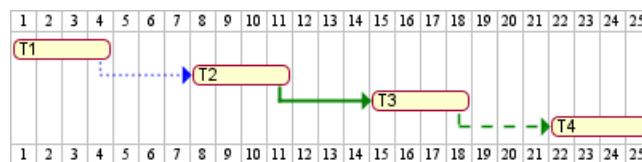
## 16.20 Pause tasks

```
@startgantt
Project starts the 5th of december 2018
saturday are closed
sunday are closed
2018/12/29 is opened
[Prototype design] lasts 17 days
[Prototype design] pauses on 2018/12/13
[Prototype design] pauses on 2018/12/14
[Prototype design] pauses on monday
[Test prototype] starts at [Prototype design]'s end and lasts 2 weeks
@endgantt
```



## 16.21 Change link colors

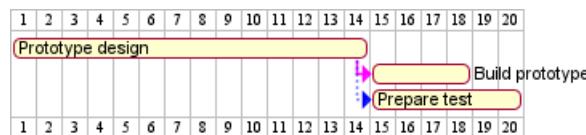
```
@startgantt
[T1] lasts 4 days
[T2] lasts 4 days and starts 3 days after [T1]'s end with blue dotted link
[T3] lasts 4 days and starts 3 days after [T2]'s end with green bold link
[T4] lasts 4 days and starts 3 days after [T3]'s end with green dashed link
@endgantt
```



```
@startuml
Links are colored in blue
[Prototype design] lasts 14 days
[Build prototype] lasts 4 days
[Prepare test] lasts 6 days
```

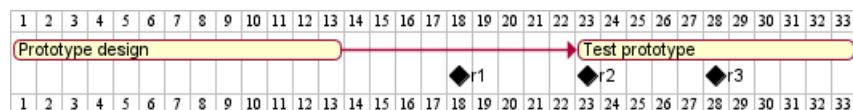


```
[Prototype design] -[#FF00FF]-> [Build prototype]
[Prototype design] -[dotted]-> [Prepare test]
@enduml
```



## 16.22 Tasks or Milestones on the same line

```
@startgantt
[Prototype design] lasts 13 days
[Test prototype] lasts 4 days and 1 week
[Test prototype] starts 1 week and 2 days after [Prototype design]'s end
[Test prototype] displays on same row as [Prototype design]
[r1] happens on 5 days after [Prototype design]'s end
[r2] happens on 5 days after [r1]'s end
[r3] happens on 5 days after [r2]'s end
[r2] displays on same row as [r1]
[r3] displays on same row as [r1]
@endgantt
```

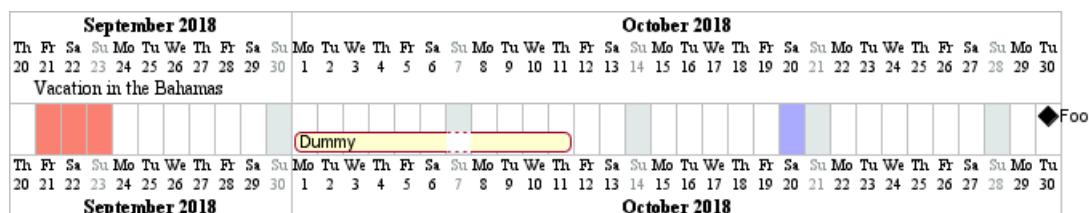


## 16.23 Highlight today

```
@startgantt
Project starts the 20th of september 2018
sunday are close
2018/09/21 to 2018/09/23 are colored in salmon
2018/09/21 to 2018/09/30 are named [Vacation in the Bahamas]
```

```
today is 30 days after start and is colored in #AAF
[Foo] happens 40 days after start
[Dummy] lasts 10 days and starts 10 days after start
```

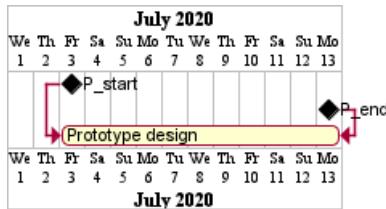
```
@endgantt
```



## 16.24 Task between two milestones

```
@startgantt
project starts on 2020-07-01
[P_start] happens 2020-07-03
[P_end] happens 2020-07-13
[Prototype design] occurs from [P_start] to [P_end]
@endgantt
```





## 16.25 Grammar and verbal form

Verbal form	Example
[T] starts	
[M] happens	

## 16.26 Add title, header, footer, caption or legend on gantt diagram

```
@startuml
```

```
header some header
```

```
footer some footer
```

```
title My title
```

```
[Prototype design] lasts 13 days
```

```
legend
```

```
The legend
```

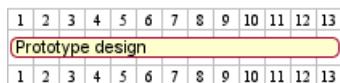
```
end legend
```

```
caption This is caption
```

```
@enduml
```

some header

**My title**



This is caption

some footer

(See also: Common commands)

## 16.27 Removing Foot Boxes

You can use the `hide footbox` keywords to remove the foot boxes of the gantt diagram (*as for sequence diagram*).

Examples on:

- daily scale (*without project start*)

```
@startgantt
```

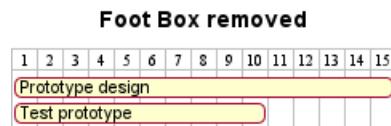
```
hide footbox
```

```
title Foot Box removed
```

```
[Prototype design] lasts 15 days  
[Test prototype] lasts 10 days
```



```
@endgantt
```



- daily scale

```
@startgantt
```

Project starts the 20th of september 2017  
 [Prototype design] as [TASK1] lasts 13 days  
 [TASK1] is colored in Lavender/LightBlue

```
hide footbox
```

```
@endgantt
```



- weekly scale

```
@startgantt
```

```
hide footbox
```

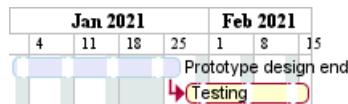
printscale weekly  
 saturday are closed  
 sunday are closed

Project starts the 1st of january 2021  
 [Prototype design end] as [TASK1] lasts 19 days  
 [TASK1] is colored in Lavender/LightBlue  
 [Testing] lasts 14 days  
 [TASK1]->[Testing]

2021-01-18 to 2021-01-22 are named [End's committee]

2021-01-18 to 2021-01-22 are colored in salmon

```
@endgantt
```



- monthly scale

```
@startgantt
```

```
hide footbox
```

projectscale monthly  
 Project starts the 20th of september 2020  
 [Prototype design] as [TASK1] lasts 130 days  
 [TASK1] is colored in Lavender/LightBlue  
 [Testing] lasts 20 days  
 [TASK1]->[Testing]

2021-01-18 to 2021-01-22 are named [End's committee]

2021-01-18 to 2021-01-22 are colored in salmon



```
@endumt
```



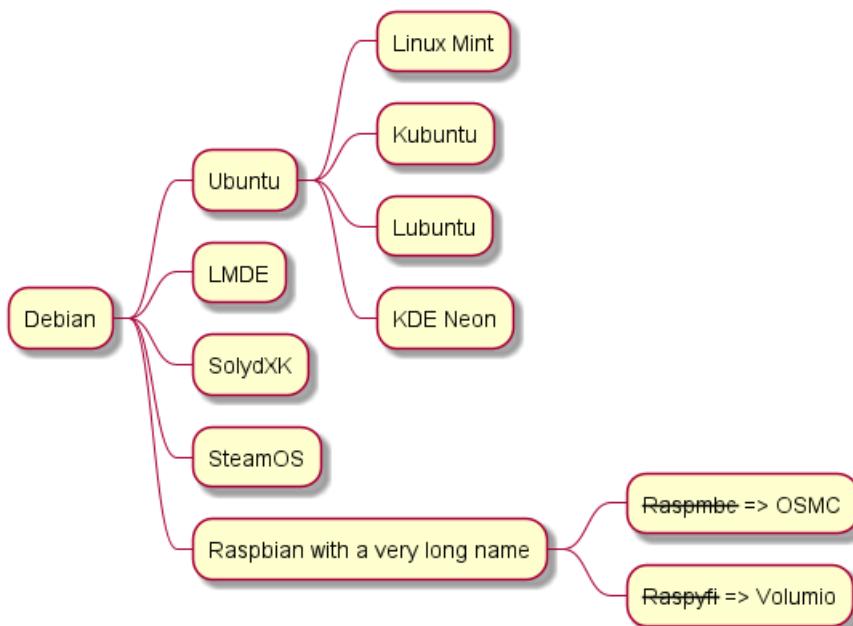
## 17 MindMap

MindMap diagram are still in beta: the syntax may change without notice.

### 17.1 OrgMode syntax

This syntax is compatible with OrgMode

```
@startmindmap
* Debian
** Ubuntu
*** Linux Mint
*** Kubuntu
*** Lubuntu
*** KDE Neon
** LMDE
** SolydXK
** SteamOS
** Raspbian with a very long name
*** <s>Raspmbc</s> => OSMC
*** <s>Raspyfi</s> => Volumio
@endmindmap
```

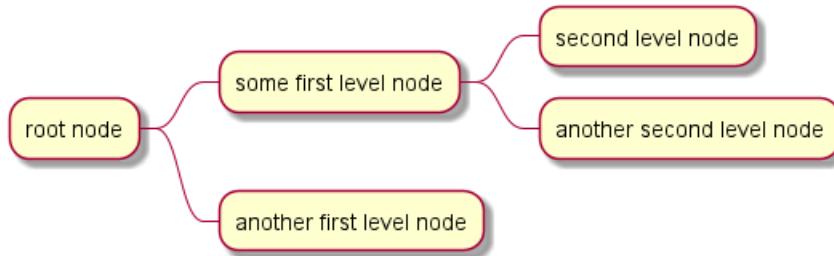


### 17.2 Markdown syntax

This syntax is compatible with Markdown

```
@startmindmap
* root node
* some first level node
* second level node
* another second level node
* another first level node
@endmindmap
```



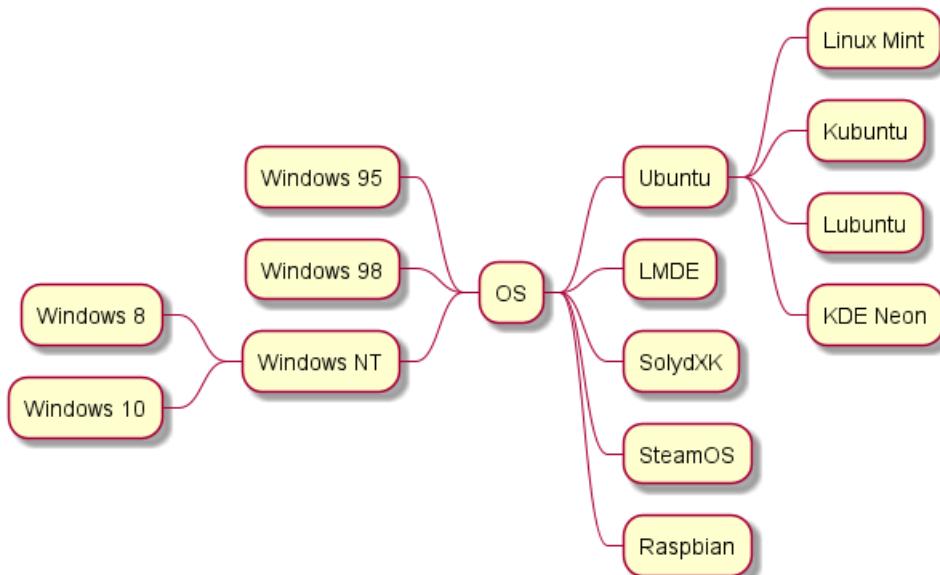


### 17.3 Arithmetic notation

You can use the following notation to choose diagram side.

```

@startmindmap
+ OS
++ Ubuntu
+++ Linux Mint
+++ Kubuntu
+++ Lubuntu
+++ KDE Neon
++ LMDE
++ SolydXK
++ SteamOS
++ Raspbian
-- Windows 95
-- Windows 98
-- Windows NT
--- Windows 8
--- Windows 10
@endmindmap
  
```



### 17.4 Multilines

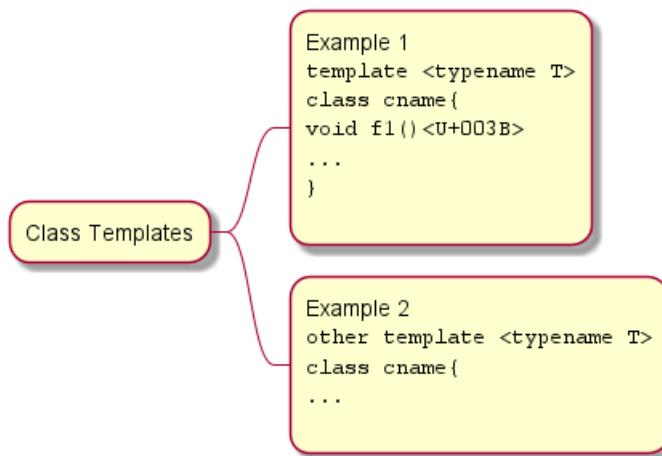
You can use : and ; to have multilines box.

```

@startmindmap
* Class Templates
  
```



```
**:Example 1
<code>
template <typename T>
class cname{
void f1()<U+003B>
...
}
</code>
;
**:Example 2
<code>
other template <typename T>
class cname{
...
</code>
;
@endmindmap
```



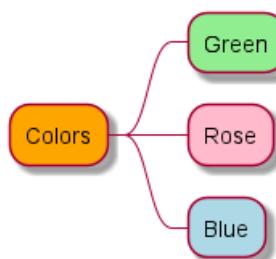
## 17.5 Colors

It is possible to change node color.

### 17.5.1 With inline color

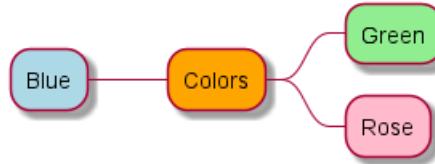
- OrgMode syntax mindmap

```
@startmindmap
*[#Orange] Colors
**[#lightgreen] Green
**[#FFBBCC] Rose
**[#lightblue] Blue
@endmindmap
```



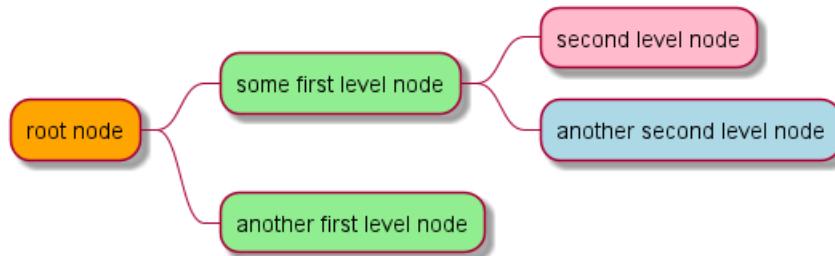
- Arithmetic notation syntax mindmap

```
@startmindmap
+[#Orange] Colors
++[#lightgreen] Green
++[#FFBBCC] Rose
--[#lightblue] Blue
@endmindmap
```



- Markdown syntax mindmap

```
@startmindmap
*[#Orange] root node
*[#lightgreen] some first level node
*[#FFBBCC] second level node
*[#lightblue] another second level node
*[#lightgreen] another first level node
@endmindmap
```

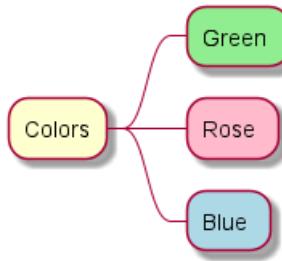


### 17.5.2 With style color

- OrgMode syntax mindmap

```
@startmindmap
<style>
mindmapDiagram {
  .green {
    BackgroundColor lightgreen
  }
  .rose {
    BackgroundColor #FFBBCC
  }
  .your_style_name {
    BackgroundColor lightblue
  }
}
</style>
* Colors
** Green <>green<>
** Rose <>rose<>
** Blue <>your_style_name<>
@endmindmap
```

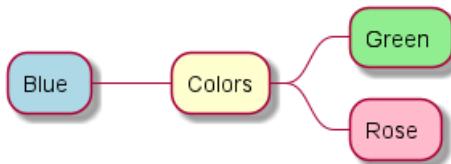




- Arithmetic notation syntax mindmap

```

@startmindmap
<style>
mindmapDiagram {
    .green {
        BackgroundColor lightgreen
    }
    .rose {
        BackgroundColor #FFBBCC
    }
    .your_style_name {
        BackgroundColor lightblue
    }
}
</style>
+ Colors
++ Green <<green>>
++ Rose <<rose>>
-- Blue <<your_style_name>>
@endmindmap
  
```



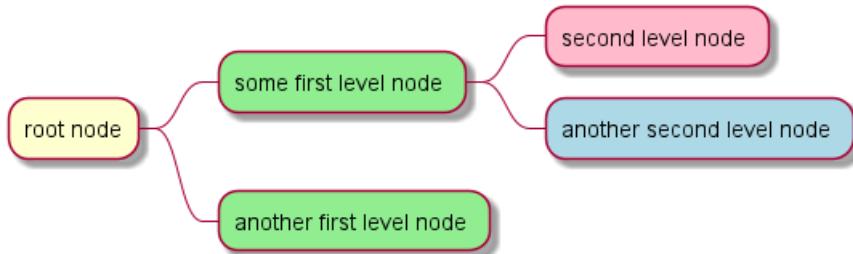
- Markdown syntax mindmap

```

@startmindmap
<style>
mindmapDiagram {
    .green {
        BackgroundColor lightgreen
    }
    .rose {
        BackgroundColor #FFBBCC
    }
    .your_style_name {
        BackgroundColor lightblue
    }
}
</style>
* root node
* some first level node <<green>>
  * second level node <<rose>>
    * another second level node <<your_style_name>>
  
```



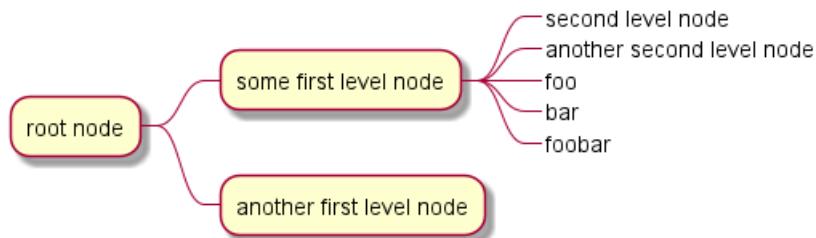
```
* another first level node <>green>>
@endmindmap
```



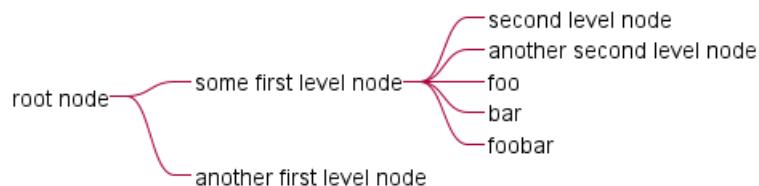
## 17.6 Removing box

You can remove the box drawing using an underscore.

```
@startmindmap
* root node
** some first level node
***_ second level node
***_ another second level node
***_ foo
***_ bar
***_ foobar
** another first level node
@endmindmap
```



```
@startmindmap
*_ root node
**_ some first level node
***_ second level node
***_ another second level node
***_ foo
***_ bar
***_ foobar
**_ another first level node
@endmindmap
```



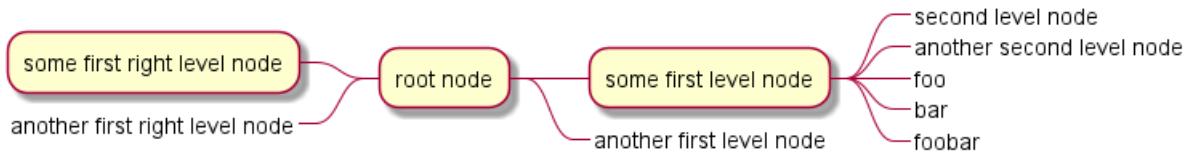
```
@startmindmap
+ root node
++ some first level node
```



```

+++_ second level node
+++_ another second level node
+++_ foo
+++_ bar
+++_ foobar
++_ another first level node
-- some first right level node
-- another first right level node
@endmindmap

```



## 17.7 Changing diagram direction

It is possible to use both sides of the diagram.

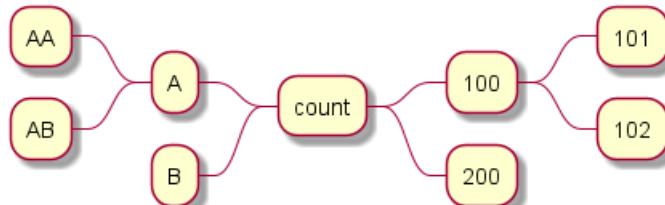
```

@startmindmap
* count
** 100
*** 101
*** 102
** 200

left side

** A
*** AA
*** AB
** B
@endmindmap

```



## 17.8 Complete example

```

@startmindmap
caption figure 1
title My super title

* <&flag>Debian
** <&globe>Ubuntu
*** Linux Mint
*** Kubuntu
*** Lubuntu
*** KDE Neon
** <&graph>LMDE
** <&pulse>SolydXK

```



```
** <&people>SteamOS
** <&star>Raspbian with a very long name
*** <s>Raspmbc</s> => OSMC
*** <s>Raspyfi</s> => Volumio
```

```
header
My super header
endheader

center footer My super footer
```

```
legend right
Short
legend
endlegend
@endmindmap
```

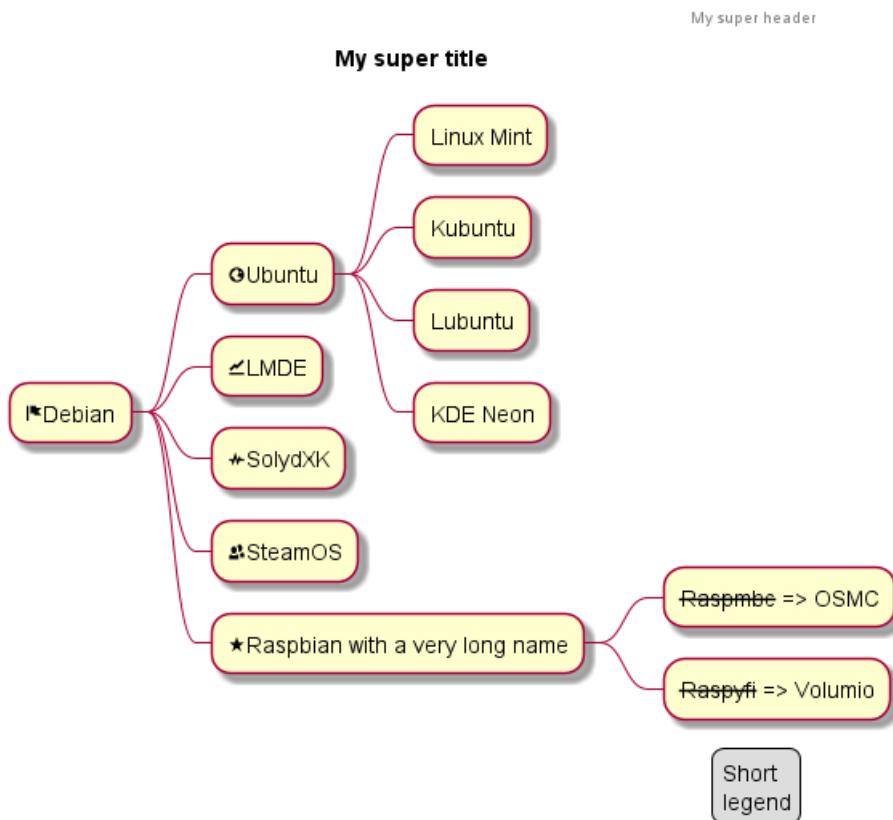


figure 1  
My super footer

## 17.9 Changing style

### 17.9.1 node, depth

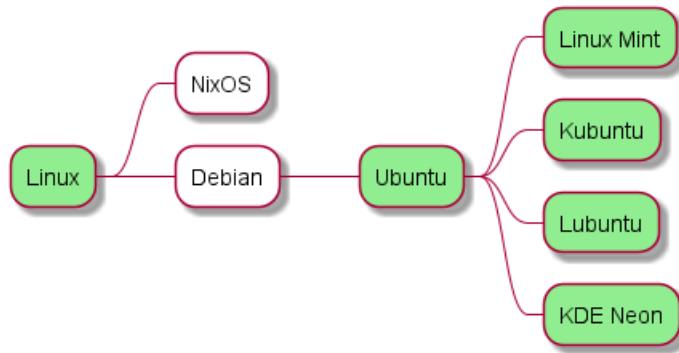
```
@startmindmap
<style>
mindmapDiagram {
    node {
        BackgroundColor lightGreen
    }
    :depth(1) {
        BackGroundColor white
    }
}
```



```

    }
}
</style>
* Linux
** NixOS
** Debian
*** Ubuntu
**** Linux Mint
**** Kubuntu
**** Lubuntu
**** KDE Neon
@endmindmap

```



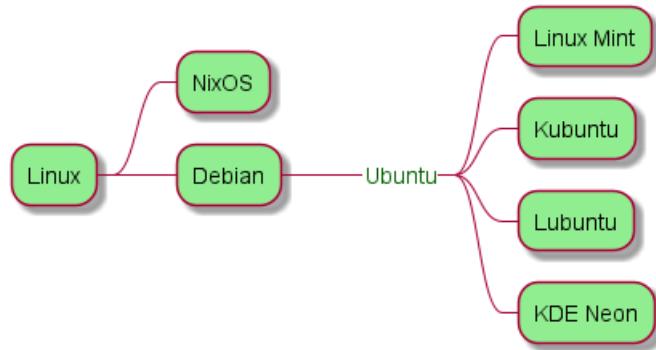
### 17.9.2 boxless

```

@startmindmap
<style>
mindmapDiagram {
    node {
        BackgroundColor lightGreen
    }
    boxless {
        FontColor darkgreen
    }
}
</style>
* Linux
** NixOS
** Debian
*** _ Ubuntu
**** Linux Mint
**** Kubuntu
**** Lubuntu
**** KDE Neon
@endmindmap

```





## 17.10 Word Wrap

Using `MaximumWidth` setting you can control automatic word wrap. Unit used is pixel.

`@startmindmap`

```

<style>
node {
    Padding 12
    Margin 3
    HorizontalAlignment center
    LineColor blue
    LineThickness 3.0
    BackgroundColor gold
    RoundCorner 40
    MaximumWidth 100
}

rootNode {
    LineStyle 8.0;3.0
    LineColor red
    BackgroundColor white
    LineThickness 1.0
    RoundCorner 0
    Shadowing 0.0
}

leafNode {
    LineColor gold
    RoundCorner 0
    Padding 3
}

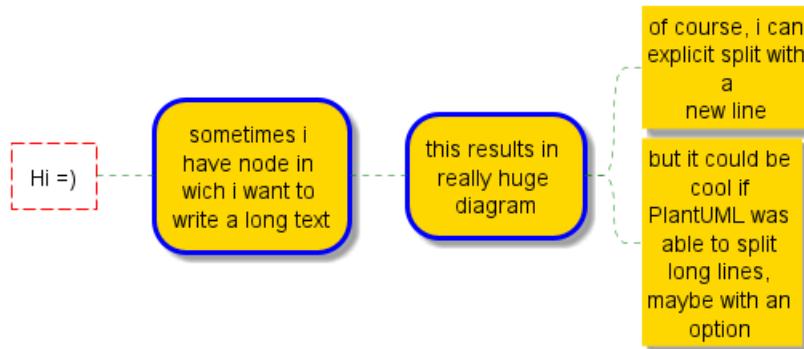
arrow {
    LineStyle 4
    LineThickness 0.5
    LineColor green
}
</style>

* Hi =)
** sometimes i have node in which i want to write a long text
*** this results in really huge diagram
**** of course, i can explicit split with a\nnew line
  
```



\*\*\*\* but it could be cool if PlantUML was able to split long lines, maybe with an option

@endmindmap



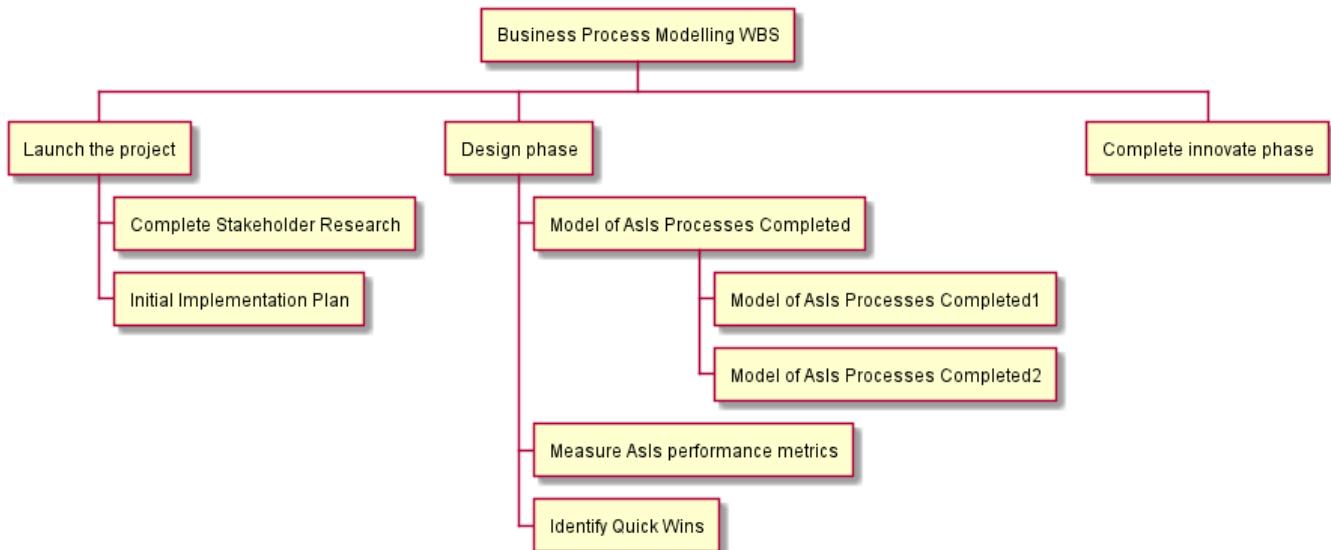
## 18 Work Breakdown Structure (WBS)

WBS diagram are still in beta: the syntax may change without notice.

### 18.1 OrgMode syntax

This syntax is compatible with OrgMode

```
@startwbs
* Business Process Modelling WBS
** Launch the project
*** Complete Stakeholder Research
**** Initial Implementation Plan
** Design phase
**** Model of AsIs Processes Completed
***** Model of AsIs Processes Completed1
***** Model of AsIs Processes Completed2
**** Measure AsIs performance metrics
**** Identify Quick Wins
** Complete innovate phase
@endwbs
```

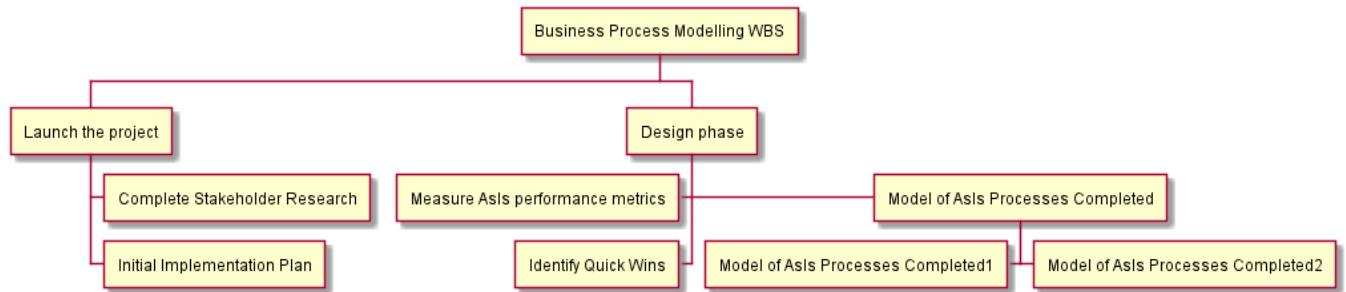


### 18.2 Change direction

You can change direction using < and >

```
@startwbs
* Business Process Modelling WBS
** Launch the project
*** Complete Stakeholder Research
**** Initial Implementation Plan
** Design phase
**** Model of AsIs Processes Completed
*****< Model of AsIs Processes Completed1
*****> Model of AsIs Processes Completed2
****< Measure AsIs performance metrics
****< Identify Quick Wins
@endwbs
```



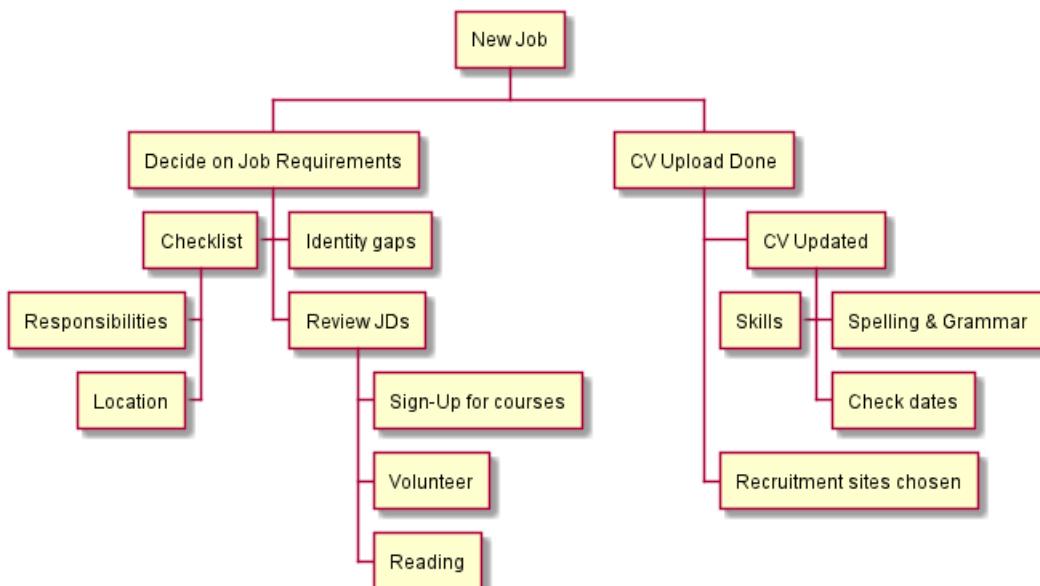


### 18.3 Arithmetic notation

You can use the following notation to choose diagram side.

```

@startwbs
+ New Job
++ Decide on Job Requirements
+++ Identity gaps
+++ Review JDs
++++ Sign-Up for courses
++++ Volunteer
++++ Reading
+++ Checklist
+++ Responsibilities
+++ Location
++ CV Upload Done
+++ CV Updated
++++ Spelling & Grammar
++++ Check dates
---- Skills
+++ Recruitment sites chosen
@endwbs
  
```



### 18.4 Removing box

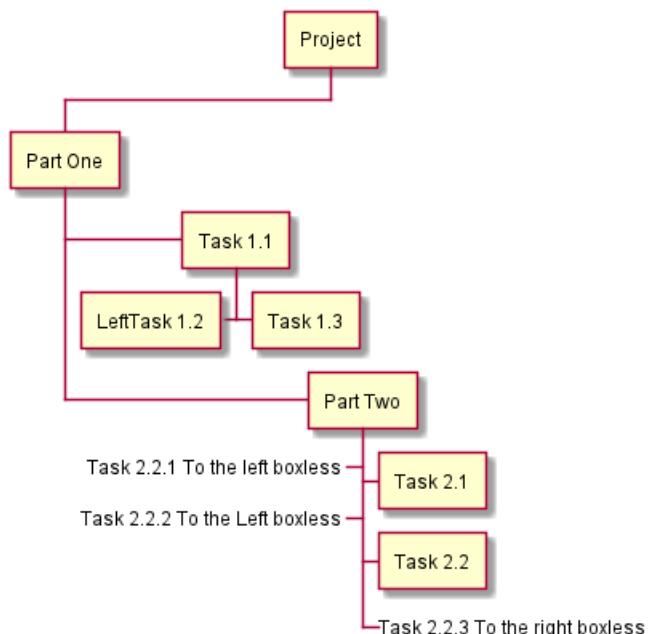
You can use underscore \_ to remove box drawing.



### 18.4.1 Boxless on Arithmetic notation

#### 18.4.2 Several boxless node

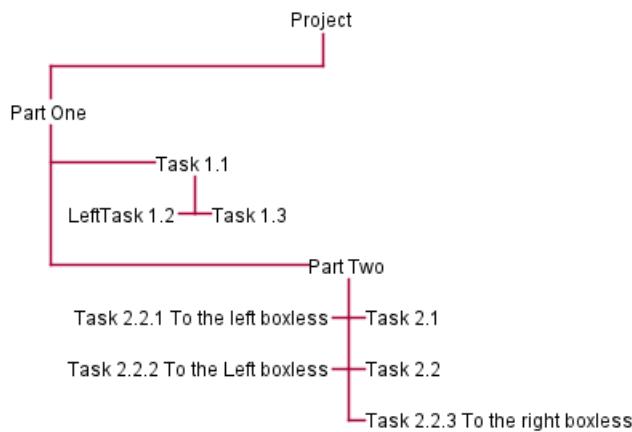
```
@startwbs
+ Project
+ Part One
+ Task 1.1
- LeftTask 1.2
+ Task 1.3
+ Part Two
+ Task 2.1
+ Task 2.2
-_ Task 2.2.1 To the left boxless
-_ Task 2.2.2 To the Left boxless
+_ Task 2.2.3 To the right boxless
@endwbs
```



#### 18.4.3 All boxless node

```
@startwbs
+_ Project
+_ Part One
+_ Task 1.1
-_ LeftTask 1.2
+_ Task 1.3
+_ Part Two
+_ Task 2.1
+_ Task 2.2
-_ Task 2.2.1 To the left boxless
-_ Task 2.2.2 To the Left boxless
+_ Task 2.2.3 To the right boxless
@endwbs
```

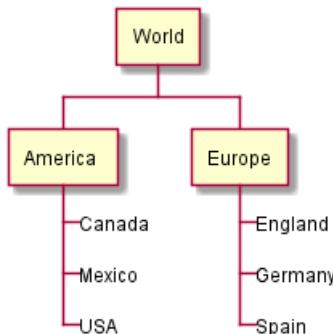




#### 18.4.4 Boxless on OrgMode syntax

#### 18.4.5 Several boxless node

```
@startwbs
* World
** America
*** _ Canada
*** _ Mexico
*** _ USA
** Europe
*** _ England
*** _ Germany
*** _ Spain
@endwbs
```

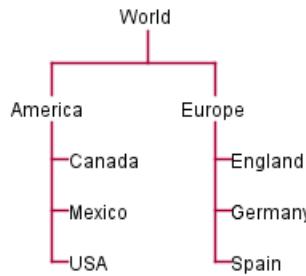


[Ref. QA-13297]

#### 18.4.6 All boxless node

```
@startwbs
*_ World
**_ America
***_ Canada
***_ Mexico
***_ USA
**_ Europe
***_ England
***_ Germany
***_ Spain
@endwbs
```





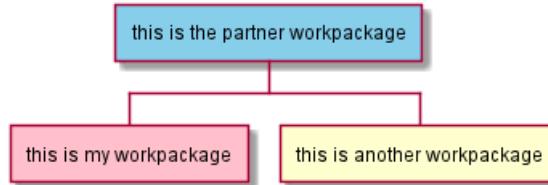
[Ref. QA-13355]

## 18.5 Colors (with inline or style color)

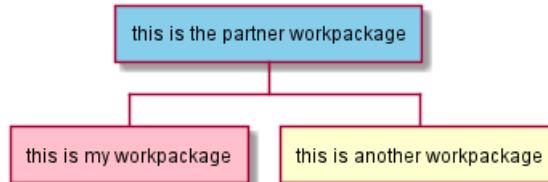
It is possible to change node color:

- with inline color

```
@startwbs
*[#SkyBlue] this is the partner workpackage
**[#pink] this is my workpackage
** this is another workpackage
@endwbs
```



```
@startwbs
+[#SkyBlue] this is the partner workpackage
++[#pink] this is my workpackage
++ this is another workpackage
@endwbs
```



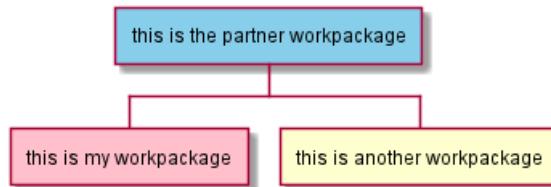
[Ref. QA-12374, only from v1.2020.20]

- with style color

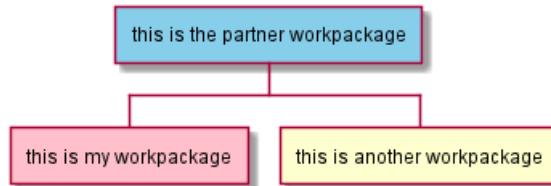
```
@startwbs
<style>
wbsDiagram {
    .pink {
        BackgroundColor pink
    }
    .your_style_name {
        BackgroundColor SkyBlue
    }
}
</style>
* this is the partner workpackage <<your_style_name>>
```



```
** this is my workpackage <<pink>>
** this is another workpackage
@endwbs
```



```
@startwbs
<style>
wbsDiagram {
    .pink {
        BackgroundColor pink
    }
    .your_style_name {
        BackgroundColor SkyBlue
    }
}
</style>
+ this is the partner workpackage <<your_style_name>>
++ this is my workpackage <<pink>>
++ this is another workpackage
@endwbs
```



## 18.6 Using style

It is possible to change diagram style.

```
@startwbs
<style>
wbsDiagram {
    // all lines (meaning connector and borders, there are no other lines in WBS) are black by default
    Linecolor black
    arrow {
        // note that connector are actually "arrow" even if they don't look like as arrow
        // This is to be consistent with other UML diagrams. Not 100% sure that it's a good idea
        // So now connector are green
        LineColor green
    }
    :depth(0) {
        // will target root node
        BackgroundColor White
        RoundCorner 10
        LineColor red
        // Because we are targetting depth(0) for everything, border and connector for level 0 will be
    }
    arrow {
        :depth(2) {

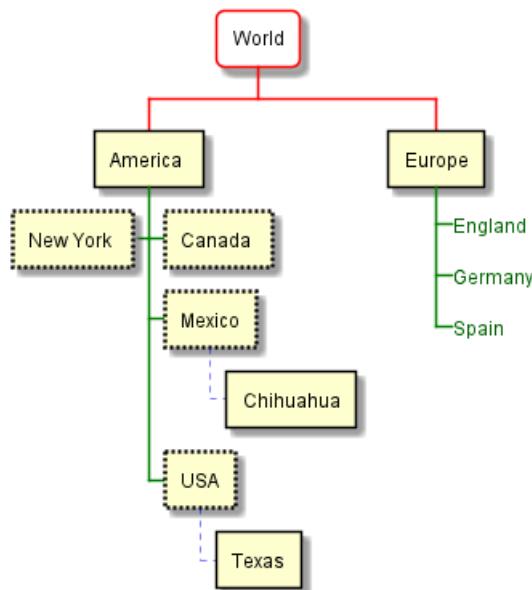
```



```

// Targetting only connector between Mexico-Chihuahua and USA-Texas
LineColor blue
LineStyle 4
LineThickness .5
}
}
node {
:depth(2) {
LineStyle 2
LineThickness 2.5
}
}
boxless {
// will target boxless node with '_'
FontColor darkgreen
}
}
</style>
* World
** America
*** Canada
*** Mexico
**** Chihuahua
*** USA
**** Texas
***< New York
** Europe
***_ England
***_ Germany
***_ Spain
@endwbs

```



## 18.7 Word Wrap

Using `MaximumWidth` setting you can control automatic word wrap. Unit used is pixel.

`@startwbs`



```
<style>
node {
    Padding 12
    Margin 3
    HorizontalAlignment center
    LineColor blue
    LineThickness 3.0
    BackgroundColor gold
    RoundCorner 40
    MaximumWidth 100
}

rootNode {
    LineStyle 8.0;3.0
    LineColor red
    BackgroundColor white
    LineThickness 1.0
    RoundCorner 0
    Shadowing 0.0
}

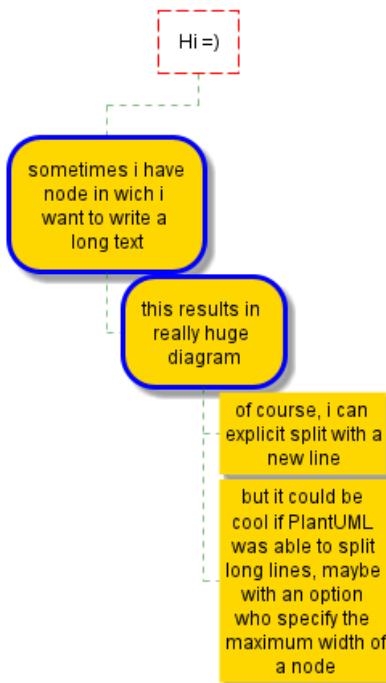
leafNode {
    LineColor gold
    RoundCorner 0
    Padding 3
}

arrow {
    LineStyle 4
    LineThickness 0.5
    LineColor green
}
</style>

* Hi =)
** sometimes i have node in which i want to write a long text
*** this results in really huge diagram
**** of course, i can explicitly split with a\nnew line
**** but it could be cool if PlantUML was able to split long lines, maybe with an option who specifies

@endwbs
```

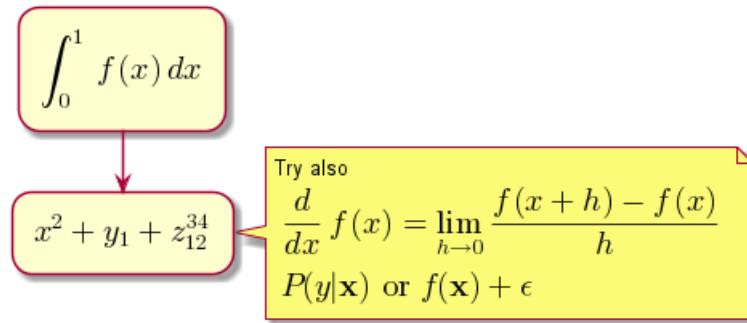




## 19 Maths

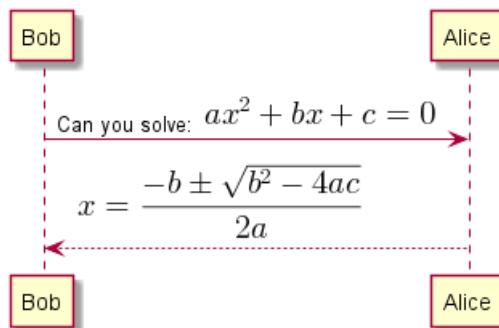
You can use AsciiMath or JLaTeXMath notation within PlantUML:

```
@startuml
:<math>\int_0^1 f(x) dx</math>;
:<math>x^2 + y_1 + z_{12}^{34}</math>;
note right
Try also
<math>d/dx f(x) = \lim_{h \rightarrow 0} (f(x+h) - f(x))/h</math>
<math>P(y|x) \text{ or } f(\mathbf{x}) + \epsilon</math>
end note
@enduml
```



or:

```
@startuml
Bob --> Alice : Can you solve: <math>ax^2+bx+c=0</math>
Alice --> Bob: <math>x = (-b \pm \sqrt{b^2-4ac})/(2a)</math>
@enduml
```



### 19.1 Standalone diagram

You can also use `@startmath/@endmath` to create standalone AsciiMath formula.

```
@startmath
f(t)=(a_0)/2 + sum_(n=1)^oo a_ncos((npit)/L)+sum_(n=1)^oo b_n\ sin((npit)/L)
@endmath
```

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(\frac{n\pi t}{L}\right) + \sum_{n=1}^{\infty} b_n \sin\left(\frac{n\pi t}{L}\right)$$

Or use `@startlatex/@endlatex` to create standalone JLaTeXMath formula.

```
@startlatex
\sum_{i=0}^{n-1} (a_i + b_i^2)
@endlatex
```



$$\sum_{i=0}^{n-1} (a_i + b_i^2)$$

## 19.2 How is this working?

To draw those formulas, PlantUML uses two open source projects:

- AsciiMath that converts AsciiMath notation to LaTeX expression;
- JLatexMath that displays mathematical formulas written in LaTeX. JLaTeXMath is the best Java library to display LaTeX code.

ASCIIMathTeXImg.js is small enough to be integrated into PlantUML standard distribution.

PlantUML relies on the Java Scripting API (specifically: `new ScriptEngineManager().getEngineByName("JavaScript")`) to load a JavaScript engine and execute JavaScript code. Java 8 includes a JavaScript engine called Nashorn but it was deprecated in Java 11.

If you are using AsciiMath in Java 11 you see the following warnings:

**Warning: Nashorn engine is planned to be removed from a future JDK release**

Nashorn was removed in Java 15. Fortunately, you can use the GraalVM JavaScript Engine instead by adding the following dependencies:

```
<dependency>
  <groupId>org.graalvm.js</groupId>
  <artifactId>js</artifactId>
  <version>20.2.0</version>
</dependency>
<dependency>
  <groupId>org.graalvm.js</groupId>
  <artifactId>js-scriptengine</artifactId>
  <version>20.2.0</version>
</dependency>
```

You can even use the GraalVM JavaScript Engine in Java 11 to get rid of the warning messages.

Since JLatexMath is bigger, you have to download it separately, then unzip the 4 jar files (*batik-all-1.7.jar*, *jlatexmath-minimal-1.0.3.jar*, *jlm\_cyrillic.jar* and *jlm\_greek.jar*) in the same folder as PlantUML.jar.



## 20 Entity Relationship Diagram

Based on the Information Engineering notation.

This is an extension to the existing Class Diagram. This extension adds:

- Additional relations for the Information Engineering notation.
- An `entity` alias that maps to the class diagram `class`.
- An additional visibility modifier `*` to identify mandatory attributes.

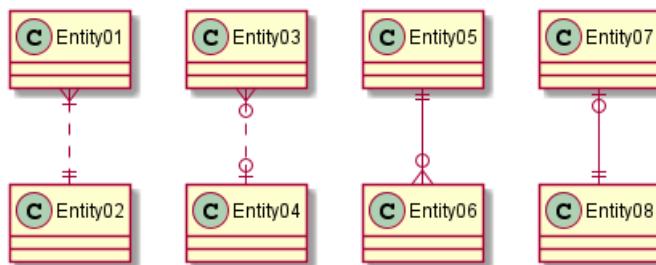
Otherwise, the syntax for drawing diagrams is the same as for class diagrams. All other features of class diagrams are also supported.

### 20.1 Information Engineering Relations

Type	Symbol
Zero or One	o--
Exactly One	--
Zero or Many	}o--
One or Many	} --

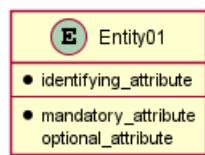
Examples:

```
@startuml
Entity01 }|...|| Entity02
Entity03 }o..o| Entity04
Entity05 ||--o{ Entity06
Entity07 |o--|| Entity08
@enduml
```



### 20.2 Entities

```
@startuml
entity Entity01 {
    * identifying_attribute
    --
    * mandatory_attribute
    optional_attribute
}
@enduml
```



Again, this is the normal class diagram syntax (aside from use of `entity` instead of `class`). Anything that you can do in a class diagram can be done here.

The `*` visibility modifier can be used to identify mandatory attributes. A space can be used after the modifier character to avoid conflicts with the creole bold:



```
@startuml
entity Entity01 {
    optional attribute
    **optional bold attribute**
    * **mandatory bold attribute**
}
@enduml
```



### 20.3 Complete Example

```
@startuml

' hide the spot
hide circle

' avoid problems with angled crows feet
skinparam linetype ortho

entity "Entity01" as e01 {
    *e1_id : number <<generated>>
    --
    *name : text
    description : text
}

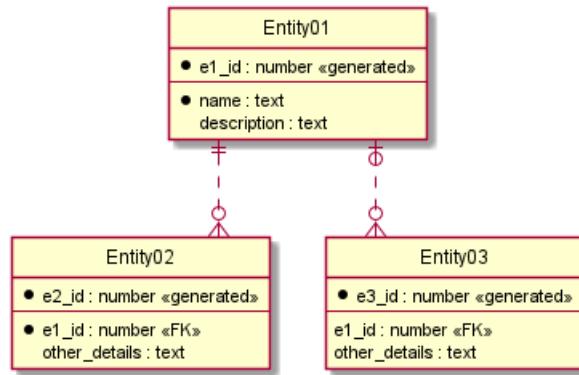
entity "Entity02" as e02 {
    *e2_id : number <<generated>>
    --
    *e1_id : number <<FK>>
    other_details : text
}

entity "Entity03" as e03 {
    *e3_id : number <<generated>>
    --
    e1_id : number <<FK>>
    other_details : text
}

e01 ||..o{ e02
e01 |o..o{ e03

@enduml
```





Currently the crows feet do not look very good when the relationship is drawn at an angle to the entity. This can be avoided by using the `linetype ortho` skinparam.



## 21 Common commands

### 21.1 Comments

Everything that starts with `simple quote '` is a comment.

You can also put comments on several lines using `'/` to start and `'/` to end.

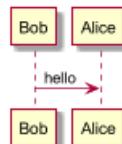
### 21.2 Zoom

You can use the `scale` command to zoom the generated image.

You can use either a number or a fraction to define the scale factor. You can also specify either width or height (in pixel). And you can also give both width and height : the image is scaled to fit inside the specified dimension.

- `scale 1.5`
- `scale 2/3`
- `scale 200 width`
- `scale 200 height`
- `scale 200*100`
- `scale max 300*200`
- `scale max 1024 width`
- `scale max 800 height`

```
@startuml
scale 180*90
Bob->Alice : hello
@enduml
```



### 21.3 Title

The `title` keyword is used to put a title. You can add newline using `\n` in the title description.

Some `skinparam` settings are available to put borders on the title.

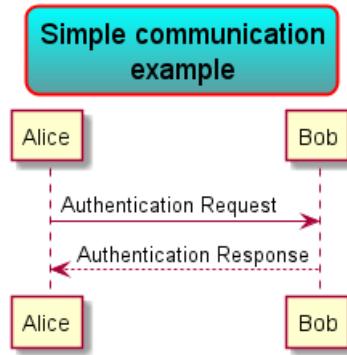
```
@startuml
skinparam titleBorderRoundCorner 15
skinparam titleBorderThickness 2
skinparam titleBorderColor red
skinparam titleBackgroundColor Aqua-CadetBlue

title Simple communication\nexample

Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response

@enduml
```





You can use creole formatting in the title.

You can also define title on several lines using `title` and `end title` keywords.

```
@startuml
```

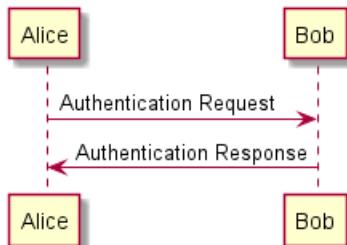
```

title
<u>Simple</u> communication example
on <i>several</i> lines and using <back:cadetblue>creole tags</back>
end title

Alice -> Bob: Authentication Request
Bob -> Alice: Authentication Response
  
```

```
@enduml
```

### Simple communication example on *several* lines and using creole tags



## 21.4 Caption

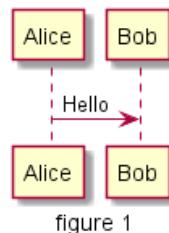
There is also a `caption` keyword to put a caption under the diagram.

```
@startuml
```

```

caption figure 1
Alice -> Bob: Hello
  
```

```
@enduml
```



## 21.5 Footer and header

You can use the commands `header` or `footer` to add a footer or a header on any generated diagram.

You can optionally specify if you want a `center`, `left` or `right` footer/header, by adding a keyword.

As with title, it is possible to define a header or a footer on several lines.

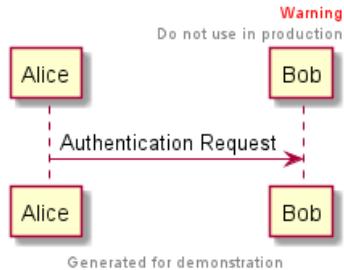
It is also possible to put some HTML into the header or footer.

```
@startuml
Alice -> Bob: Authentication Request
```

```
header
<font color=red>Warning:</font>
Do not use in production.
endheader

center footer Generated for demonstration
```

```
@enduml
```

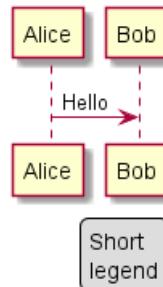


## 21.6 Legend the diagram

The `legend` and `end legend` are keywords used to put a legend.

You can optionally specify to have `left`, `right`, `top`, `bottom` or `center` alignment for the legend.

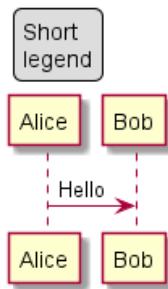
```
@startuml
Alice -> Bob : Hello
legend right
Short
legend
endlegend
@enduml
```



```
@startuml
Alice -> Bob : Hello
legend top left
Short
legend
endlegend
```



```
@enduml
```



## 21.7 Appendix: Examples on all diagram

### 21.7.1 Activity

```
@startuml
header some header
```

```
footer some footer
```

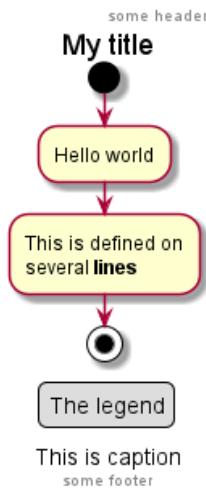
```
title My title
```

```
caption This is caption
```

```
legend
The legend
end legend
```

```
start
:Hello world;
:This is defined on
several **lines**;
stop
```

```
@enduml
```



### 21.7.2 Archimate

```
@startuml
header some header
```



```

footer some footer

title My title

caption This is caption

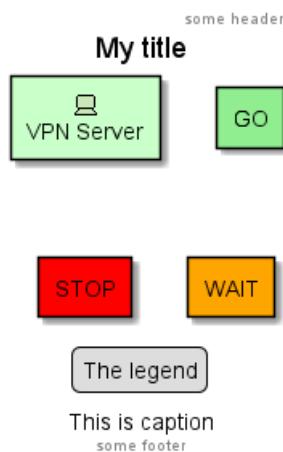
legend
The legend
end legend

archimate #Technology "VPN Server" as vpnServerA <<technology-device>>

rectangle GO #lightgreen
rectangle STOP #red
rectangle WAIT #orange

@enduml

```



### 21.7.3 Class

```

@startuml
header some header

footer some footer

title My title

caption This is caption

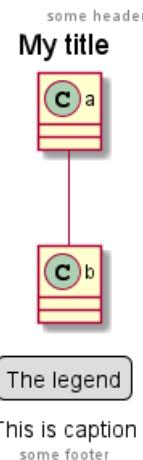
legend
The legend
end legend

a -- b

@enduml

```





#### 21.7.4 Component, Deployment, Use-Case

```
@startuml
header some header

footer some footer

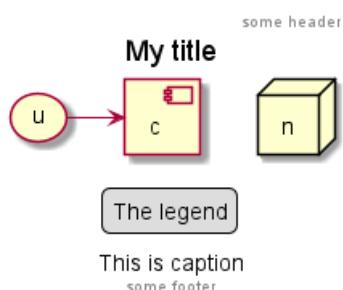
title My title

caption This is caption

legend
The legend
end legend

node n
(u) -> [c]

@enduml
```



#### 21.7.5 Gantt project planning

```
@startuml
header some header

footer some footer

title My title

caption This is caption

legend
The legend
```



```
end legend
```

```
[t] lasts 5 days
```

```
@enduml
```

some header

### My title

1	2	3	4	5
t				
1	2	3	4	5

The legend

This is caption

some footer

**TODO:** DONE *[(Header, footer) corrected on V1.2020.18]*

#### 21.7.6 Object

```
@startuml
header some header
```

```
footer some footer
```

```
title My title
```

```
caption This is caption
```

```
legend
```

```
The legend
```

```
end legend
```

```
object user {
    name = "Dummy"
    id = 123
}
```

```
@enduml
```

some header

### My title

user
name = "Dummy"

The legend

This is caption

some footer

#### 21.7.7 MindMap

```
@startmindmap
header some header
```

```
footer some footer
```

```
title My title
```

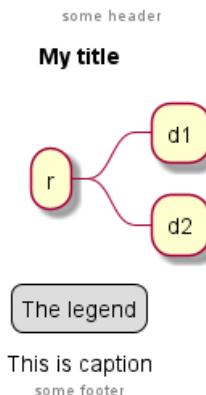


```
caption This is caption
```

```
legend
The legend
end legend
```

```
* r
** d1
** d2
```

```
@endmindmap
```



### 21.7.8 Network (nwdiag)

```
@startuml
header some header

footer some footer

title My title

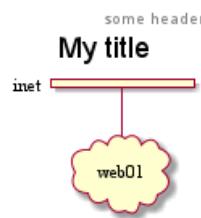
caption This is caption
```

```
legend
The legend
end legend
```

```
nwdiag {
    network inet {
        web01 [shape = cloud]
    }
}
```

```
@enduml
```





This is caption  
some footer

### 21.7.9 Sequence

```

@startuml
header some header

footer some footer

title My title

caption This is caption

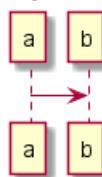
legend
The legend
end legend

a->b
@enduml

```

some header

**My title**



The legend

This is caption

some footer

### 21.7.10 State

```

@startuml
header some header

footer some footer

title My title

caption This is caption

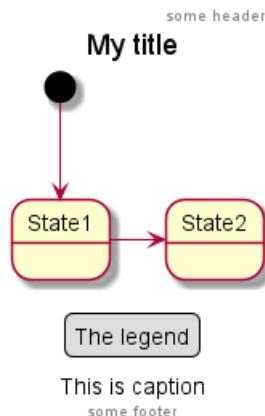
legend
The legend
end legend

```



```
[*] --> State1
State1 -> State2
```

```
@enduml
```



### 21.7.11 Timing

```
@startuml
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

robust "Web Browser" as WB
concise "Web User" as WU

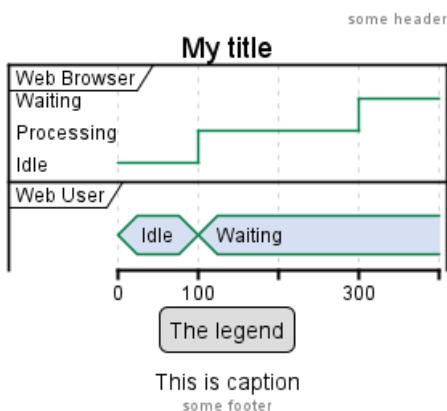
@0
WU is Idle
WB is Idle

@100
WU is Waiting
WB is Processing

@300
WB is Waiting

@enduml
```





### 21.7.12 Work Breakdown Structure (WBS)

```
@startwbs
header some header

footer some footer

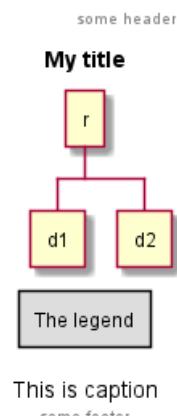
title My title

caption This is caption

legend
The legend
end legend

* r
** d1
** d2

@endwbs
```



**TODO:** DONE [*Corrected on V1.2020.17*]

### 21.7.13 Wireframe (SALT)

```
@startsalt
header some header

footer some footer
```

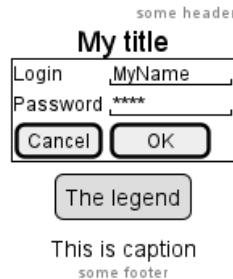


```

title My title
caption This is caption
legend
The legend
end legend

{+
    Login | "MyName"
    Password | "****"
    [Cancel] | [ OK ]
}
@endsalt

```



**TODO:** DONE [*Corrected on V1.2020.18*]

## 21.8 Appendix: Examples on all diagram with style

**TODO:** DONE

FYI:

- all is only good for **Sequence diagram**
- **title**, **caption** and **legend** are good for all diagrams except for **salt diagram**

**TODO:** FIXME

- Now (*test on 1.2020.18-19*) **header**, **footer** are not good for **all other diagrams** except only for **Sequence diagram**.

To be fix; Thanks

**TODO:** FIXME

Here are tests of **title**, **header**, **footer**, **caption** or **legend** on all the diagram with the debug style:

```

<style>
title {
    HorizontalAlignment right
    FontSize 24
    FontColor blue
}

header {
    HorizontalAlignment center
    FontSize 26
    FontColor purple
}

```

```

footer {
    HorizontalAlignment left
    FontSize 28
}

```



```

    FontColor red
}

legend {
    FontSize 30
    BackGroundColor yellow
    Margin 30
    Padding 50
}

caption {
    FontSize 32
}
</style>

```

### 21.8.1 Activity

```

@startuml
<style>
title {
    HorizontalAlignment right
    FontSize 24
    FontColor blue
}

header {
    HorizontalAlignment center
    FontSize 26
    FontColor purple
}

footer {
    HorizontalAlignment left
    FontSize 28
    FontColor red
}

legend {
    FontSize 30
    BackGroundColor yellow
    Margin 30
    Padding 50
}

caption {
    FontSize 32
}
</style>
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

```

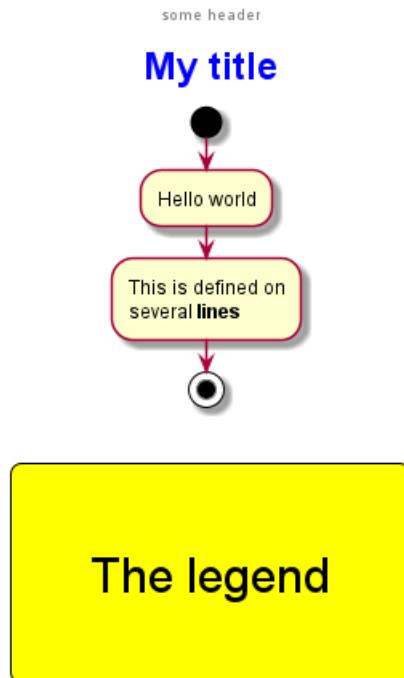


```

start
:Hello world;
:This is defined on
several **lines**;
stop

```

```
@enduml
```



## This is caption

some footer

### 21.8.2 Archimate

```

@startuml
<style>
title {
    HorizontalAlignment right
    FontSize 24
    FontColor blue
}

header {
    HorizontalAlignment center
    FontSize 26
    FontColor purple
}

footer {
    HorizontalAlignment left
    FontSize 28
    FontColor red
}

```



```

legend {
    FontSize 30
    BackGroundColor yellow
    Margin 30
    Padding 50
}

caption {
    FontSize 32
}
</style>
header some header

footer some footer

title My title

caption This is caption

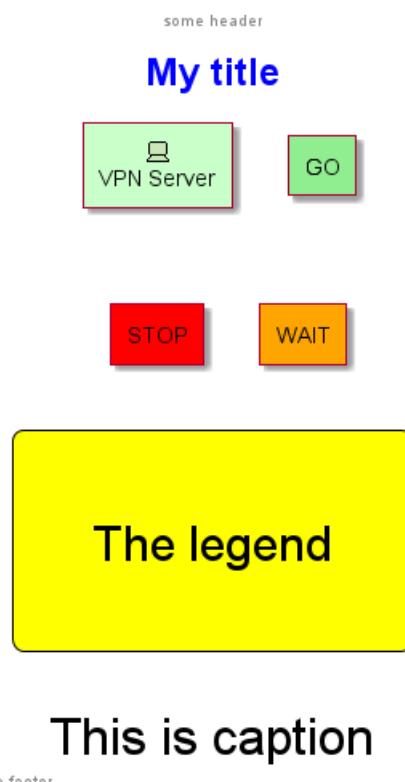
legend
The legend
end legend

archimate #Technology "VPN Server" as vpnServerA <<technology-device>>

rectangle GO #lightgreen
rectangle STOP #red
rectangle WAIT #orange

@enduml

```



### 21.8.3 Class

```
@startuml
<style>
title {
    HorizontalAlignment right
    FontSize 24
    FontColor blue
}

header {
    HorizontalAlignment center
    FontSize 26
    FontColor purple
}

footer {
    HorizontalAlignment left
    FontSize 28
    FontColor red
}

legend {
    FontSize 30
    BackGroundColor yellow
    Margin 30
    Padding 50
}

caption {
    FontSize 32
}
</style>
header some header

footer some footer

title My title

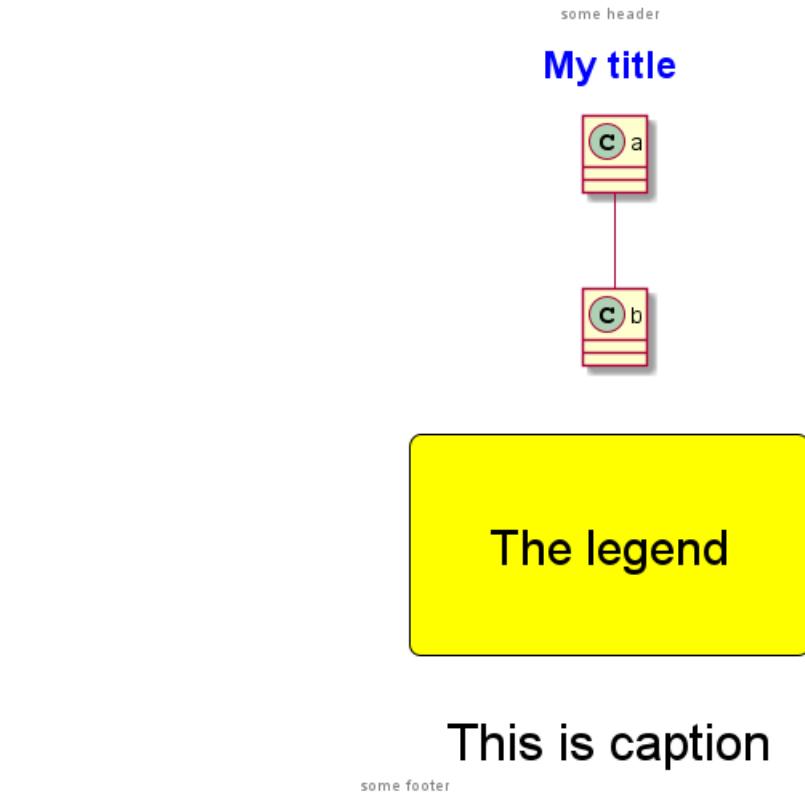
caption This is caption

legend
The legend
end legend

a -- b

@enduml
```





#### 21.8.4 Component, Deployment, Use-Case

```

@startuml
<style>
title {
    HorizontalAlignment right
    FontSize 24
    FontColor blue
}

header {
    HorizontalAlignment center
    FontSize 26
    FontColor purple
}

footer {
    HorizontalAlignment left
    FontSize 28
    FontColor red
}

legend {
    FontSize 30
    BackGroundColor yellow
    Margin 30
    Padding 50
}

caption {
    FontSize 32
}

```



```
</style>
header some header

footer some footer

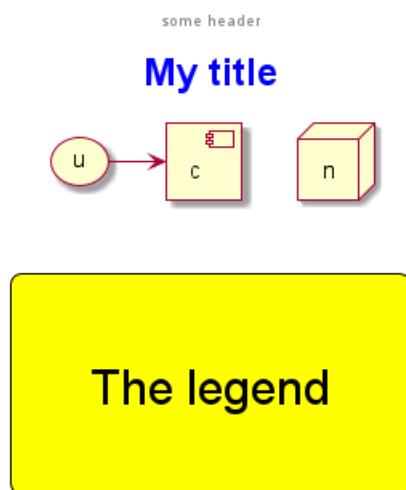
title My title

caption This is caption

legend
The legend
end legend

node n
(u) -> [c]

@enduml
```



This is caption

some footer

#### 21.8.5 Gantt project planning

```
@startuml
<style>
title {
    HorizontalAlignment right
    FontSize 24
    FontColor blue
}

header {
    HorizontalAlignment center
    FontSize 26
    FontColor purple
}

footer {
    HorizontalAlignment left
    FontSize 28
    FontColor red
}
```



```

}

legend {
    FontSize 30
    BackGroundColor yellow
    Margin 30
    Padding 50
}

caption {
    FontSize 32
}
</style>
header some header

footer some footer

title My title

caption This is caption

```

```

legend
The legend
end legend

```

```
[t] lasts 5 days
```

```
@enduml
```

some header

## My title

1	2	3	4	5
t				
1	2	3	4	5



The legend

This is caption

some footer

### 21.8.6 Object

```

@startuml
<style>
title {
    HorizontalAlignment right
    FontSize 24
    FontColor blue
}

```



```
header {
    HorizontalAlignment center
    FontSize 26
    FontColor purple
}

footer {
    HorizontalAlignment left
    FontSize 28
    FontColor red
}

legend {
    FontSize 30
    BackGroundColor yellow
    Margin 30
    Padding 50
}

caption {
    FontSize 32
}
</style>
header some header

footer some footer

title My title

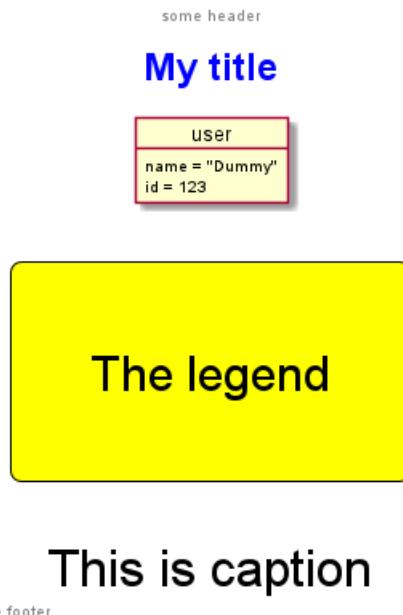
caption This is caption

legend
The legend
end legend

object user {
    name = "Dummy"
    id = 123
}

@enduml
```





### 21.8.7 MindMap

```
@startmindmap
<style>
title {
    HorizontalAlignment right
    FontSize 24
    FontColor blue
}

header {
    HorizontalAlignment center
    FontSize 26
    FontColor purple
}

footer {
    HorizontalAlignment left
    FontSize 28
    FontColor red
}

legend {
    FontSize 30
    BackGroundColor yellow
    Margin 30
    Padding 50
}

caption {
    FontSize 32
}
</style>
header some header

footer some footer

title My title
```

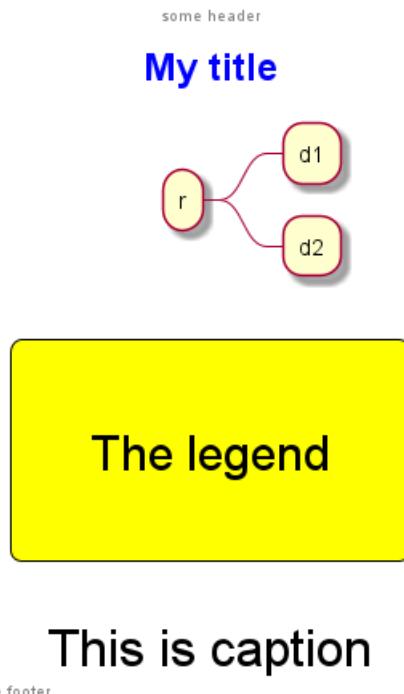


```
caption This is caption
```

```
legend
The legend
end legend
```

```
* r
** d1
** d2
```

```
@endmindmap
```



#### 21.8.8 Network (nwdiag)

```
@startuml
<style>
title {
    HorizontalAlignment right
    FontSize 24
    FontColor blue
}

header {
    HorizontalAlignment center
    FontSize 26
    FontColor purple
}

footer {
    HorizontalAlignment left
    FontSize 28
    FontColor red
}
```



```

legend {
    FontSize 30
    BackGroundColor yellow
    Margin 30
    Padding 50
}

caption {
    FontSize 32
}
</style>
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

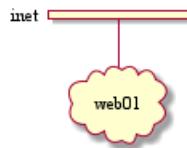
nwdiag {
    network inet {
        web01 [shape = cloud]
    }
}

```

@enduml

some header

## My title



The legend

This is caption

some footer

### 21.8.9 Sequence

```

@startuml
<style>
title {

```



```
HorizontalAlignment right
FontSize 24
FontColor blue
}

header {
    HorizontalAlignment center
    FontSize 26
    FontColor purple
}

footer {
    HorizontalAlignment left
    FontSize 28
    FontColor red
}

legend {
    FontSize 30
    BackGroundColor yellow
    Margin 30
    Padding 50
}

caption {
    FontSize 32
}
</style>
header some header

footer some footer

title My title

caption This is caption

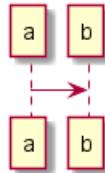
legend
The legend
end legend

a->b
@enduml
```



some header

## My title



The legend

This is caption  
some footer

### 21.8.10 State

```

@startuml
<style>
title {
    HorizontalAlignment right
    FontSize 24
    FontColor blue
}

header {
    HorizontalAlignment center
    FontSize 26
    FontColor purple
}

footer {
    HorizontalAlignment left
    FontSize 28
    FontColor red
}

legend {
    FontSize 30
    BackGroundColor yellow
    Margin 30
    Padding 50
}

caption {
    FontSize 32
}
</style>
  
```



```

header some header

footer some footer

title My title

caption This is caption

```

```

legend
The legend
end legend

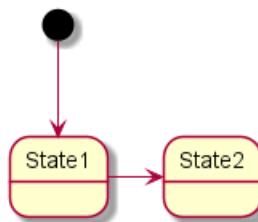
[*] --> State1
State1 -> State2

```

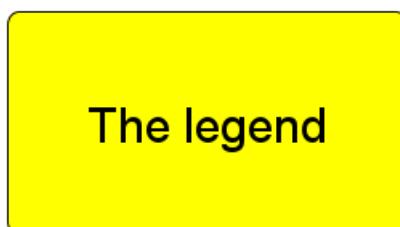
```
@enduml
```

some header

## My title



The legend



## This is caption

some footer

### 21.8.11 Timing

```

@startuml
<style>
title {
    HorizontalAlignment right
    FontSize 24
    FontColor blue
}

header {
    HorizontalAlignment center
    FontSize 26
    FontColor purple
}

```

```

footer {

```



```
HorizontalAlignment left
FontSize 28
FontColor red
}
```

```
legend {
    FontSize 30
    BackGroundColor yellow
    Margin 30
    Padding 50
}
```

```
caption {
    FontSize 32
}
</style>
header some header
```

```
footer some footer
```

```
title My title
```

```
caption This is caption
```

```
legend
The legend
end legend
```

```
robust "Web Browser" as WB
concise "Web User" as WU
```

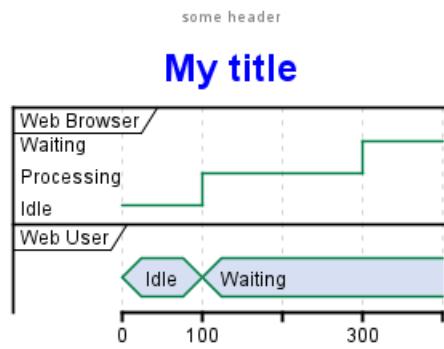
```
@0
WU is Idle
WB is Idle
```

```
@100
WU is Waiting
WB is Processing
```

```
@300
WB is Waiting
```

```
@enduml
```





This is caption

some footer

#### 21.8.12 Work Breakdown Structure (WBS)

```

@startwbs
<style>
title {
    HorizontalAlignment right
    FontSize 24
    FontColor blue
}

header {
    HorizontalAlignment center
    FontSize 26
    FontColor purple
}

footer {
    HorizontalAlignment left
    FontSize 28
    FontColor red
}

legend {
    FontSize 30
    BackGroundColor yellow
    Margin 30
    Padding 50
}

caption {
    FontSize 32
}
</style>

```



```

header some header

footer some footer

title My title

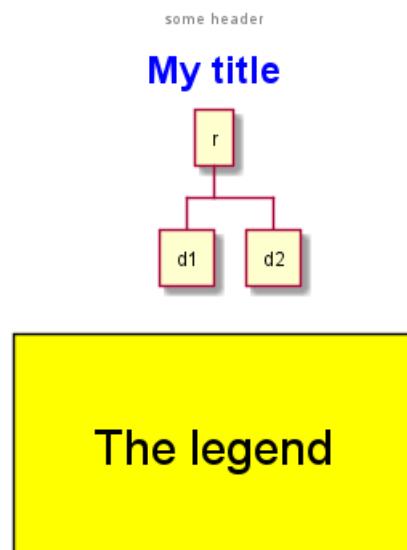
caption This is caption

legend
The legend
end legend

* r
** d1
** d2

@endwbs

```



This is caption

some footer

### 21.8.13 Wireframe (SALT)

**TODO:**FIXME Fix all (**title**, **caption**, **legend**, **header**, **footer**) for salt. **TODO:**FIXME

```

@startsalt
<style>
title {
    HorizontalAlignment right
    FontSize 24
    FontColor blue
}

header {
    HorizontalAlignment center
    FontSize 26
    FontColor purple
}

```



```
footer {
    HorizontalAlignment left
    FontSize 28
    FontColor red
}
```

```
legend {
    FontSize 30
    BackGroundColor yellow
    Margin 30
    Padding 50
}
```

```
caption {
    FontSize 32
}
</style>
@startsalt
header some header
```

```
footer some footer
```

```
title My title
```

```
caption This is caption
```

```
legend
The legend
end legend
```

```
{+
    Login | "MyName"
    Password | "*****"
    [Cancel] | [ OK ]
}
@endsalt
```



## 22 Creole

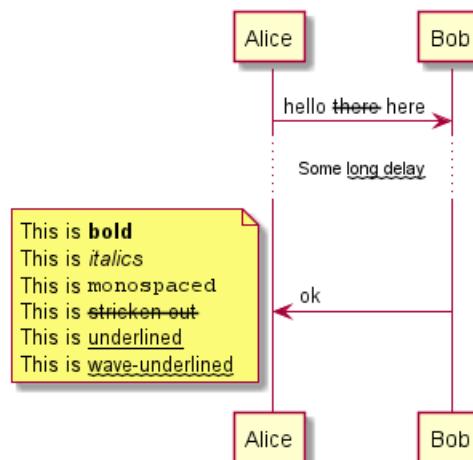
Creole is a lightweight common markup language for various wikis. A light-weight Creole engine is integrated in PlantUML to have a standardized way to emit styled text.

All diagrams support this syntax.

Note that compatibility with HTML syntax is preserved.

### 22.1 Emphasized text

```
@startuml
Alice -> Bob : hello --there-- here
... Some ~~long delay~~ ...
Bob -> Alice : ok
note left
    This is **bold**
    This is //italics//
    This is ""monospaced"""
    This is --stricken-out--
    This is __underlined__
    This is ~~wave-underlined~~
end note
@enduml
```



### 22.2 Lists

You can use numbered and bulleted lists in node text, notes, etc.

**TODO: FIXME** You cannot quite mix numbers and bullets in a list and its sublist.

```
@startuml
object demo {
    * Bullet list
    * Second item
}
note left
    * Bullet list
    * Second item
    ** Sub item
end note

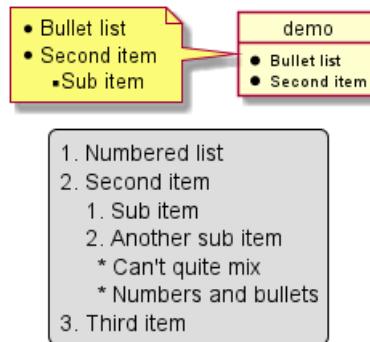
legend
    # Numbered list
    # Second item
```



```

## Sub item
## Another sub item
  * Can't quite mix
  * Numbers and bullets
# Third item
end legend
@enduml

```



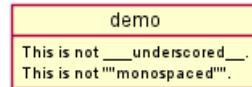
## 22.3 Escape character

You can use the tilde ~ to escape special creole characters.

```

@startuml
object demo {
    This is not ~__underscored__.
    This is not ~""monospaced"".
}
@enduml

```



## 22.4 Horizontal lines

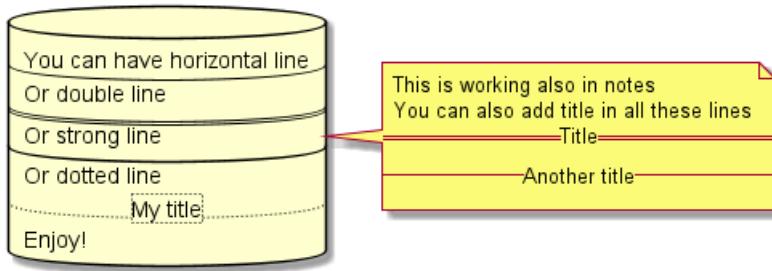
```

@startuml
database DB1 as "
You can have horizontal line
-----
Or double line
=====
Or strong line
-----
Or dotted line
..My title..
Enjoy!
"
note right
  This is working also in notes
  You can also add title in all these lines
  ==Title==
  --Another title--
end note

@enduml

```





## 22.5 Headings

```
@startuml
usecase UC1 as "
= Extra-large heading
Some text
== Large heading
Other text
==== Medium heading
Information
....
===== Small heading"
@enduml
```



## 22.6 Legacy HTML

You can mix Creole with the following HTML tags:

- **<b>** for bold text
- <u> or <u:#AAAAAA> or <u: [[color|colorName]]> for underline
- *<i>* for italic
- ~~<s>~~ or ~~<s:#AAAAAA>~~ or ~~<s: [[color|colorName]]>~~ for strike text
- w> or w:#AAAAAA> or w: [[color|colorName]]> for wave underline text
- 
- 
- - -

```
@startuml
*: You can change <color:red>text color</color>
* You can change <back:cadetblue>background color</back>
* You can change <size:18>size</size>
* You use <u>legacy</u> <b>HTML <i>tag</i></b>
```



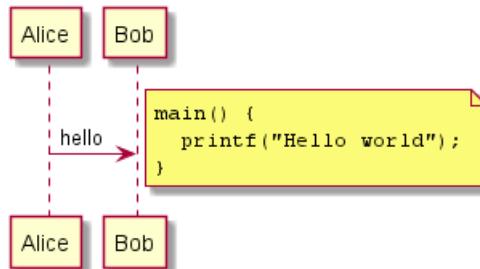
```
* You use <u:red>color</u> <s:green>in HTML</s> <w:#0000FF>tag</w>
-----
* Use image : <img:http://plantuml.com/logo3.png>
;
@enduml
```



## 22.7 Code

You can use `<code>` to display some programming code in your diagram (sorry, syntax highlighting is not yet supported).

```
@startuml
Alice -> Bob : hello
note right
<code>
main() {
    printf("Hello world");
}
</code>
end note
@enduml
```

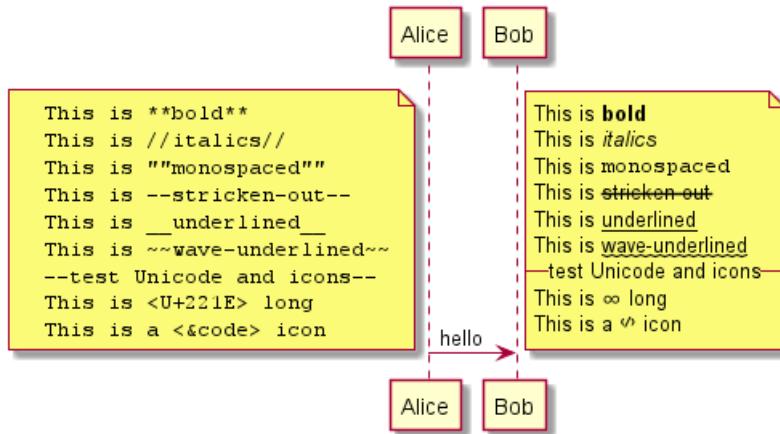


This is especially useful to illustrate some PlantUML code and the resulting rendering:

```
@startuml
Alice -> Bob : hello
note left
<code>
This is **bold**
This is //italics//
This is ""monospaced"""
This is --stricken-out--
This is __underlined__
This is ~~wave-underlined~~
</code>
```



```
--test Unicode and icons--
This is <U+221E> long
This is a <&code> icon
</code>
end note
note right
  This is **bold**
  This is //italics//
  This is ""monospaced"""
  This is --stricken-out--
  This is __underlined__
  This is ~~wave-underlined~~
--test Unicode and icons--
This is <U+221E> long
This is a <&code> icon
end note
@enduml
```



## 22.8 Table

### 22.8.1 Create a table

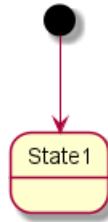
It is possible to build table, with | separator.

```
@startuml
skinparam titleFontSize 14
title
  Example of simple table
  |= |= table |= header |
  | a | table | row |
  | b | table | row |
end title
[*] --> State1
@enduml
```



Example of simple table

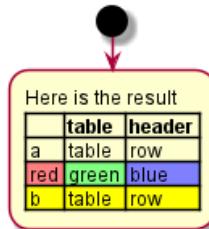
	table	header
a	table	row
b	table	row



### 22.8.2 Add color on rows or cells

You can specify background colors of rows and cells:

```
@startuml
start
:Here is the result
|= |= table |= header |
| a | table | row |
|<#FF8080> red |<#80FF80> green |<#8080FF> blue |
<#yellow>| b | table | row |;
@enduml
```



### 22.8.3 Add color on border and text

You can also specify colors of text and borders.

```
@startuml
title
<#lightblue,#red>|= Step |= Date |= Name |= Status |= Link |
<#lightgreen>| 1.1 | TBD | plantuml news |<#Navy><color:OrangeRed><b> Unknown | [[https://plantu
end title
@enduml
```

Step	Date	Name	Status	Link
1.1	TBD	plantuml news	Unknown	plantuml news

[Ref. QA-7184]

### 22.8.4 No border or same color as the background

You can also set the border color to the same color as the background.

```
@startuml
node n
note right of n
<#FBFB77,&#FBFB77>|= Husky / Yorkie |= Foo |
| SourceTree1 | foo1 |
| ST2 | foo2 |
end note
```



```
@enduml
```



[Ref. QA-12448]

### 22.8.5 Bold header or not

= as the first char of a cell indicates whether to make it bold (usually used for headers), or not.

```
@startuml
```

```
note as deepCSS0
```

```
|<#white> Husky / Yorkie |
|= <#gainsboro> SourceTree0 |
```

```
endnote
```

```
note as deepCSS1
```

```
|= <#white> Husky / Yorkie |= Foo |
|<#gainsboro><r> SourceTree1 | foo1 |
```

```
endnote
```

```
note as deepCSS2
```

```
|= Husky / Yorkie |
|<#gainsboro> SourceTree2 |
```

```
endnote
```

```
note as deepCSS3
```

```
<#white>|= Husky / Yorkie |= Foo |
|<#gainsboro> SourceTree1 | foo1 |
```

```
endnote
```

```
@enduml
```



[Ref. QA-10923]

## 22.9 Tree

You can use |\_ characters to build a tree.

On common commands, like title:

```
@startuml
```

```
skinparam titleFontSize 14
```

```
title
```

```
Example of Tree
```

```
|_ First line
```

```
|_ **Bom (Model)**
```

```
|_ prop1
```

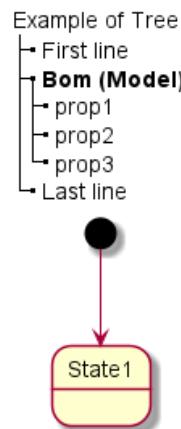
```
|_ prop2
```

```
|_ prop3
```

```
|_ Last line
```



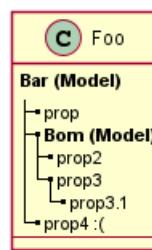
```
end title
[*] --> State1
@enduml
```



On Class diagram.

(Please note how we have to use an empty second compartment, else the parentheses in **(Model)** cause that text to be moved to a separate first compartment):

```
@startuml
class Foo {
**Bar (Model)**
|_ prop
|_ **Bom (Model)**
|_ prop2
|_ prop3
|_ prop3.1
|_ prop4 :(
-- 
}
@enduml
```



[Ref. QA-3448]

On Component or Deployment diagrams:

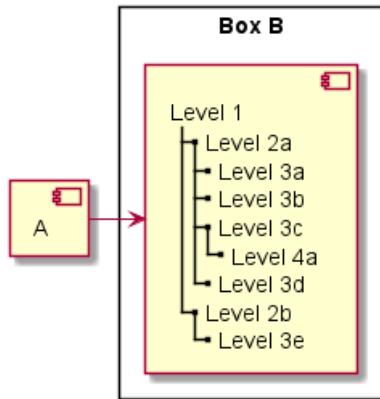
```
@startuml
[A] as A
rectangle "Box B" {
  component B [
    Level 1
    |_ Level 2a
    |_ Level 3a
    |_ Level 3b
    |_ Level 3c
    |_ Level 4a
    |_ Level 3d
    |_ Level 2b
  ]
}
```



```

|_ Level 3e
]
}
A -> B
@enduml

```



[Ref. QA-11365]

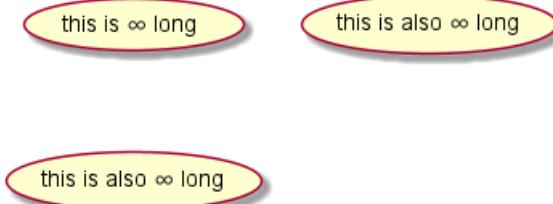
## 22.10 Special characters

It's possible to use any unicode character, either directly or with syntax &#XXX or <U+XXXX>:

```

@startuml
usecase direct as "this is ☺ long"
usecase ampHash as "this is also &#8734; long"
usecase angleBrackets as "this is also <U+221E> long"
@enduml

```



## 22.11 OpenIconic

OpenIconic is a very nice open-source icon set. Those icons are integrated in the creole parser, so you can use them out-of-the-box.

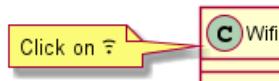
Use the following syntax: <&ICON\_NAME>.

```

@startuml
title: <size:20><&heart>Use of OpenIconic<&heart></size>
class Wifi
note left
    Click on <&wifi>
end note
@enduml

```

♥Use of OpenIconic♥



The complete list is available at the OpenIconic Website, or you can use the following special command to list them:

```
@startuml
listopeniconic
@enduml
```

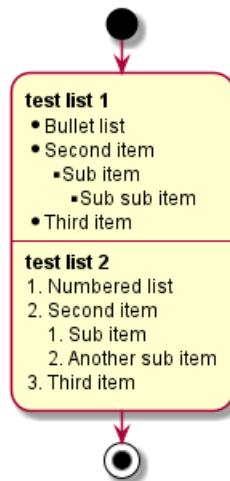
<b>List Open Iconic</b>	<b>bell</b>	<b>cloud</b>	<b>excerpt</b>	<b>justify-right</b>	<b>musical-note</b>	<b>star</b>
<i>Credit to</i>						
<a href="https://useiconic.com/open">https://useiconic.com/open</a>						
<b>B bold</b>						

## 22.12 Appendix: Examples of "Creole List" on all diagrams

### 22.12.1 Activity

```
@startuml
start
:***test list 1**
* Bullet list
* Second item
** Sub item
*** Sub sub item
* Third item
-----
**test list 2**
# Numbered list
# Second item
## Sub item
## Another sub item
# Third item;
stop
@enduml
```





### 22.12.2 Class

**TODO:** FIXME

- *Sub item*
- *Sub sub item*

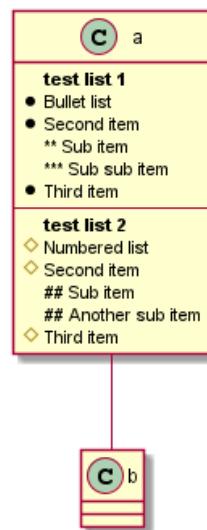
**TODO:** FIXME

@startuml

```
class a {  
**test list 1**  
* Bullet list  
* Second item  
** Sub item  
*** Sub sub item  
* Third item  
----  
**test list 2**  
# Numbered list  
# Second item  
## Sub item  
## Another sub item  
# Third item  
}  
  
a -- b
```

@enduml





### 22.12.3 Component, Deployment, Use-Case

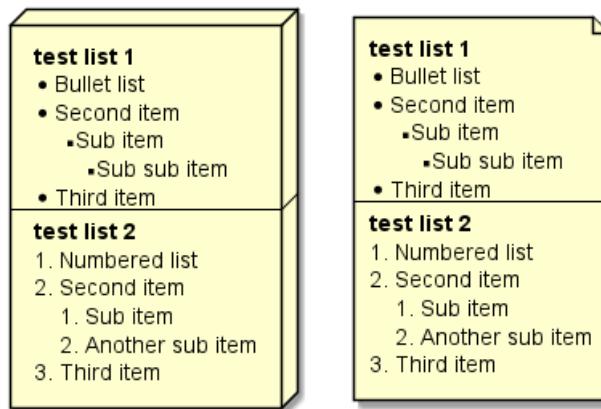
```

@startuml
node n [
**test list 1**
* Bullet list
* Second item
** Sub item
*** Sub sub item
* Third item
-----
**test list 2**
# Numbered list
# Second item
## Sub item
## Another sub item
# Third item
]

file f as "
**test list 1**
* Bullet list
* Second item
** Sub item
*** Sub sub item
* Third item
-----
**test list 2**
# Numbered list
# Second item
## Sub item
## Another sub item
# Third item
"
@enduml

```





**TODO:** DONE [Corrected in V1.2020.18]

#### 22.12.4 Gantt project planning

N/A

#### 22.12.5 Object

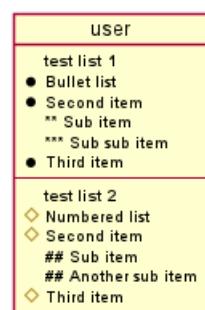
**TODO:** FIXME

- *Sub item*
- *Sub sub item*

**TODO:** FIXME

```
@startuml
object user {
**test list 1**
* Bullet list
* Second item
** Sub item
*** Sub sub item
* Third item
-----
**test list 2**
# Numbered list
# Second item
## Sub item
## Another sub item
# Third item
}
```

@enduml

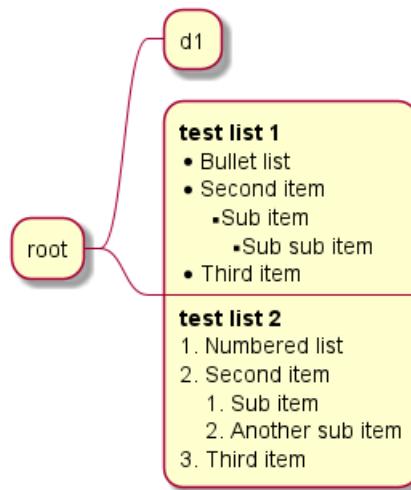


### 22.12.6 MindMap

```
@startmindmap
```

```
* root
** d1
***:***test list 1**
* Bullet list
* Second item
** Sub item
*** Sub sub item
* Third item
-----
**test list 2**
# Numbered list
# Second item
## Sub item
## Another sub item
# Third item;
```

```
@endmindmap
```



### 22.12.7 Network (nwdiag)

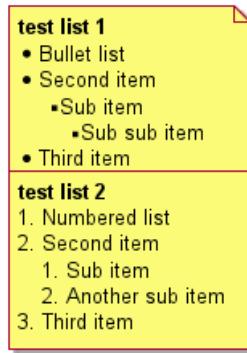
N/A

### 22.12.8 Note

```
@startuml
note as n
**test list 1**
* Bullet list
* Second item
** Sub item
*** Sub sub item
* Third item
-----
**test list 2**
# Numbered list
# Second item
## Sub item
```



```
## Another sub item
# Third item
end note
@enduml
```



### 22.12.9 Sequence

N/A (*or on note or common commands*)

### 22.12.10 State

N/A (*or on note or common commands*)

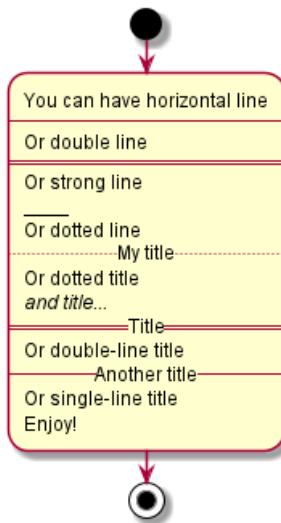
## 22.13 Appendix: Examples of "Creole horizontal lines" on all diagrams

### 22.13.1 Activity

**TODO:** FIXME strong line ---- **TODO:** FIXME

```
@startuml
start
:You can have horizontal line
-----
Or double line
=====
Or strong line
-----
Or dotted line
..My title..
Or dotted title
//and title... //
==Title==
Or double-line title
--Another title--
Or single-line title
Enjoy!;
stop
@enduml
```





### 22.13.2 Class

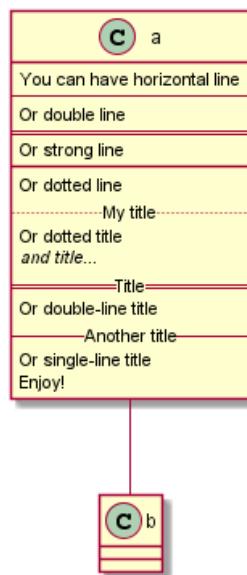
```
@startuml
```

```
class a {
    You can have horizontal line
    ----
    Or double line
    -----
    Or strong line
    ----
    Or dotted line
    ..My title..
    Or dotted title
    //and title... //
    ==Title==
    Or double-line title
    --Another title--
    Or single-line title
    Enjoy!
}
```

```
a -- b
```

```
@enduml
```





### 22.13.3 Component, Deployment, Use-Case

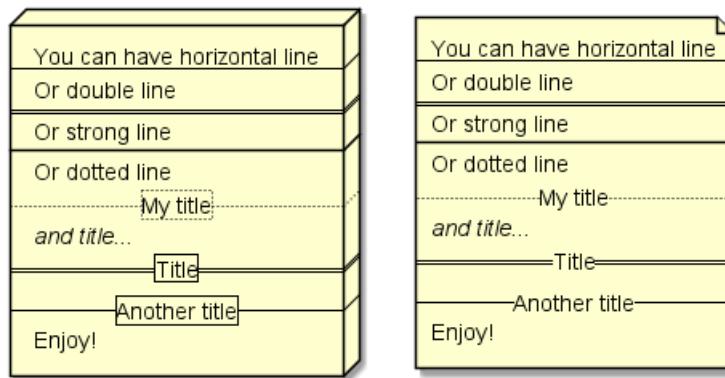
```

@startuml
node n [
You can have horizontal line
-----
Or double line
=====
Or strong line
-----
Or dotted line
..My title..
//and title... //
==Title==
--Another title--
Enjoy!
]

file f as "
You can have horizontal line
-----
Or double line
=====
Or strong line
-----
Or dotted line
..My title..
//and title... //
==Title==
--Another title--
Enjoy!
"
@enduml

```





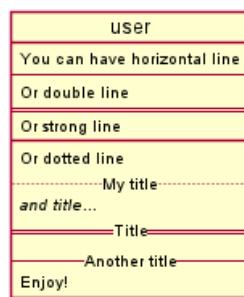
#### 22.13.4 Gantt project planning

N/A

#### 22.13.5 Object

```
@startuml
object user {
    You can have horizontal line
    ----
    Or double line
    ----
    Or strong line
    ----
    Or dotted line
    ..My title..
    //and title... //
    ==Title==
    --Another title--
    Enjoy!
}
```

@enduml



**TODO:** DONE [Corrected on V1.2020.18]

#### 22.13.6 MindMap

**TODO:** FIXME strong line ---- **TODO:** FIXME

@startmindmap

```
* root
** d1
**:You can have horizontal line
----
```



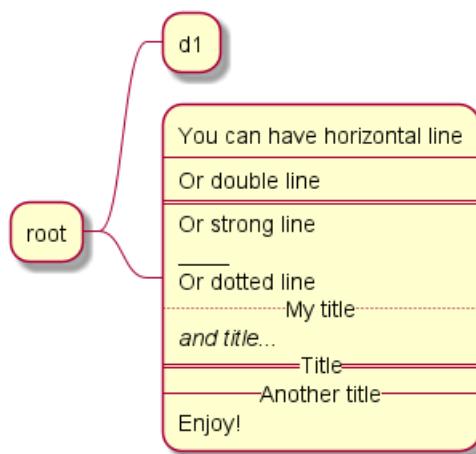
```
Or double line
=====
```

```
Or strong line
=====
```

```
Or dotted line
```

```
..My title...
//and title... //
==Title==
--Another title--
Enjoy!;
```

```
@endmindmap
```



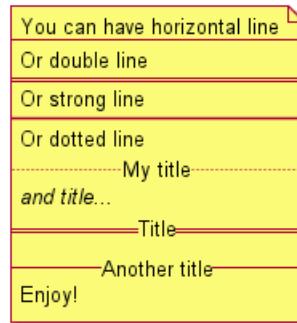
### 22.13.7 Network (nwdiag)

N/A

### 22.13.8 Note

```
@startuml
note as n
You can have horizontal line
=====
Or double line
=====
Or strong line
=====
Or dotted line
..My title...
//and title... //
==Title==
--Another title--
Enjoy!
end note
@enduml
```





### 22.13.9 Sequence

N/A (or on note or common commands)

### 22.13.10 State

N/A (or on note or common commands)

## 22.14 Style equivalent (between Creole and HTML)

Style	Creole	Legacy HTML like
<b>bold</b>	This is **bold**	This is <b>bold</b>
<i>italics</i>	This is //italics//	This is <i>italics</i>
<u>monospaced</u>	This is ""monospaced""	This is <font:monospaced>monospaced</font>
<u>stroked</u>	This is --stroked--	This is <s>stroked</s>
<u>underlined</u>	This is __underlined__	This is <u>underlined</u>
waved	This is ~~~	This is <w>waved</w>

```

@startmindmap
* Style equivalent\n(between Creole and HTML)
**:**Creole**
-----
<#silver>|= code|= output
| \n This is ""~**bold**""\n | \n This is **bold** |
| \n This is ""~//italics//""\n | \n This is //italics// |
| \n This is ""~"monospaced~"" ""\n | \n This is ""monospaced"" |
| \n This is ""~~stroked--"\n | \n This is --stroked-- |
| \n This is ""~__underlined__""\n | \n This is __underlined__ |
| \n This is ""<U+007E><U+007E>waved<U+007E><U+007E>""\n | \n This is ~~waved~~ |
**:<b>Legacy HTML like
-----
<#silver>|= code|= output
| \n This is ""~<b>bold</b>""\n | \n This is <b>bold</b> |
| \n This is ""~<i>italics</i>""\n | \n This is <i>italics</i> |
| \n This is ""~<font:monospaced>monospaced</font>""\n | \n This is <font:monospaced>monospaced</font> |
| \n This is ""~<s>stroked</s>""\n | \n This is <s>stroked</s> |
| \n This is ""~<u>underlined</u>""\n | \n This is <u>underlined</u> |
| \n This is ""~<w>waved</w>""\n | \n This is <w>waved</w> |

And color as a bonus...
<#silver>|= code|= output
| \n This is ""~<s:color:green>""green""</color>"">stroked</s>""\n | \n This is <s:green>stroked</s> |
| \n This is ""~<u:color:red>""red""</color>"">underlined</u>""\n | \n This is <u:red>underlined</u> |
| \n This is ""~<w:#0000FF>""#0000FF""</color>"">waved</w>""\n | \n This is <w:#0000FF>waved</w>
@endmindmap
  
```



Style equivalent (between Creole and HTML)

Creole	
code	output
This is **bold**	This is <b>bold</b>
This is //italics//	This is <i>italics</i>
This is ""monospaced""	This is monospaced
This is --stroked--	This is <del>stroked</del>
This is __underlined__	This is <u>underlined</u>
This is ~~waved~~	This is <u>waved</u>

Legacy HTML like	
code	output
This is <b>bold</b>	This is <b>bold</b>
This is <i>italics</i>	This is <i>italics</i>
This is <font:monospaced>monospaced</font>	This is monospaced
This is <s>stroked</s>	This is <del>stroked</del>
This is <u>underlined</u>	This is <u>underlined</u>
This is <w>waved</w>	This is <u>waved</u>

And color as a bonus...

code		output
This is <s:green>stroked</s>	This is stroked	
This is <u:red>underlined</u>	This is underlined	
This is <w:#0000FF>waved</w>	This is <u>waved</u>	



## 23 Defining and using sprites

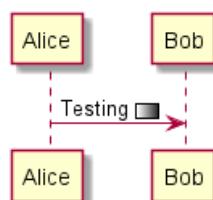
A *Sprite* is a small graphic element that can be used in diagrams.

In PlantUML, sprites are monochrome and can have either 4, 8 or 16 gray level.

To define a sprite, you have to use a hexadecimal digit between 0 and F per pixel.

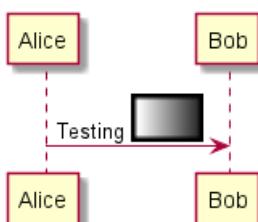
Then you can use the sprite using <\$XXX> where XXX is the name of the sprite.

```
@startuml
sprite $foo1 {
    FFFFFFFFFFFFFF
    F0123456789ABC
    F0123456789ABC
    F0123456789ABC
    F0123456789ABC
    F0123456789ABC
    F0123456789ABC
    F0123456789ABC
    F0123456789ABC
    F0123456789ABC
    FFFFFFFFFFFFFF
}
Alice -> Bob : Te
@enduml
```



You can scale the sprite.

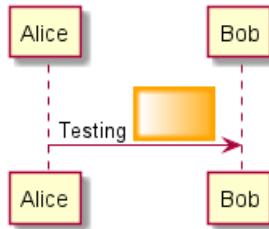
```
@startuml
sprite $foo1 {
    FFFFFFFFFFFFFF
    F0123456789ABCF
    F0123456789ABCF
    F0123456789ABCF
    F0123456789ABCF
    F0123456789ABCF
    F0123456789ABCF
    F0123456789ABCF
    F0123456789ABCF
    FFFFFFFFFFFFFF
}
Alice -> Bob : Testing <$foo1{scale=3}>
@enduml
```



## 23.1 Changing colors

Although sprites are monochrome, it's possible to change their color.

```
@startuml
sprite $foo1 {
    FFFFFFFFFFFFFF
    F0123456789ABCF
    FFFFFFFFFFFFFF
}
Alice -> Bob : Testing <$foo1,scale=3.4,color=orange>
@enduml
```



## 23.2 Encoding Sprite

To encode sprite, you can use the command line like:

```
java -jar plantuml.jar -encodesprite 16z foo.png
```

where `foo.png` is the image file you want to use (it will be converted to gray automatically).

After `-encodesprite`, you have to specify a format: `4`, `8`, `16`, `4z`, `8z` or `16z`.

The number indicates the gray level and the optional `z` is used to enable compression in sprite definition.

## 23.3 Importing Sprite

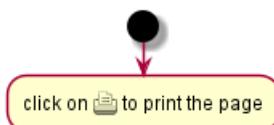
You can also launch the GUI to generate a sprite from an existing image.

Click in the menubar then on `File/Open Sprite Window`.

After copying an image into your clipboard, several possible definitions of the corresponding sprite will be displayed : you will just have to pickup the one you want.

## 23.4 Examples

```
@startuml
sprite $printer [15x15/8z] N0tH3WOW208HxFz_kMAhj7lHWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvI
start
:click on <$printer> to print the page;
@enduml
```



```

@startuml
sprite $bug [15x15/16z] PKzR2i0m2BFMi15p__FEjQEqB1z27aeqCqixa8S40T7C53cKpsHpaYPDJY_12MHM-BLRyywPhrr
sprite $printer [15x15/8z] N0tH3W0W208HxFz_kMAhj71HWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXw
sprite $disk {
    444445566677881
    4360000000009991
    436000000000ACA1
    53700000001A7A1
    53700000012B8A1
    53800000123B8A1
    63800001233C9A1
    634999AABC99B1
    744566778899AB1
    7456AAAAA99AAB1
    8566AFC228AABB1
    8567AC8118BBBB1
    867BD4433BBBB1
    39AAAAABBBBBBC1
}

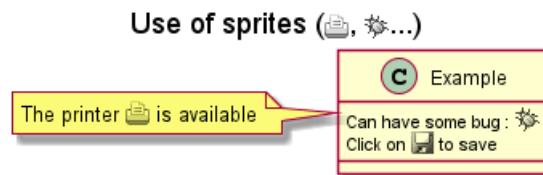
title Use of sprites (<$printer>, <$bug>...)

class Example {
    Can have some bug : <$bug>
    Click on <$disk> to save
}

note left : The printer <$printer> is available

@enduml

```



## 23.5 StdLib

The PlantUML StdLib includes a number of ready icons in various IT areas such as architecture, cloud services, logos etc. It including AWS, Azure, Kubernetes, C4, product Logos and many others. To explore these libraries:

- Browse the Github folders of PlantUML StdLib
- Browse the source repos of StdLib collections that interest you. Eg if you are interested in logos you can find that it came from gilbarbara-plantuml-sprites, and quickly find its

sprites-list. (The next section shows how to list selected sprites but unfortunately that's in grayscale whereas this custom listing is in color.)

- Study the in-depth Hitchhiker's Guide to PlantUML, eg sections Standard Library Sprites and PlantUML Stdlib Overview

## 23.6 Listing Sprites

You can use the `listsprites` command to show available sprites:

- Used on its own, it just shows ArchiMate sprites



- If you include some sprite libraries in your diagram, the command shows all these sprites, as explained in View all the icons with listsprites.

(Example from Hitchhikers Guide to PlantUML)

```
@startuml
```

```
!define osaPuml https://raw.githubusercontent.com/Crashedmind/PlantUML-opensecurityarchitecture2-icon/master/sprites.puml
!include osaPuml/Common.puml
!include osaPuml/User/all.puml
```

```
listsprites
@enduml
```



Most collections have files called `all` that allow you to see a whole sub-collection at once. Else you need to find the sprites that interest you and include them one by one. Unfortunately, the version of a collection included in StdLib often does not have such `all` files, so as you see above we include the collection from github, not from StdLib.

All sprites are in grayscale, but most collections define specific macros that include appropriate (vendor-specific) colors.

## 24 Skinparam command

You can change colors and font of the drawing using the `skinparam` command.

Example:

```
skinparam backgroundColor transparent
```

### 24.1 Usage

You can use this command :

- In the diagram definition, like any other commands,
- In an included file,
- In a configuration file, provided in the command line or the ANT task.

### 24.2 Nested

To avoid repetition, it is possible to nest definition. So the following definition :

```
skinparam xxxxParam1 value1
skinparam xxxxParam2 value2
skinparam xxxxParam3 value3
skinparam xxxxParam4 value4
```

is strictly equivalent to:

```
skinparam xxxx {
    Param1 value1
    Param2 value2
    Param3 value3
    Param4 value4
}
```

### 24.3 Black and White

You can force the use of a black&white output using `skinparam monochrome true` command.

```
@startuml
```

```
skinparam monochrome true

actor User
participant "First Class" as A
participant "Second Class" as B
participant "Last Class" as C

User -> A: DoWork
activate A

A -> B: Create Request
activate B

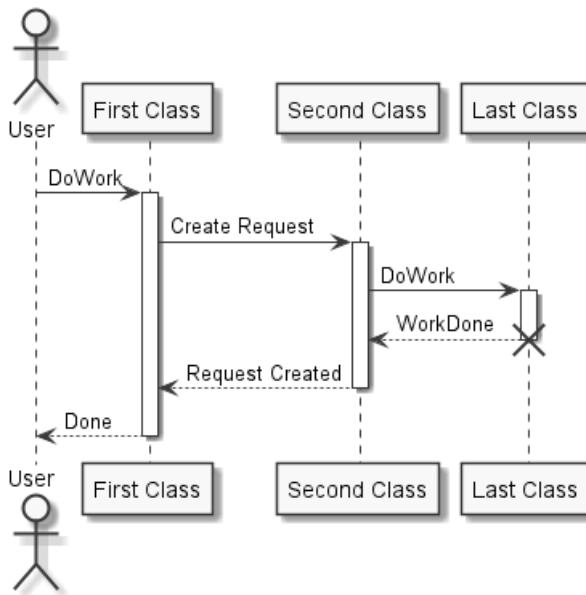
B -> C: DoWork
activate C
C --> B: WorkDone
destroy C

B --> A: Request Created
deactivate B
```



```
A --> User: Done
deactivate A

@enduml
```



## 24.4 Shadowing

You can disable the shadowing using the `skinparam shadowing false` command.

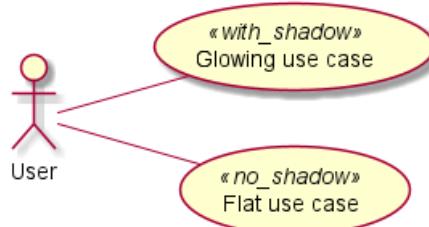
```
@startuml
```

left to right direction

```
skinparam shadowing<<no_shadow>> false
skinparam shadowing<<with_shadow>> true

actor User
(Glowing use case) <<with_shadow>> as guc
(Flat use case) <<no_shadow>> as fuc
User -- guc
User -- fuc
```

```
@enduml
```



## 24.5 Reverse colors

You can force the use of a black&white output using `skinparam monochrome reverse` command. This can be useful for black background environment.

```
@startuml
```

```
skinparam monochrome reverse
```



```
actor User
participant "First Class" as A
participant "Second Class" as B
participant "Last Class" as C
```

```
User -> A: DoWork
activate A
```

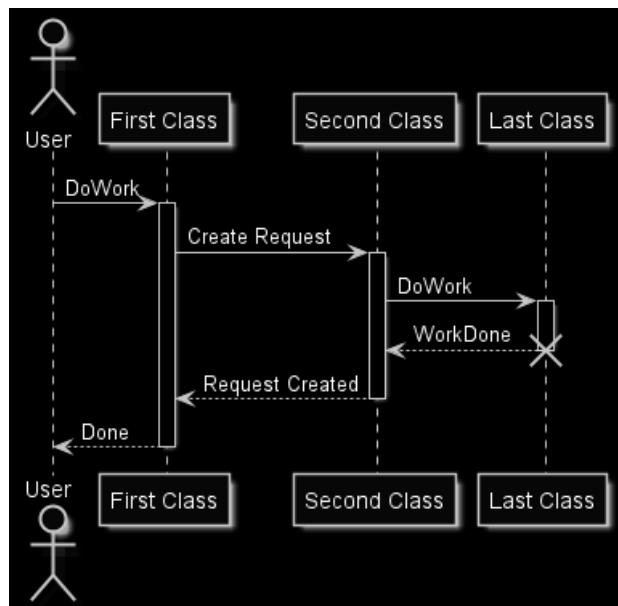
```
A -> B: Create Request
activate B
```

```
B -> C: DoWork
activate C
C --> B: WorkDone
destroy C
```

```
B --> A: Request Created
deactivate B
```

```
A --> User: Done
deactivate A
```

```
@enduml
```



## 24.6 Colors

You can use either standard color name or RGB code.

```
@startuml
colors
@enduml
```



APPLICATION	Crimson	DeepPink	Indigo	LightYellow	Navy	RoyalBlue	Turquoise
AliceBlue	Cyan	DeepSkyBlue	Ivory	Lime	OldLace	STRATEGY	Violet
AntiqueWhite	DarkBlue	DimGray	Khaki	LimeGreen	Olive	SaddleBrown	Wheat
Aqua	DarkCyan	DimGrey	Lavender	Linen	OliveDrab	Salmon	White
Aquamarine	DarkGoldenRod	DodgerBlue	LavenderBlush	MOTIVATION	Orange	SandyBrown	WhiteSmoke
Azure	DarkGray	FireBrick	LawnGreen	Magenta	OrangeRed	SeaGreen	Yellow
BUSINESS	DarkGreen	FloralWhite	LemonChiffon	Maroon	Orchid	SeaShell	YellowGreen
Beige	DarkGrey	ForestGreen	LightBlue	MediumAquaMarine	PHYSICAL	Sienna	
Bisque	DarkKhaki	Fuchsia	LightCoral	MediumBlue	PaleGoldenRod	Silver	
Black	DarkMagenta	Gainsboro	LightCyan	MediumOrchid	PaleGreen	SkyBlue	
BlanchedAlmond	DarkOliveGreen	GhostWhite	LightGoldenRodYellow	MediumPurple	PaleTurquoise	SlateBlue	
Blue	DarkOrchid	Gold	LightGray	MediumSeaGreen	PaleVioletRed	SlateGray	
BlueViolet	DarkRed	GoldenRod	LightGreen	MediumSlateBlue	PapayaWhip	SlateGrey	
Brown	DarkSalmon	Gray	LightGrey	MediumSpringGreen	PeachPuff	Snow	
BurlyWood	DarkSeaGreen	Green	LightPink	MediumTurquoise	Peru	SpringGreen	
CadetBlue	DarkSlateBlue	GreenYellow	LightSalmon	MediumVioletRed	Pink	SteelBlue	
Chartreuse	DarkSlateGray	Grey	LightSeaGreen	MidnightBlue	Plum	TECHNOLOGY	
Chocolate	DarkSlateGrey	HoneyDew	LightSkyBlue	MintCream	PowderBlue	Tan	
Coral	DarkTurquoise	HotPink	LightSlateGray	MistyRose	Purple	Teal	
CornflowerBlue	DarkViolet	IMPLEMENTATION	LightSlateGrey	Moccasin	Red	Thistle	
Cornsilk	DarkOrange	IndianRed	LightSteelBlue	NavajoWhite	RosyBrown	Tomato	

transparent can only be used for background of the image.

## 24.7 Font color, name and size

You can change the font for the drawing using `xxxFontColor`, `xxxFontSize` and `xxxFontName` parameters.

Example:

```
skinparam classFontColor red
skinparam classFontSize 10
skinparam classFontName Aapex
```

You can also change the default font for all fonts using `skinparam defaultFontName`.

Example:

```
skinparam defaultFontName Aapex
```

Please note the fontname is highly system dependent, so do not over use it, if you look for portability. `Helvetica` and `Courier` should be available on all system.

A lot of parameters are available. You can list them using the following command:

```
java -jar plantuml.jar -language
```

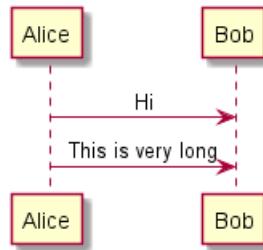
## 24.8 Text Alignment

Text alignment can be set up to `left`, `right` or `center`. You can also use `direction` or `reverseDirection` values for `sequenceMessageAlign` which align text depending on arrow direction.

Param name	Default value	Comment
sequenceMessageAlign	left	Used for messages in sequence diagrams
sequenceReferenceAlign	center	Used for <code>ref</code> over in sequence diagrams

```
@startuml
skinparam sequenceMessageAlign center
Alice -> Bob : Hi
Alice -> Bob : This is very long
@enduml
```





## 24.9 Examples

```
@startuml
skinparam backgroundColor #EEEBDC
skinparam handwritten true
```

```
skinparam sequence {
ArrowColor DeepSkyBlue
ActorBorderColor DeepSkyBlue
LifeLineBorderColor blue
LifeLineBackgroundColor #A9DCDF

ParticipantBorderColor DeepSkyBlue
ParticipantBackgroundColor DodgerBlue
ParticipantFontName Impact
ParticipantFontSize 17
ParticipantFontColor #A9DCDF
```

```
ActorBackgroundColor aqua
ActorFontColor DeepSkyBlue
ActorFontSize 17
ActorFontName Aapex
}
```

```
actor User
participant "First Class" as A
participant "Second Class" as B
participant "Last Class" as C
```

```
User -> A: DoWork
activate A
```

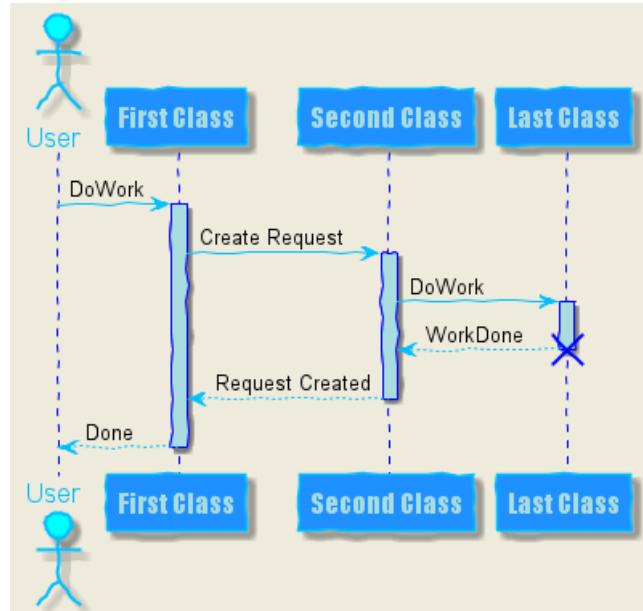
```
A -> B: Create Request
activate B
```

```
B -> C: DoWork
activate C
C --> B: WorkDone
destroy C
```

```
B --> A: Request Created
deactivate B
```

```
A --> User: Done
deactivate A
@enduml
```





```

@startuml
skinparam handwritten true

skinparam actor {
BorderColor black
FontName Courier
    BackgroundColor<< Human >> Gold
}

skinparam usecase {
BackgroundColor DarkSeaGreen
BorderColor DarkSlateGray

BackgroundColor<< Main >> YellowGreen
BorderColor<< Main >> YellowGreen

ArrowColor Olive
}

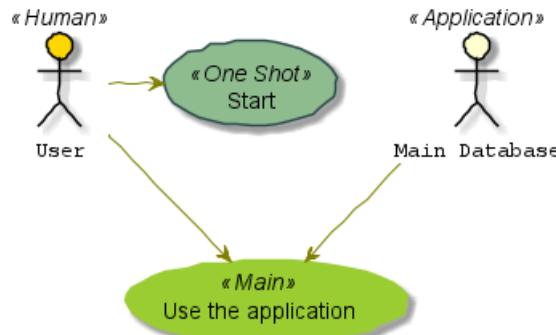
User << Human >>
:Main Database: as MySql << Application >>
(Start) << One Shot >>
(Use the application) as (Use) << Main >>

User -> (Start)
User --> (Use)

MySql --> (Use)
@enduml

```





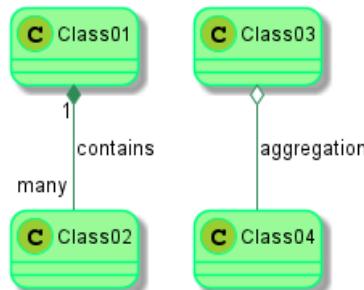
```

@startuml
skinparam roundcorner 20
skinparam class {
BackgroundColor PaleGreen
ArrowColor SeaGreen
BorderColor SpringGreen
}
skinparam stereotypeCBackgroundColor YellowGreen

Class01 "1" *-- "many" Class02 : contains

Class03 o-- Class04 : aggregation
@enduml

```



```

@startuml
skinparam interface {
    backgroundColor RosyBrown
    borderColor orange
}

skinparam component {
    FontSize 13
    backgroundColor<<Apache>> LightCoral
    borderColor<<Apache>> #FF6655
    FontName Courier
    BorderColor black
    backgroundColor gold
    ArrowFontName Impact
    ArrowColor #FF6655
    ArrowFontColor #777777
}

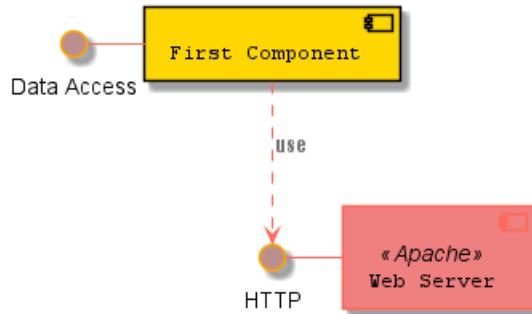
() "Data Access" as DA
[Web Server] << Apache >>

DA - [First Component]

```



```
[First Component] ..> () HTTP : use
HTTP - [Web Server]
@enduml
```

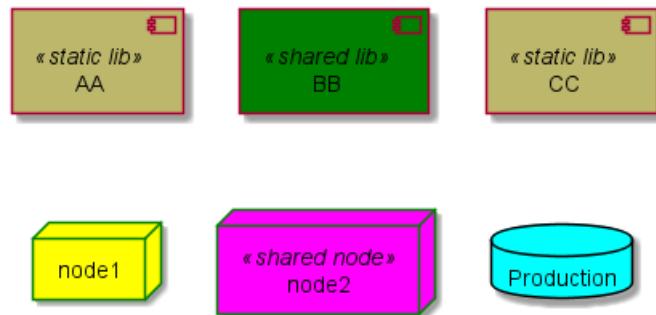


```
@startuml
[AA] <<static lib>>
[BB] <<shared lib>>
[CC] <<static lib>>

node node1
node node2 <<shared node>>
database Production

skinparam component {
    backgroundColor<<static lib>> DarkKhaki
    backgroundColor<<shared lib>> Green
}

skinparam node {
borderColor Green
backgroundColor Yellow
backgroundColor<<shared node>> Magenta
}
skinparam databaseBackgroundColor Aqua
@enduml
```



## 24.10 List of all skinparam parameters

Since the documentation is not always up to date, you can have the complete list of parameters using this command:

```
java -jar plantuml.jar -language
```

Or you can generate a "diagram" with a list of all the skinparam parameters using `help skinparams`.

That will give you the following result, from this page (*code of this command*):

- CommandHelpSkinparam.java

```
@startuml
```



```
help skinparams
@enduml
```



### Help on skinparam

The code of this command is located in `net.sourceforge.plantuml.help` package.

You may improve it on <https://github.com/plantuml/plantuml/tree/master/src/net/sourceforge/plantuml/help>

The possible skinparam are :

- ActivityBackgroundColor
- ActivityBarColor
- ActivityBorderColor
- ActivityBorderThickness
- ActivityDiamondBackgroundColor
- ActivityDiamondBorderColor
- ActivityDiamondFontColor
- ActivityDiamondFontName
- ActivityDiamondFontSize
- ActivityDiamondFontStyle
- ActivityEndColor
- ActivityFontColor
- ActivityFontName
- ActivityFontSize
- ActivityFontStyle
- ActivityStartColor
- ActorBackgroundColor
- ActorBorderColor
- ActorFontColor
- ActorFontName
- ActorFontSize
- ActorFontStyle
- ActorStereotypeFontColor
- ActorStereotypeFontName
- ActorStereotypeFontSize
- ActorStereotypeFontStyle
- AgentBackgroundColor
- AgentBorderColor
- AgentBorderThickness
- AgentFontColor
- AgentFontName
- AgentFontSize
- AgentFontStyle
- AgentStereotypeFontColor
- AgentStereotypeFontName
- AgentStereotypeFontSize
- AgentStereotypeFontStyle
- ArchimateBackgroundColor
- ArchimateBorderColor
- ArchimateBorderThickness
- ArchimateFontColor
- ArchimateFontName
- ArchimateFontSize
- ArchimateFontStyle
- ArchimateStereotypeFontColor
- ArchimateStereotypeFontName
- ArchimateStereotypeFontSize
- ArchimateStereotypeFontStyle
- ArrowColor
- ArrowFontColor
- ArrowFontName
- ArrowFontSize
- ArrowFontStyle
- ArrowHeadColor
- ArrowLollipopColor
- ArrowMessageAlignment
- ArrowThickness



You can also view each skinparam parameters with its results displayed at the page **All Skin Parameters of Ashley's PlantUML Doc**:

- <https://plantuml-documentation.readthedocs.io/en/latest/formatting/all-skin-params.html>.



## 25 Preprocessing

Some preprocessing capabilities are included in **PlantUML**, and available for *all* diagrams.

Those functionalities are very similar to the C language preprocessor, except that the special character # has been changed to the exclamation mark !.

### 25.1 Migration notes

The current preprocessor is an update from some legacy preprocessor.

Even if some legacy features are still supported with the actual preprocessor, you should not use them any more (they might be removed in some long term future).

- You should not use `!define` and `!definelong` anymore. Use `!function`, `!procedure` or variable definition instead.
  - `!define` should be replaced by return `!function`
  - `!definelong` should be replaced by `!procedure`.
- `!include` now allows multiple inclusions : you don't have to use `!include_many` anymore
- `!include` now accepts a URL, so you don't need `!includeurl`
- Some features (like `%date%`) have been replaced by builtin functions (for example `%date()`)
- When calling a legacy `!definelong` macro with no arguments, you do have to use parenthesis. You have to use `my_own_definelong()` because `my_own_definelong` without parenthesis is not recognized by the new preprocessor.

Please contact us if you have any issues.

### 25.2 Variable definition

Although this is not mandatory, we highly suggest that variable names start with a \$.

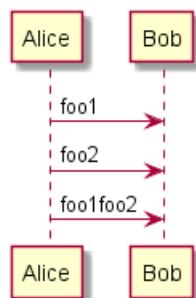
There are two types of data:

- **Integer number (int);**
- **String (str)** - these must be surrounded by single quote or double quote.

Variables created outside function are **global**, that is you can access them from everywhere (including from functions). You can emphasize this by using the optional **global** keyword when defining a variable.

```
@startuml
!$ab = "foo1"
!$cd = "foo2"
!$ef = $ab + $cd

Alice -> Bob : $ab
Alice -> Bob : $cd
Alice -> Bob : $ef
@enduml
```



## 25.3 Boolean expression

### 25.3.1 Boolean representation [0 is false]

There is not real boolean type, but PlantUML use this integer convention:

- Integer 0 means **false**
- and any non-null number (as 1) or any string (as "1", or even "0") means **true**.

[Ref. QA-9702]

### 25.3.2 Boolean operation and operator [&&, ||, ()]

You can use boolean expression, in the test, with :

- *parenthesis ()*;
- *and operator &&*;
- *or operator ||*.

(See next example, within *if* test.)

### 25.3.3 Boolean builtin functions [%false(), %true(), %not(<exp>) ]

For convenience, you can use those boolean builtin functions:

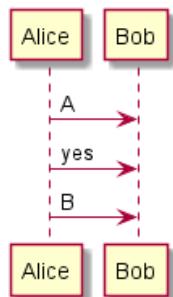
- `%false()`
- `%true()`
- `%not(<exp>)`

[See also *Builtin functions*]

## 25.4 Conditions [*!if*, *!else*, *!elseif*, *!endif*]

- You can use expression in condition.
- *else* and *elseif* are also implemented

```
@startuml
!$a = 10
!$ijk = "foo"
Alice -> Bob : A
!if ($ijk == "foo") && ($a+10>=4)
Alice -> Bob : yes
!else
Alice -> Bob : This should not appear
!endif
Alice -> Bob : B
@enduml
```

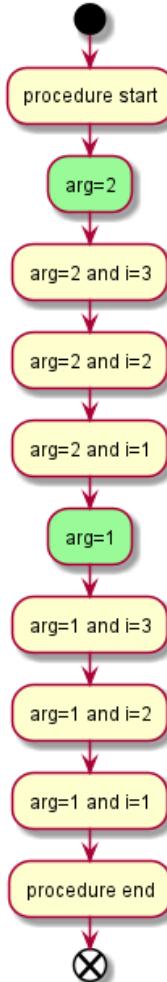


## 25.5 While loop [`!while`, `!endwhile`]

You can use `!while` and `!endwhile` keywords to have repeat loops.

```
@startuml
!procedure $foo($arg)
:procedure start;
!while $arg!=0
    !$i=3
    #palegreen:arg=$arg;
    !while $i!=0
        :arg=$arg and i==$i;
        !$i = $i - 1
    !endwhile
    !$arg = $arg - 1
!endwhile
:procedure end;
!endprocedure

start
$foo(2)
end
@enduml
```



[Adapted from QA-10838]

```
@startmindmap
!procedure $foo($arg)
```



```

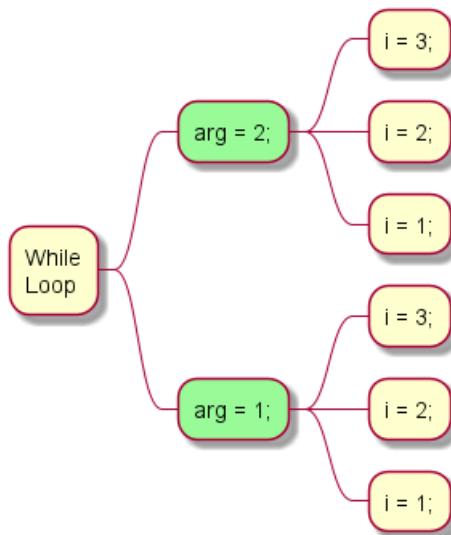
!while $arg!=0
 !$i=3
 **[#palegreen] arg = $arg;
 !while $i!=0
   *** i = $i;
   !$i = $i - 1
 !endwhile
 !$arg = $arg - 1
 !endwhile
!endprocedure

```

```

*:While
Loop;
$foo(2)
@endmindmap

```



## 25.6 Procedure [!procedure, !endprocedure]

- Procedure names *should* start with a \$
- Argument names *should* start with a \$
- Procedures can call other procedures

Example:

```

@startuml
!procedure $msg($source, $destination)
  $source --> $destination
!endprocedure

```

```

!procedure $init_class($name)
  class $name {
    $addCommonMethod()
  }
!endprocedure

```

```

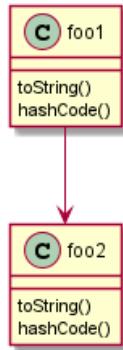
!procedure $addCommonMethod()
  toString()
  hashCode()

```



```
!endprocedure
```

```
$init_class("foo1")
)init_class("foo2")
$msg("foo1", "foo2")
@enduml
```



Variables defined in procedures are **local**. It means that the variable is destroyed when the procedure ends.

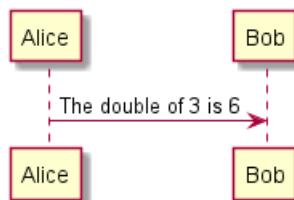
## 25.7 Return function [!function, !endfunction]

A return function does not output any text. It just define a function that you can call:

- directly in variable definition or in diagram text
- from other return functions
- from procedures
- Function name *should* start with a \$
- Argument names *should* start with a \$

```
@startuml
!function $double($a)
!return $a + $a
!endfunction
```

```
Alice -> Bob : The double of 3 is $double(3)
@enduml
```

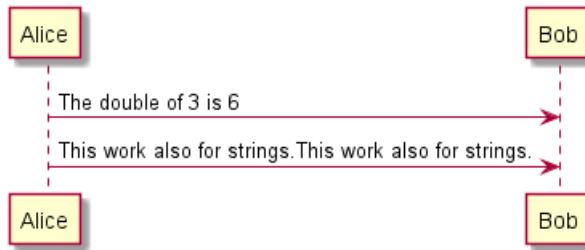


It is possible to shorten simple function definition in one line:

```
@startuml
!function $double($a) !return $a + $a

Alice -> Bob : The double of 3 is $double(3)
Alice -> Bob : $double("This work also for strings.")
@enduml
```





As in procedure (void function), variable are local by default (they are destroyed when the function is exited). However, you can access to global variables from function. However, you can use the `local` keyword to create a local variable if ever a global variable exists with the same name.

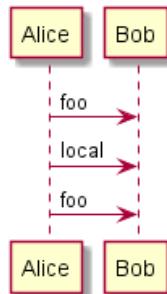
```

@startuml
!function $dummy()
!local $ijk = "local"
!return "Alice -> Bob : " + $ijk
!endfunction

!global $ijk = "foo"

Alice -> Bob : $ijk
$dummy()
Alice -> Bob : $ijk
@enduml

```



## 25.8 Default argument value

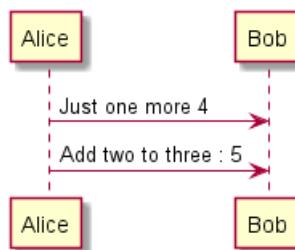
In both procedure and return functions, you can define default values for arguments.

```

@startuml
!function $inc($value, $step=1)
!return $value + $step
!endfunction

Alice -> Bob : Just one more $inc(3)
Alice -> Bob : Add two to three : $inc(3, 2)
@enduml

```

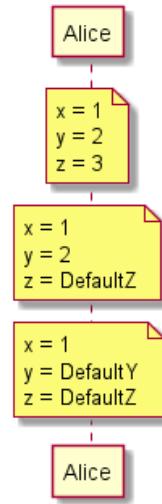


Only arguments at the end of the parameter list can have default values.



```
@startuml
!procedure defaulttest($x, $y="DefaultY", $z="DefaultZ")
note over Alice
    x = $x
    y = $y
    z = $z
end note
!endprocedure

defaulttest(1, 2, 3)
defaulttest(1, 2)
defaulttest(1)
@enduml
```

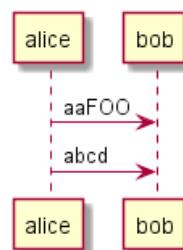


## 25.9 Unquoted procedure or function [!unquoted]

By default, you have to put quotes when you call a function or a procedure. It is possible to use the `unquoted` keyword to indicate that a function or a procedure does not require quotes for its arguments.

```
@startuml
!unquoted function id($text1, $text2="FOO") !return $text1 + $text2

alice -> bob : id(aa)
alice -> bob : id(ab,cd)
@enduml
```



## 25.10 Keywords arguments

Like in Python, you can use keywords arguments :

```
@startuml
```

```
!unquoted procedure $element($alias, $description="", $label="", $technology="", $size=12, $colour="g
rectangle $alias as "
```



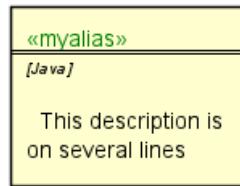
```

<color:$colour><<$alias>></color>
==$label==
//<size:$size>[$technology]</size>//

$description"
!endprocedure

$element(myalias, "This description is %newline()on several lines", $size=10, $technology="Java")
@enduml

```



## 25.11 Including files or URL [`!include`, `!include_many`, `!include_once`]

Use the `!include` directive to include file in your diagram. Using URL, you can also include file from Internet/Intranet.

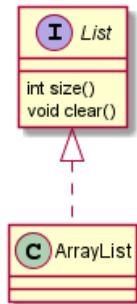
Imagine you have the very same class that appears in many diagrams. Instead of duplicating the description of this class, you can define a file that contains the description.

```
@startuml
```

```

interface List
List : int size()
List : void clear()
List <|.. ArrayList
@enduml

```



### File List.iuml

```

interface List
List : int size()
List : void clear()

```

The file `List.iuml` can be included in many diagrams, and any modification in this file will change all diagrams that include it.

You can also put several `@startuml/@enduml` text block in an included file and then specify which block you want to include adding `!0` where 0 is the block number. The `!0` notation denotes the first diagram.

For example, if you use `!include foo.txt!1`, the second `@startuml/@enduml` block within `foo.txt` will be included.

You can also put an id to some `@startuml/@enduml` text block in an included file using `@startuml(id=MY OWN_ID)` syntax and then include the block adding `!MY OWN_ID` when including the file, so using something like `!include foo.txt!MY OWN_ID`.



By default, a file can only be included once. You can use `!include_many` instead of `!include` if you want to include some file several times. Note that there is also a `!include_once` directive that raises an error if a file is included several times.

## 25.12 Including Subpart [`!startsub`, `!endsub`, `!includesub`]

You can also use `!startsub NAME` and `!endsub` to indicate sections of text to include from other files using `!includesub`. For example:

**file1.puml:**

```
@startuml

A -> A : stuff1
!startsub BASIC
B -> B : stuff2
!endsub
C -> C : stuff3
!startsub BASIC
D -> D : stuff4
!endsub
@enduml
```

`file1.puml` would be rendered exactly as if it were:

```
@startuml

A -> A : stuff1
B -> B : stuff2
C -> C : stuff3
D -> D : stuff4
@enduml
```

However, this would also allow you to have another `file2.puml` like this:

**file2.puml**

```
@startuml

title this contains only B and D
!includesub file1.puml!BASIC
@enduml
```

This file would be rendered exactly as if:

```
@startuml

title this contains only B and D
B -> B : stuff2
D -> D : stuff4
@enduml
```

## 25.13 Builtin functions [%]

Some functions are defined by default. Their name starts by %



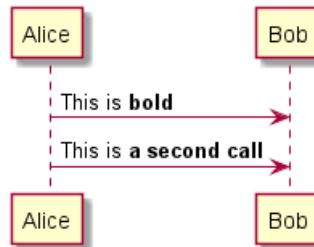
Name	Description	Example
%date	Retrieve current date. You can provide an optional format for the date	%date("yyyy.MM.dd")
%dirname	Retrieve current dirname	%dirname()
%false	Return always false	%false()
%file_exists	Check if a file exists on the local filesystem	%file_exists("c:/")
%filename	Retrieve current filename	%filename()
%function_exists	Check if a function exists	%function_exists()
%get_variable_value	Retrieve some variable value	%get_variable_value()
%getenv	Retrieve environment variable value	%getenv("OS")
%intval	Convert a String to Int	%intval("42")
%lower	Return a lowercase string	%lower("Hello")
%newline	Return a newline	%newline()
%not	Return the logical negation of an expression	%not(2+2==4)
%set_variable_value	Set a global variable	%set_variable_value()
%string	Convert an expression to String	%string(1 + 2)
%strlen	Calculate the length of a String	%strlen("foo")
%strpos	Search a substring in a string	%strpos("abcdef", "def")
%substr	Extract a substring. Takes 2 or 3 arguments	%substr("abcdef", 3, 3)
%true	Return always true	%true()
%upper	Return an uppercase string	%upper("Hello")
%variable_exists	Check if a variable exists	%variable_exists()
%version	Return PlantUML current version	%version()

## 25.14 Logging [!log]

You can use `!log` to add some log output when generating the diagram. This has no impact at all on the diagram itself. However, those logs are printed in the command line's output stream. This could be useful for debug purpose.

```
@startuml
!function bold($text)
 !$result = "<b>" + $text + "</b>"
 !log Calling bold function with $text. The result is $result
 !return $result
!endfunction

Alice -> Bob : This is bold("bold")
Alice -> Bob : This is bold("a second call")
@enduml
```



## 25.15 Memory dump [!memory\_dump]

You can use `!memory_dump` to dump the full content of the memory when generating the diagram. An optional string can be put after `!memory_dump`. This has no impact at all on the diagram itself. This could be useful for debug purpose.

```
@startuml
!function $inc($string)
 !$val = %intval($string)
 !log value is $val
!endfunction
```

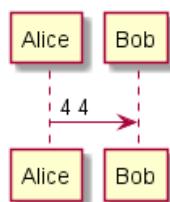


```

!dump_memory
!return $val+1
!endfunction

Alice -> Bob : 4 $inc("3")
!unused = "foo"
!dump_memory EOF
@enduml

```



## 25.16 Assertion [*!assert*]

You can put assertions in your diagram.

```

@startuml
Alice -> Bob : Hello
!assert %strpos("abcdef", "cd") == 3 : "This always fails"
@enduml

```

**Welcome to PlantUML!**

You can start with a simple UML Diagram like:

Bob->Alice: Hello

Or

class Example



You will find more information about PlantUML syntax on <https://plantuml.com>

(If you use this software, you accept its license)  
(Details by typing `license` keyword)

```

PlantUML 1.2021.3beta6
[From string (line 3) ]

@startuml
Alice -> Bob : Hello
!assert %strpos("abcdef", "cd") == 3 : "This always fails"
Assertion error : This always fails

```

## 25.17 Building custom library [*!import*, *!include*]

It's possible to package a set of included files into a single .zip or .jar archive. This single zip/jar can then be imported into your diagram using *!import* directive.

Once the library has been imported, you can *!include* file from this single zip/jar.

**Example:**

```

@startuml

!import /path/to/customLibrary.zip
' This just adds "customLibrary.zip" in the search path

!include myFolder/myFile.iuml

```



```
' Assuming that myFolder/myFile.iuml is located somewhere
' either inside "customLibrary.zip" or on the local filesystem
```

...

## 25.18 Search path

You can specify the java property `plantuml.include.path` in the command line.

For example:

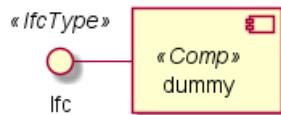
```
java -Dplantuml.include.path="c:/mydir" -jar plantuml.jar atest1.txt
```

Note the this -D option has to put before the -jar option. -D options after the -jar option will be used to define constants within plantuml preprocessor.

## 25.19 Argument concatenation [##]

It is possible to append text to a macro argument using the `##` syntax.

```
@startuml
!unquoted procedure COMP_TEXTGENCOMP(name)
[name] << Comp >>
interface Ifc << IfcType >> AS name##Ifc
name##Ifc - [name]
!endprocedure
COMP_TEXTGENCOMP(dummy)
@enduml
```



## 25.20 Dynamic invocation [%invoke\_procedure(), %call\_user\_func()]

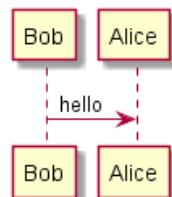
You can dynamically invoke a procedure using the special `%invoke_procedure()` procedure. This procedure takes as first argument the name of the actual procedure to be called. The optional following arguments are copied to the called procedure.

For example, you can have:

```
@startuml
!procedure $go()
Bob -> Alice : hello
!endprocedure

!$wrapper = "$go"

%invoke_procedure($wrapper)
@enduml
```



```
@startuml
!procedure $go($txt)
Bob -> Alice : $txt
```

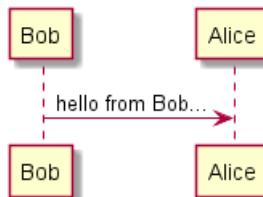


```

!endprocedure

%invoke_procedure("$go", "hello from Bob...")
@enduml

```



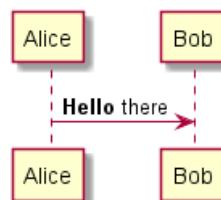
For return functions, you can use the corresponding special function `%call_user_func()` :

```

@startuml
!function bold($text)
!return "<b>" + $text + "</b>"
!endfunction

Alice -> Bob : %call_user_func("bold", "Hello") there
@enduml

```



## 25.21 Evaluation of addition depending of data types [+]

Evaluation of `$a + $b` depending of type of `$a` or `$b`

```

@startuml
title
<#LightBlue>|= |= $a |= $b |= <U+0025>string($a + $b) |
<#LightGray>| type | str | str | str (concatenation) |
| example |= "a" |= "b" |= %string("a" + "b") |
<#LightGray>| type | str | int | str (concatenation) |
| ex.|= "a" |= 2 |= %string("a" + 2) |
<#LightGray>| type | str | int | str (concatenation) |
| ex.|= 1 |= "b" |= %string(1 + "b") |
<#LightGray>| type | bool | str | str (concatenation) |
| ex.|= <U+0025>true() |= "b" |= %string(%true() + "b") |
<#LightGray>| type | str | bool | str (concatenation) |
| ex.|= "a" |= <U+0025>false() |= %string("a" + %false()) |
<#LightGray>| type | int | int | int (addition of int) |
| ex.|= 1 |= 2 |= %string(1 + 2) |
<#LightGray>| type | bool | int | int (addition) |
| ex.|= <U+0025>true() |= 2 |= %string(%true() + 2) |
<#LightGray>| type | int | bool | int (addition) |
| ex.|= 1 |= <U+0025>false() |= %string(1 + %false()) |
<#LightGray>| type | int | int | int (addition) |
| ex.|= 1 |= <U+0025>intval("2") |= %string(1 + %intval("2")) |
end title
@enduml

```



	<b>\$a</b>	<b>\$b</b>	<b>%string(\$a + \$b)</b>
<b>type</b>	str	str	str (concatenation)
<b>example</b>	"a"	"b"	ab
<b>type</b>	str	int	str (concatenation)
<b>ex.</b>	"a"	2	a2
<b>type</b>	str	int	str (concatenation)
<b>ex.</b>	1	"b"	1b
<b>type</b>	bool	str	str (concatenation)
<b>ex.</b>	%true()	"b"	1b
<b>type</b>	str	bool	str (concatenation)
<b>ex.</b>	"a"	%false()	a0
<b>type</b>	int	int	int (addition of int)
<b>ex.</b>	1	2	3
<b>type</b>	bool	int	int (addition)
<b>ex.</b>	%true()	2	3
<b>type</b>	int	bool	int (addition)
<b>ex.</b>	1	%false()	1
<b>type</b>	int	int	int (addition)
<b>ex.</b>	1	%intval("2")	3

## 25.22 Preprocessing JSON

You can extend the functionality of the current Preprocessing with JSON Preprocessing features:

- JSON Variable definition
- Access to JSON data
- Loop over JSON array

(See more details on *Preprocessing-JSON page*)



## 26 Unicode

The PlantUML language use *letters* to define actor, usecase and soon.

But *letters* are not only A-Z latin characters, it could be *any kind of letter from any language*.

### 26.1 Examples

```
@startuml
skinparam handwritten true
skinparam backgroundColor #EEEBDC

actor 使用者
participant "頭等艙" as A
participant "第二類" as B
participant "最後一堂課" as 別的東西

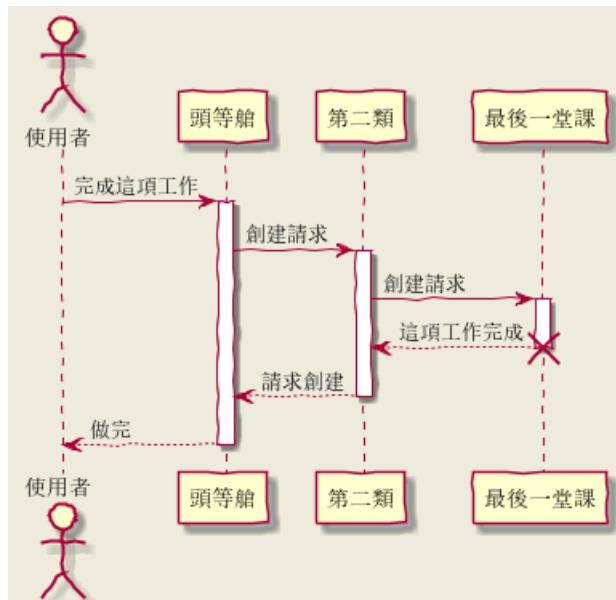
使用者 -> A: 完成這項工作
activate A

A -> B: 創建請求
activate B

B -> 別的東西: 創建請求
activate 別的東西
別的東西 --> B: 這項工作完成
destroy 別的東西

B --> A: 請求創建
deactivate B

A --> 使用者: 做完
deactivate A
@enduml
```

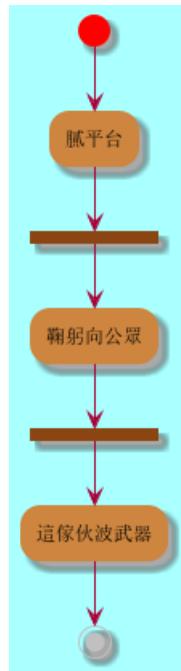


```
@startuml
(*) --> "膩平台"
--> === S1 ===
--> 鞠躬向公眾
```



```
--> === S2 ===
--> 這傢伙波武器
--> (*)
```

```
skinparam backgroundColor #AFFFFF
skinparam activityStartColor red
skinparam activityBarColor SaddleBrown
skinparam activityEndColor Silver
skinparam activityBackgroundColor Peru
skinparam activityBorderColor Peru
@enduml
```



```
@startuml

skinparam usecaseBackgroundColor DarkSeaGreen
skinparam usecaseArrowColor Olive
skinparam actorBorderColor black
skinparam usecaseBorderColor DarkSlateGray
```

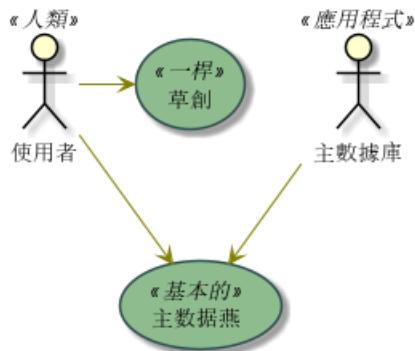
使用者 << 人類 >>  
 "主數據庫" as 數據庫 << 應用程式 >>  
 (草創) << 一桿 >>  
 "主数据燕" as (贏余) << 基本的 >>

使用者 -> (草創)  
 使用者 --> (贏余)

數據庫 --> (贏余)

@enduml

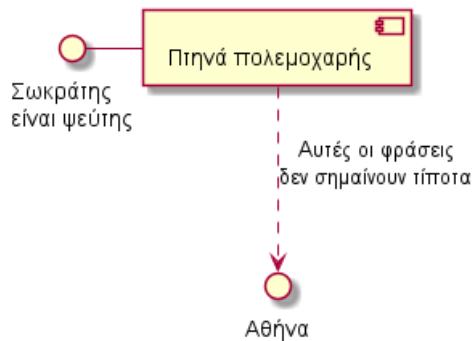




@startuml

```

() "Σ" as Σ
Σ - [Π]
[Π] ..> () A : A
@enduml
  
```



## 26.2 Charset

The default charset used when *reading* the text files containing the UML text description is system dependent.

Normally, it should just be fine, but in some case, you may want to use another charset. For example, with the command line:

```
java -jar plantuml.jar -charset UTF-8 files.txt
```

Or, with the ant task:

```
<!-- Put images in c:/images directory -->
<target name="main">
<plantuml dir=".src" charset="UTF-8" />
```

Depending of your Java installation, the following charset should be available: ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, UTF-16.



## 27 Standard Library

This page explains the official Standard Library (`stdlib`) for PlantUML. This Standard Library is now included in official releases of PlantUML. Including files follows the C convention for "C standard library".

Contents of the library come from third party contributors. We thank them for their useful contribution!

### 27.1 List of Standard Library

You can list standard library folders using the special diagram:

```
@startuml
stdlib
@enduml
```

```
archimate
Version 0.0.1
Delivered by https://github.com/ebbypeter/Archimate-PlantUML

aws
Version 18.02.22
Delivered by https://github.com/milo-minderbinder/AWS-PlantUML

awslib
Version 7.0.0
Delivered by https://github.com/awslabs/aws-icons-for-plantuml

azure
Version 2.1.0
Delivered by https://github.com/RicardoNiepel/Azure-PlantUML

c4
Version 2.0.0
Delivered by https://github.com/RicardoNiepel/C4-PlantUML

cloudinsight
Version 1.0.0
Delivered by https://github.com/rabelenda/cicon-plantuml-sprites/

cloudogu
Version 1.0.2
Delivered by https://github.com/cloudogu/plantuml-cloudogu-sprites

elastic
Version 0.0.1
Delivered by https://github.com/Crashedmind/PlantUML-Elastic-icons

kubernetes
Version 5.3.45
Delivered by https://github.com/michiel/plantuml-kubernetes-sprites

logos
Version 1.0.0
Delivered by https://github.com/rabelenda/gilbarbara-plantuml-sprites

material
Version 0.0.1
Delivered by https://github.com/Templarian/MaterialDesign

office
Version 1.0.0
Delivered by https://github.com/Roemer/plantuml-office

osa
Version 0.0.1
Delivered by https://github.com/Crashedmind/PlantUML-opensecurityarchitecture-icons

tupadr3
Version 2.2.0
Delivered by https://github.com/tupadr3/plantuml-icon-font-sprites
```



It is also possible to use the command line `java -jar plantuml.jar -stdlib` to display the same list.



Finally, you can extract the full standard library sources using `java -jar plantuml.jar -extractstdlib`. All files will be extracted in the folder `stdlib`.

Sources used to build official PlantUML releases are hosted here <https://github.com/plantuml/plantuml-stdlib>. You can create Pull Request to update or add some library if you find it relevant.

## 27.2 ArchiMate [archimate]

- <https://github.com/plantuml/plantuml-stdlib/tree/master/archimate>
- <https://github.com/ebbypeter/Archimate-PlantUML>

This repository contains ArchiMate PlantUML macros and other includes for creating Archimate Diagrams easily and consistently.

```
@startuml
!include <archimate/Archimate>

title Archimate Sample - Internet Browser

' Elements
Business_Object(businessObject, "A Business Object")
Business_Process(someBusinessProcess,"Some Business Process")
Business_Service(itSupportService, "IT Support for Business (Application Service)")

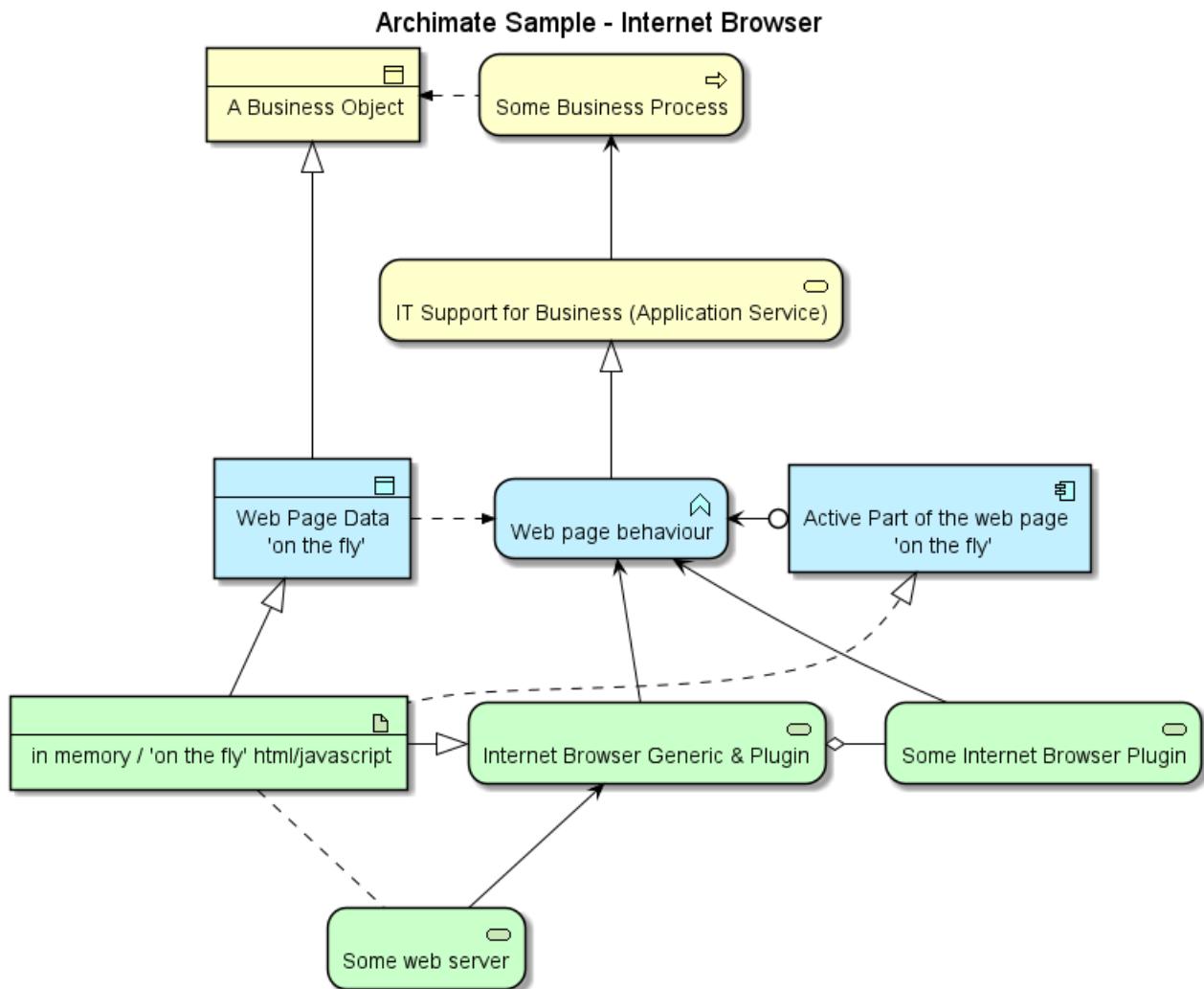
Application_DataObject(dataObject, "Web Page Data \n 'on the fly'")
Application_Function(webpageBehaviour, "Web page behaviour")
Application_Component(ActivePartWebPage, "Active Part of the web page \n 'on the fly')

Technology_Artifact(inMemoryItem,"in memory / 'on the fly' html/javascript")
Technology_Service(internetBrowser, "Internet Browser Generic & Plugin")
Technology_Service(internetBrowserPlugin, "Some Internet Browser Plugin")
Technology_Service(webServer, "Some web server")

'Relationships
Rel_Flow_Left(someBusinessProcess, businessObject, "")
Rel_Serving_Up(itSupportService, someBusinessProcess, "")
Rel_Specialization_Up(webpageBehaviour, itSupportService, "")
Rel_Flow_Right(dataObject, webpageBehaviour, "")
Rel_Specialization_Up(dataObject, businessObject, "")
Rel_Assignment_Left(ActivePartWebPage, webpageBehaviour, "")
Rel_Specialization_Up(inMemoryItem, dataObject, "")
Rel_Realization_Up(inMemoryItem, ActivePartWebPage, "")
Rel_Specialization_Right(inMemoryItem,internetBrowser, "")
Rel_Serving_Up(internetBrowser, webpageBehaviour, "")
Rel_Serving_Up(internetBrowserPlugin, webpageBehaviour, "")
Rel_Aggregation_Right(internetBrowser, internetBrowserPlugin, "")
Rel_Access_Up(webServer, inMemoryItem, "")
Rel_Serving_Up(webServer, internetBrowser, "")

@enduml
```





### 27.3 AWS library [aws]

- <https://github.com/plantuml/plantuml-stdlib/tree/master/aws>
- <https://github.com/milo-minderbinder/AWS-PlantUML>

The AWS library consists of Amazon AWS icons, it provides icons of two different sizes (normal and large).

Use it by including the file that contains the sprite, eg: !include <aws/Storage/AmazonS3/AmazonS3>. When imported, you can use the sprite as normally you would, using <\$sprite\_name>.

You may also include the common.puml file, eg: !include <aws/common>, which contains helper macros defined. With the common.puml imported, you can use the NAME\_OF\_SPRITE(parameters...) macro.

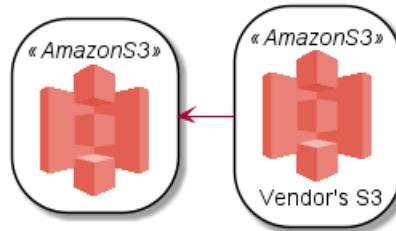
Example of usage:

```

@startuml
!include <aws/common>
!include <aws/Storage/AmazonS3/AmazonS3>

AMAZONS3(s3_internal)
AMAZONS3(s3_partner, "Vendor's S3")
s3_internal <- s3_partner
@enduml
  
```

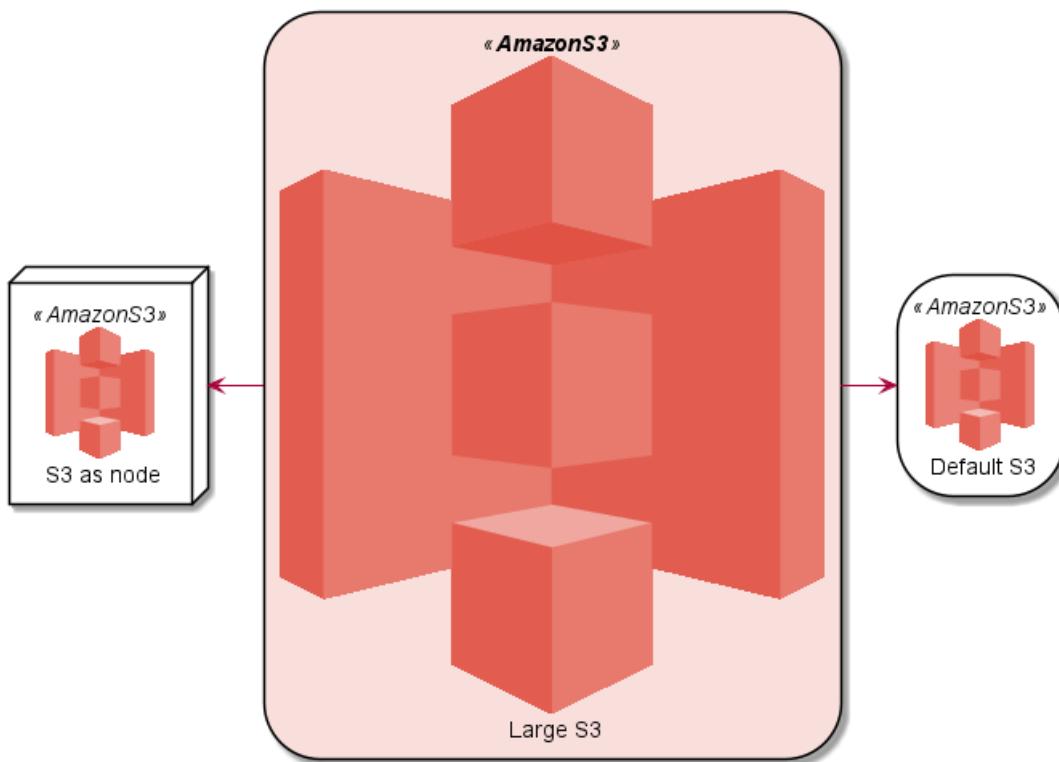




```
@startuml
!include <aws/common>
!include <aws/Storage/AmazonS3/AmazonS3>
!include <aws/Storage/AmazonS3/AmazonS3_LARGE>

skinparam nodeBackgroundColor White
skinparam storage<<**AmazonS3**>> {
    backgroundColor #F9DFDC
}
AMAZONS3(s3_internal,"Default S3")
AMAZONS3(s3_internal2,"S3 as node",node)
AMAZONS3_LARGE(s3_partner,"Large S3")

s3_internal2 <-r- s3_partner
s3_internal <-l- s3_partner
@enduml
```



## 27.4 Amazon Labs AWS Library [awslib]

- <https://github.com/plantuml/plantuml-stdlib/tree/master/awslib>
- <https://github.com/awslabs/aws-icons-for-plantuml>

The Amazon Labs AWS library provides PlantUML sprites, macros, and other includes for Amazon Web Services (AWS) services and resources.

Used to create PlantUML diagrams with AWS components. All elements are generated from the official



AWS Architecture Icons and when combined with PlantUML and the C4 model, are a great way to communicate your design, deployment, and topology as code.

```
@startuml
'Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
'SPDX-License-Identifier: MIT (For details, see https://github.com/awslabs/aws-icons-for-plantuml/bl

!include <awslib/AWSCommon>

' Uncomment the following line to create simplified view
' !include <awslib/AWSSimplified>

!include <awslib/General/Users>
!include <awslib/Mobile/APIGateway>
!include <awslib/SecurityIdentityAndCompliance/Cognito>
!include <awslib/Compute/Lambda>
!include <awslib/Database/DynamoDB>

left to right direction

Users(sources, "Events", "millions of users")
APIGateway(votingAPI, "Voting API", "user votes")
Cognito(userAuth, "User Authentication", "jwt to submit votes")
Lambda(generateToken, "User Credentials", "return jwt")
Lambda(recordVote, "Record Vote", "enter or update vote per user")
DynamoDB(voteDb, "Vote Database", "one entry per user")

sources --> userAuth
sources --> votingAPI
userAuth <--> generateToken
votingAPI --> recordVote
recordVote --> voteDb
@enduml
```

## 27.5 Azure library [azure]

- <https://github.com/plantuml/plantuml-stdlib/tree/master/azure>
- <https://github.com/RicardoNiepel/Azure-PlantUML/>

The Azure library consists of Microsoft Azure icons.

Use it by including the file that contains the sprite, eg: !include <azure/Analytics/AzureEventHub.puml>. When imported, you can use the sprite as normally you would, using <\$sprite\_name>.

You may also include the `AzureCommon.puml` file, eg: !include <azure/AzureCommon.puml>, which contains helper macros defined. With the `AzureCommon.puml` imported, you can use the `NAME_OF_SPRITE(parameters...)` macro.

Example of usage:

```
@startuml
!include <azure/AzureCommon.puml>
!include <azure/Analytics/AzureEventHub.puml>
!include <azure/Analytics/AzureStreamAnalytics.puml>
!include <azure/Database/AzureCosmosDb.puml>
```

left to right direction

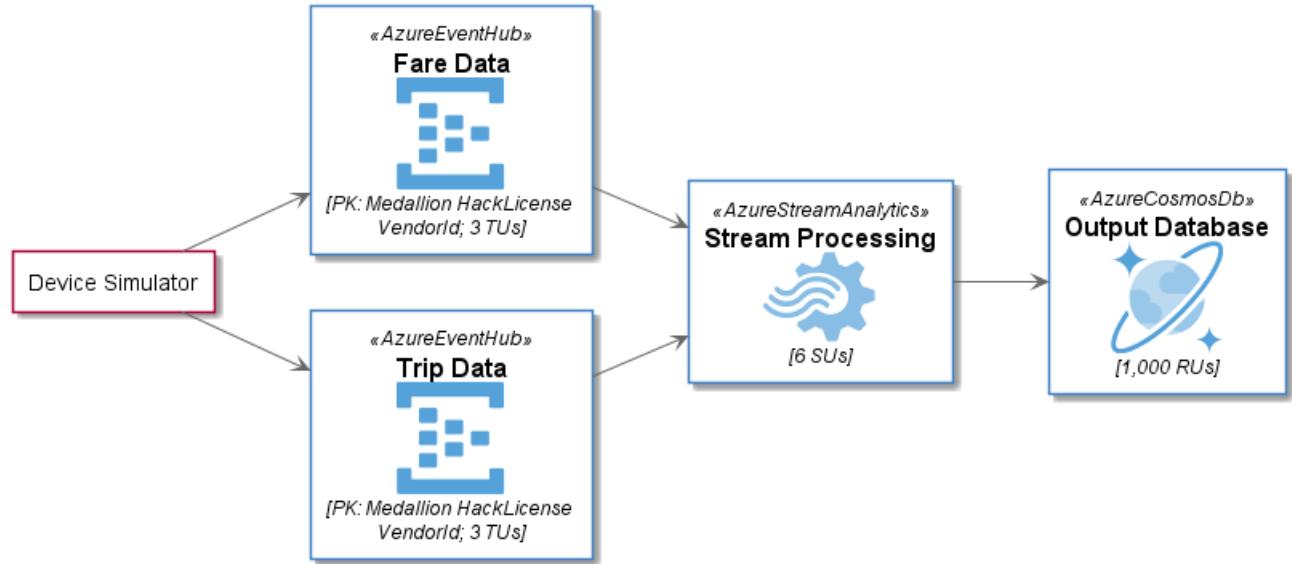
```
agent "Device Simulator" as devices #fff
```

```
AzureEventHub(fareDataEventHub, "Fare Data", "PK: Medallion HackLicense VendorId; 3 TUs")
```



```
AzureEventHub(tripDataEventHub, "Trip Data", "PK: Medallion HackLicense VendorId; 3 TUs")
AzureStreamAnalytics(streamAnalytics, "Stream Processing", "6 SUs")
AzureCosmosDb(outputCosmosDb, "Output Database", "1,000 RUs")
```

```
devices --> fareDataEventHub
devices --> tripDataEventHub
fareDataEventHub --> streamAnalytics
tripDataEventHub --> streamAnalytics
streamAnalytics --> outputCosmosDb
@enduml
```



## 27.6 C4 Library [C4]

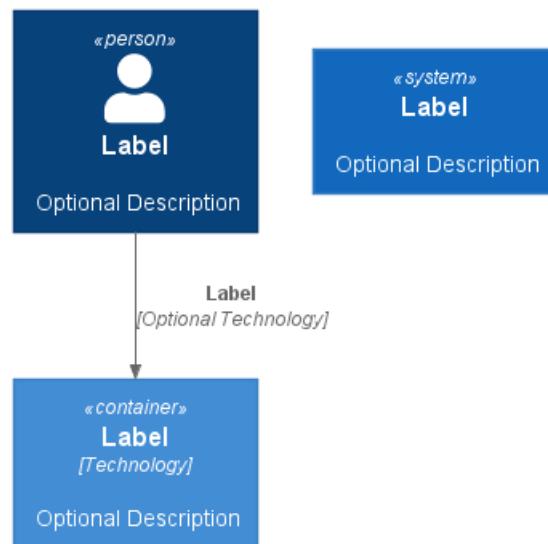
- <https://github.com/plantuml/plantuml-stdlib/tree/master/C4>
- <https://github.com/plantuml-stdlib/C4-PlantUML>

```
@startuml
!include <C4/C4_Container>
```

```
Person(personAlias, "Label", "Optional Description")
Container(containerAlias, "Label", "Technology", "Optional Description")
System(systemAlias, "Label", "Optional Description")
```

```
Rel(personAlias, containerAlias, "Label", "Optional Technology")
@enduml
```





## 27.7 Cloud Insight [cloudinsight]

- <https://github.com/plantuml/plantuml-stdlib/tree/master/cloudinsight>
- <https://github.com/rabelenda/cicon-plantuml-sprites>

This repository contains PlantUML sprites generated from Cloudinsight icons, which can easily be used in PlantUML diagrams for nice visual representation of popular technologies.

```

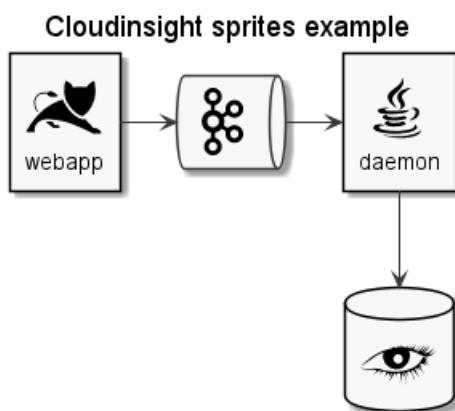
@startuml
!include <cloudinsight/tomcat>
!include <cloudinsight/kafka>
!include <cloudinsight/java>
!include <cloudinsight/cassandra>

title Cloudinsight sprites example

skinparam monochrome true

rectangle "<$tomcat>\nwebapp" as webapp
queue "<$kafka>" as kafka
rectangle "<$java>\ndaemon" as daemon
database "<$cassandra>" as cassandra

webapp -> kafka
kafka -> daemon
daemon --> cassandra
@enduml
  
```



## 27.8 Cloudogu [cloudogu]

- <https://github.com/plantuml/plantuml-stdlib/tree/master/cloudogu>
- <https://github.com/cloudogu/plantuml-cloudogu-sprites>
- <https://cloudogu.com>

The Cloudogu library provides PlantUML sprites, macros, and other includes for Cloudogu services and resources.

```
@startuml
!include <cloudogu/common.puml>
!include <cloudogu/dogus/jenkins.puml>
!include <cloudogu/dogus/cloudogu.puml>
!include <cloudogu/dogus/scm.puml>
!include <cloudogu/dogus/smeagol.puml>
!include <cloudogu/dogus/nexus.puml>
!include <cloudogu/tools/k8s.puml>

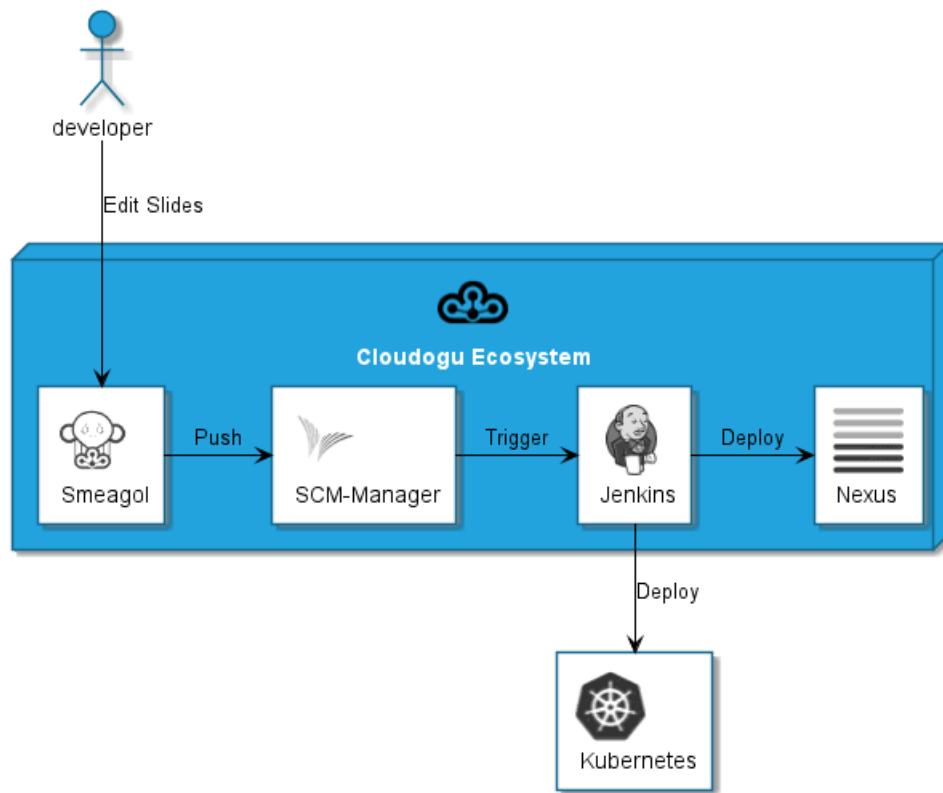
node "Cloudogu Ecosystem" <<$cloudogu>> {
    DOGU_JENKINS(jenkins, Jenkins) #ffffff
    DOGU_SCM(scm, SCM-Manager) #ffffff
    DOGU_SMEAGOL(smeagol, Smeagol) #ffffff
    DOGU_NEXUS(nexus, Nexus) #ffffff
}

TOOL_K8S(k8s, Kubernetes) #ffffff

actor developer

developer --> smeagol : "Edit Slides"
smeagol -> scm : Push
scm -> jenkins : Trigger
jenkins -> nexus : Deploy
jenkins --> k8s : Deploy
@enduml
```





### All cloudogu sprites

See all possible cloudogu sprites on [plantuml-cloudogu-sprites](#).

## 27.9 Elastic library [elastic]

- <https://github.com/plantuml/plantuml-stdlib/tree/master/elastic>
- <https://github.com/Crashedmind/PlantUML-Elastic-icons>

The Elastic library consists of Elastic icons. It is similar in use to the AWS and Azure libraries (it used the same tool to create them).

Use it by including the file that contains the sprite, eg: !include elastic/elasticsearch/elasticsearch.puml. When imported, you can use the sprite as normally you would, using <\$sprite\_name>.

You may also include the common.puml file, eg: !include <elastic/common>, which contains helper macros defined. With the common.puml imported, you can use the NAME//OF//SPRITE(parameters...) macro.

Example of usage:

```

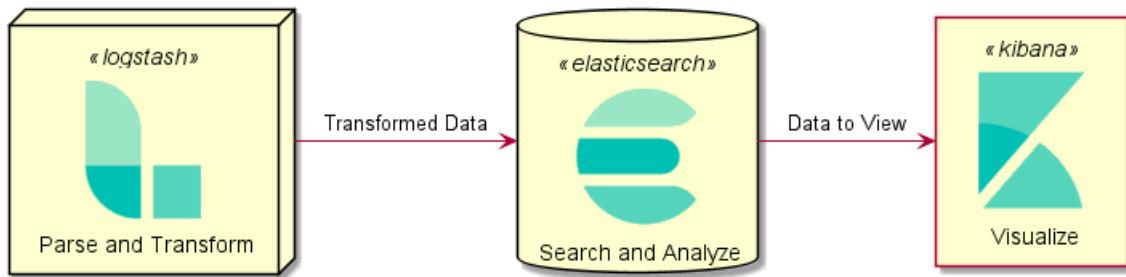
@startuml
!include <elastic/common>
!include <elastic/elasticsearch/elasticsearch>
!include <elastic/logstash/logstash>
!include <elastic/kibana/kibana>

ELASTICSEARCH(ElasticSearch, "Search and Analyze", database)
LOGSTASH(Logstash, "Parse and Transform", node)
KIBANA(Kibana, "Visualize", agent)

Logstash -right-> ElasticSearch: Transformed Data
ElasticSearch -right-> Kibana: Data to View
@enduml

```



**All Elastic Sprite Set**

@startuml

'Adapted from <https://github.com/Crashedmind/PlantUML-Elastic-icons/blob/master/All.puml>

'Elastic stuff here

```

'=====
!include <elastic/common.puml>
!include <elastic/apm/apm.puml>
!include <elastic/app_search/app_search.puml>
!include <elastic/beats/beats.puml>
!include <elastic/cloud/cloud.puml>
!include <elastic/cloud_in_kubernetes/cloud_in_kubernetes.puml>
!include <elastic/code_search/code_search.puml>
!include <elastic/ece/ece.puml>
!include <elastic/eck/eck.puml>
' Beware of the difference between Crashedmind and plantuml-stdlib version: with '_' usage!
!include <elastic/elasticsearch/elasticsearch.puml>
!include <elastic/endpoint/endpoint.puml>
!include <elastic/enterprise_search/enterprise_search.puml>
!include <elastic/kibana/kibana.puml>
!include <elastic/logging/logging.puml>
!include <elastic/logstash/logstash.puml>
!include <elastic/maps/maps.puml>
!include <elastic/metrics/metrics.puml>
!include <elastic/siem/siem.puml>
!include <elastic/site_search/site_search.puml>
!include <elastic/stack/stack.puml>
!include <elastic/uptime/uptime.puml>

```

skinparam agentBackgroundColor White

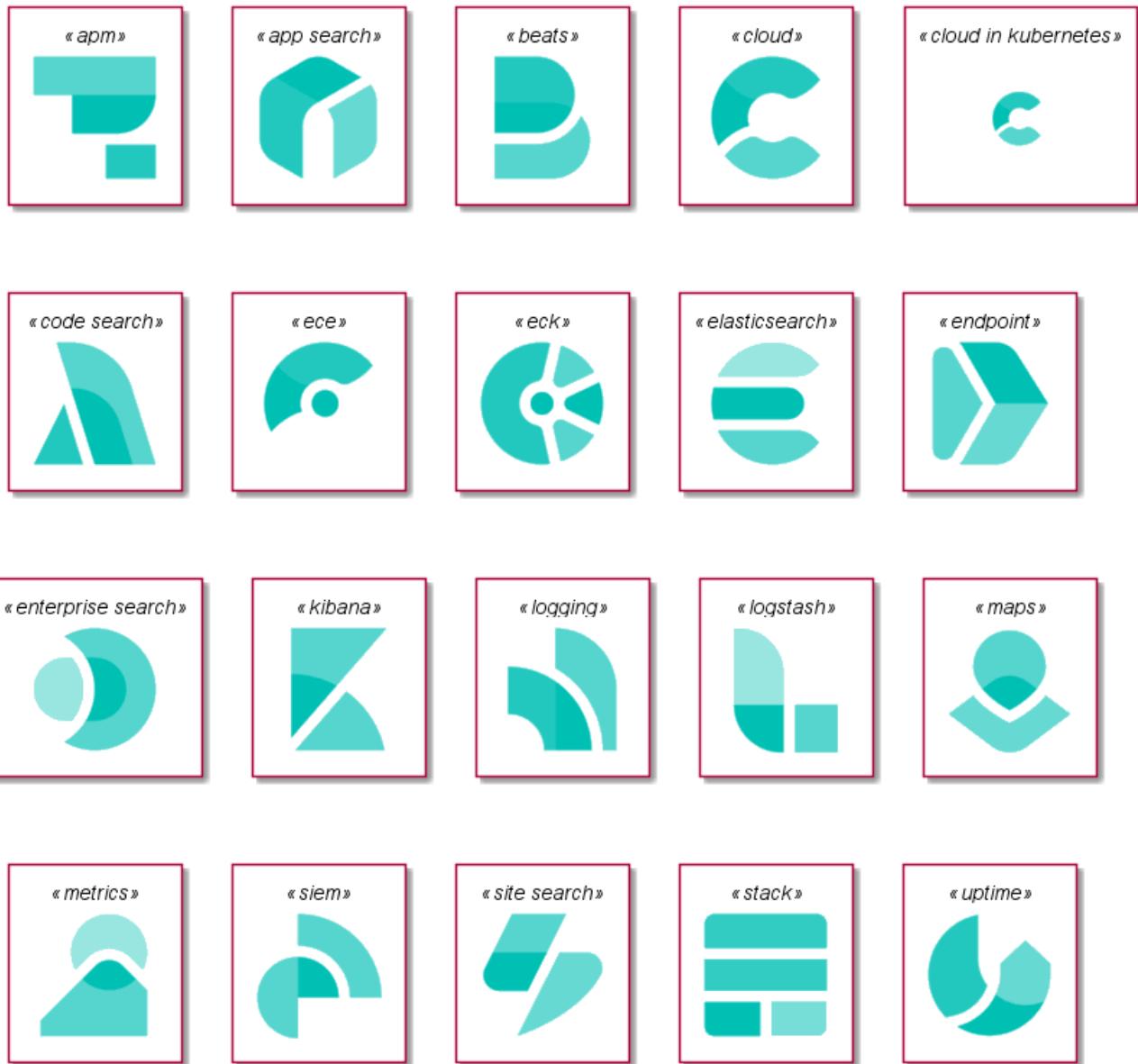
```

APM(apm)
APP_SEARCH(app_search)
BEATS(beats)
CLOUD(cloud)
CLOUD_IN_KUBERNETES(cloud_in_kubernetes)
CODE_SEARCH(code_search)
ECE(ece)
ECK(eck)
ELASTICSEARCH(elastic_search)
ENDPOINT(endpoint)
ENTERPRISE_SEARCH(enterprise_search)
KIBANA(kibana)
LOGGING(logging)
LOGSTASH(logstash)
MAPS(maps)
METRICS(metrics)

```



```
SIEM(siem)
SITE_SEARCH(site_search)
STACK(stack)
UPTIME(uptime)
@enduml
```



## 27.10 Google Material Icons [material]

- <https://github.com/plantuml/plantuml-stdlib/tree/master/material>
- <https://github.com/Templarian/MaterialDesign>

This library consists of a free Material style icons from Google and other artists.

Use it by including the file that contains the sprite, eg: `!include <material/ma_folder_move>`. When imported, you can use the sprite as normally you would, using `<$ma_sprite_name>`. Notice that this library requires an `ma_` prefix on sprites names, this is to avoid clash of names if multiple sprites have the same name on different libraries.

You may also include the `common.puml` file, eg: `!include <material/common>`, which contains helper macros defined. With the `common.puml` imported, you can use the `MA_NAME_OF_SPRITE(parameters...)`



macro, note again the use of the prefix MA\_.

Example of usage:

```
@startuml
!include <material/common>
' To import the sprite file you DON'T need to place a prefix!
!include <material/folder_move>

MA_FOLDER_MOVE(Red, 1, dir, rectangle, "A label")
@enduml
```



#### Notes:

When mixing sprites macros with other elements you may get a syntax error if, for example, trying to add a rectangle along with classes. In those cases, add { and } after the macro to create the empty rectangle.

Example of usage:

```
@startuml
!include <material/common>
' To import the sprite file you DON'T need to place a prefix!
!include <material/folder_move>

MA_FOLDER_MOVE(Red, 1, dir, rectangle, "A label") {
}

class foo {
    bar
}
@enduml
```

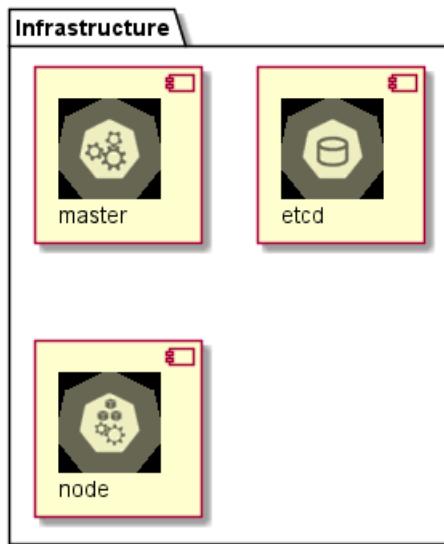


## 27.11 Kubernetes [kubernetes]

- <https://github.com/plantuml/plantuml-stdlib/tree/master/kubernetes>
- <https://github.com/michiel/plantuml-kubernetes-sprites>

```
@startuml
!include <kubernetes/k8s-sprites-unlabeled-25pct>
package "Infrastructure" {
    component "<$master>\nmaster" as master
    component "<$etcd>\netcd" as etcd
    component "<$node>\nnode" as node
}
@enduml
```





## 27.12 Logos [logos]

- <https://github.com/plantuml/plantuml-stdlib/tree/master/logos>
- <https://github.com/plantuml-stdlib/gilbarbara-plantuml-sprites>

This repository contains PlantUML sprites generated from Gil Barbara's logos, which can easily be used in PlantUML diagrams for nice visual aid.

```
@startuml
!include <logos/flask.puml>
!include <logos/kafka.puml>
!include <logos/kotlin.puml>
!include <logos/cassandra.puml>
```

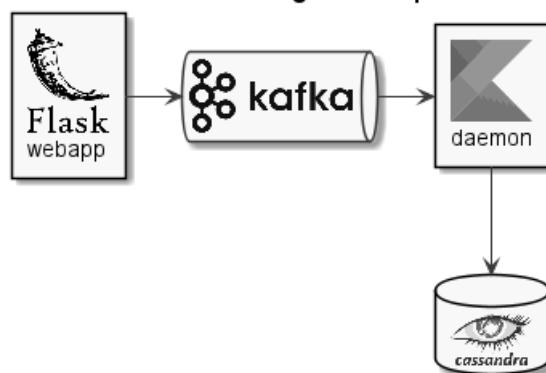
```
title Gil Barbara's logos example
```

```
skinparam monochrome true
```

```
rectangle "<$flask>\nwebapp" as webapp
queue "<$kafka>" as kafka
rectangle "<$kotlin>\ndaemon" as daemon
database "<$cassandra>" as cassandra
```

```
webapp -> kafka
kafka -> daemon
daemon --> cassandra
@enduml
```

Gil Barbara's logos example

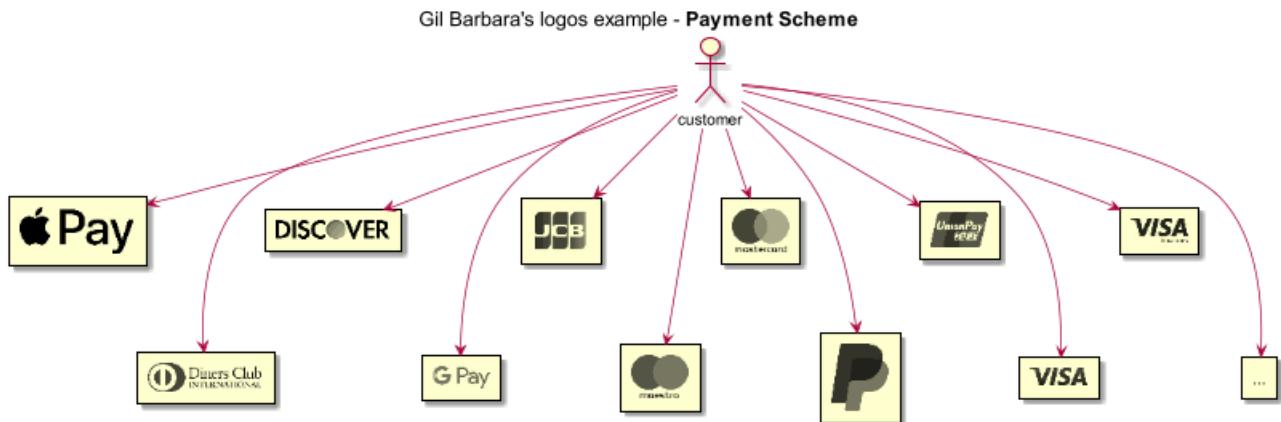


```
@startuml
scale 0.7
!include <logos/apple-pay.puml>
!include <logos/dinersclub.puml>
!include <logos/discover.puml>
!include <logos/google-pay.puml>
!include <logos/jcb.puml>
!include <logos/maestro.puml>
!include <logos/mastercard.puml>
!include <logos/paypal.puml>
!include <logos/unionpay.puml>
!include <logos/visaelectron.puml>
!include <logos/visa.puml>
' ...
title Gil Barbara's logos example - **Payment Scheme**

actor customer
rectangle "<$apple-pay>" as ap
rectangle "<$dinersclub>" as dc
rectangle "<$discover>" as d
rectangle "<$google-pay>" as gp
rectangle "<$jcb>" as j
rectangle "<$maestro>" as ma
rectangle "<$mastercard>" as m
rectangle "<$paypal>" as p
rectangle "<$unionpay>" as up
rectangle "<$visa>" as v
rectangle "<$visaelectron>" as ve
rectangle "..." as etc

customer --> ap
customer ---> dc
customer --> d
customer ---> gp
customer --> j
customer ---> ma
customer --> m
customer ---> p
customer --> up
customer ---> v
customer --> ve
customer ---> etc
@enduml
```





### 27.13 Office [office]

- <https://github.com/plantuml/plantuml-stdlib/tree/master/office>
- <https://github.com/Roemer/plantuml-office>

There are sprites (\*.puml) and colored png icons available. Be aware that the sprites are all only monochrome even if they have a color in their name (due to automatically generating the files). You can either color the sprites with the macro (see examples below) or directly use the fully colored pngs. See the following examples on how to use the sprites, the pngs and the macros.

Example of usage:

```

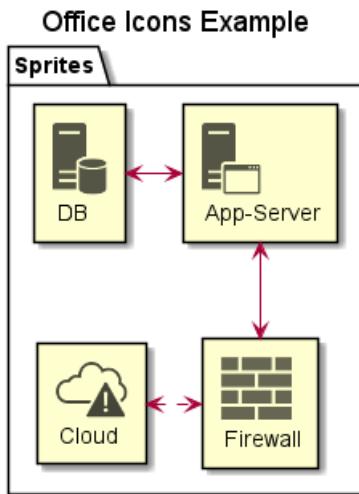
@startuml
!include <tupadr3/common>

!include <office/Servers/database_server>
!include <office/Servers/application_server>
!include <office/Concepts/firewall_orange>
!include <office/Clouds/cloud_disaster_red>

title Office Icons Example

package "Sprites" {
    OFF_DATABASE_SERVER(db,DB)
    OFF_APPLICATION_SERVER(app,App-Server)
    OFF_FIREWALL_ORANGE(fw,Firewall)
    OFF_CLOUD_DISASTER_RED(cloud,Cloud)
    db <-> app
    app <--> fw
    fw <.left.> cloud
}
@enduml
  
```





```

@startuml
!include <tupadr3/common>

!include <office/servers/database_server>
!include <office/servers/application_server>
!include <office/Concepts/firewall_orange>
!include <office/Clouds/cloud_disaster_red>

' Used to center the label under the images
skinparam defaultTextAlignment center

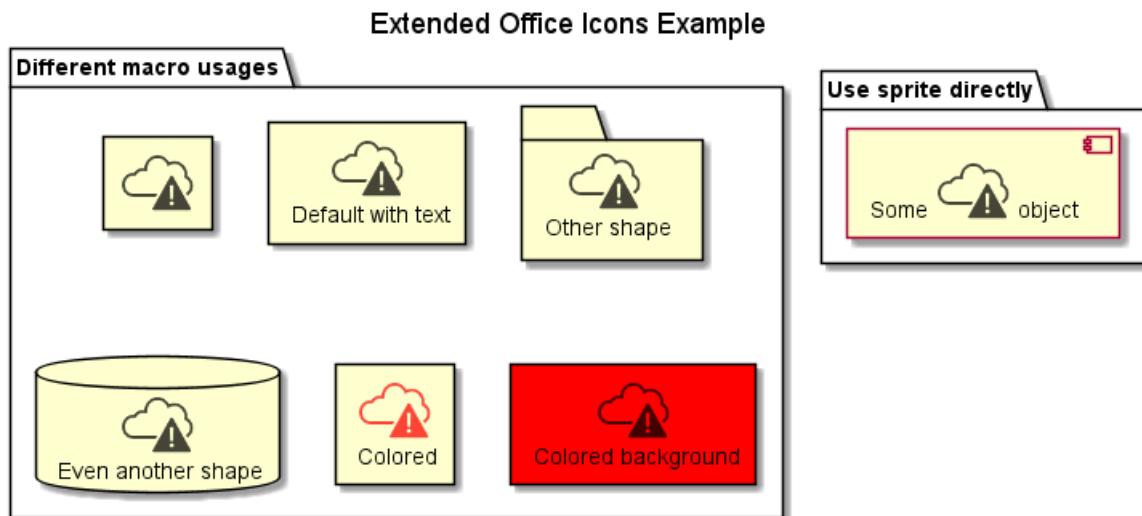
title Extended Office Icons Example

package "Use sprite directly" {
    [Some <$cloud_disaster_red> object]
}

package "Different macro usages" {
    OFF_CLOUD_DISASTER_RED(cloud1)
    OFF_CLOUD_DISASTER_RED(cloud2,Default with text)
    OFF_CLOUD_DISASTER_RED(cloud3,Other shape,Folder)
    OFF_CLOUD_DISASTER_RED(cloud4,Even another shape,Database)
    OFF_CLOUD_DISASTER_RED(cloud5,Colored,Rectangle, red)
    OFF_CLOUD_DISASTER_RED(cloud6,Colored background) #red
}
@enduml

```





## 27.14 Open Security Architecture (OSA) [osa]

- <https://github.com/plantuml/plantuml-stdlib/tree/master/osa>
- <https://github.com/Crashedmind/PlantUML-opensecurityarchitecture-icons>
- <https://www.opensecurityarchitecture.org>

```
@startuml
'Adapted from https://github.com/Crashedmind/PlantUML-opensecurityarchitecture-icons/blob/master/all
scale .5
!include <osa/arrow/green/left/left.puml>
!include <osa/arrow/yellow/right/right.puml>
!include <osa/awareness/awareness.puml>
!include <osa/contract/contract.puml>
!include <osa/database/database.puml>
!include <osa/desktop/desktop.puml>
!include <osa/desktop/imac/imac.puml>
!include <osa/device_music/device_music.puml>
!include <osa/device_scanner/device_scanner.puml>
!include <osa/device_usb/device_usb.puml>
!include <osa/device_wireless_router/device_wireless_router.puml>
!include <osa/disposal/disposal.puml>
!include <osa/drive_optical/drive_optical.puml>
!include <osa/firewall/firewall.puml>
!include <osa/hub/hub.puml>
!include <osa/ics/drive/drive.puml>
!include <osa/ics/plc/plc.puml>
!include <osa/ics/thermometer/thermometer.puml>
!include <osa/id/card/card.puml>
!include <osa/laptop/laptop.puml>
!include <osa/lifecycle/lifecycle.puml>
!include <osa/lightning/lightning.puml>
!include <osa/media_flash/media_flash.puml>
!include <osa/media_optical/media_optical.puml>
!include <osa/media_tape/media_tape.puml>
!include <osa/mobile/pda/pda.puml>
!include <osa/padlock/padlock.puml>
!include <osa/printer/printer.puml>
!include <osa/site_branch/site_branch.puml>
!include <osa/site_factory/site_factory.puml>
!include <osa/vpn/vpn.puml>
```



```
!include <osa/wireless/network/network.puml>

rectangle "OSA" {
rectangle "Left: <$left>" 
rectangle "Right: <$right>" 
rectangle "Awareness: <$awareness>" 
rectangle "Contract: <$contract>" 
rectangle "Database: <$database>" 
rectangle "Desktop: <$desktop>" 
rectangle "Imac: <$imac>" 
rectangle "Device_music: <$device_music>" 
rectangle "Device_scanner: <$device_scanner>" 
rectangle "Device_usb: <$device_usb>" 
rectangle "Device_wireless_router: <$device_wireless_router>" 
rectangle "Disposal: <$disposal>" 
rectangle "Drive_optical: <$drive_optical>" 
rectangle "Firewall: <$firewall>" 
rectangle "Hub: <$hub>" 
rectangle "Drive: <$drive>" 
rectangle "Plc: <$plc>" 
rectangle "Thermometer: <$thermometer>" 
rectangle "Card: <$card>" 
rectangle "Laptop: <$laptop>" 
rectangle "Lifecycle: <$lifecycle>" 
rectangle "Lightning: <$lightning>" 
rectangle "Media_flash: <$media_flash>" 
rectangle "Media_optical: <$media_optical>" 
rectangle "Media_tape: <$media_tape>" 
rectangle "Pda: <$pda>" 
rectangle "Padlock: <$padlock>" 
rectangle "Printer: <$printer>" 
rectangle "Site_branch: <$site_branch>" 
rectangle "Site_factory: <$site_factory>" 
rectangle "Vpn: <$vpn>" 
rectangle "Network: <$network>" 
}
@enduml
```





## 27.15 Tupadr3 library [tupadr3]

- <https://github.com/plantuml/plantuml-stdlib/tree/master/tupadr3>
- <https://github.com/tupadr3/plantuml-icon-font-sprites>

This library contains several libraries of icons (including Devicons and Font Awesome).

Use it by including the file that contains the sprite, eg: `!include <font-awesome/common>`. When imported, you can use the sprite as normally you would, using `<$sprite_name>`.

You may also include the `common.puml` file, eg: `!include <font-awesome/common>`, which contains helper macros defined. With the `common.puml` imported, you can use the `NAME_OF_SPRITE(parameters...)` macro.

Example of usage:

```
@startuml
!include <tupadr3/common>
!include <tupadr3/font-awesome/server>
!include <tupadr3/font-awesome/database>
```

`title Styling example`

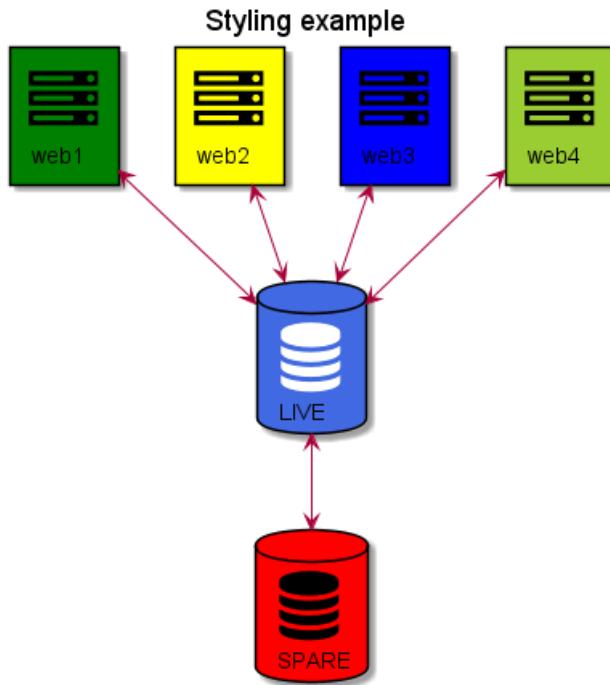
```
FA_SERVER(web1,web1) #Green
FA_SERVER(web2,web2) #Yellow
FA_SERVER(web3,web3) #Blue
FA_SERVER(web4,web4) #YellowGreen
```



```
FA_DATABASE(db1,LIVE,database,white) #RoyalBlue
FA_DATABASE(db2,SPARE,database) #Red
```

```
db1 <--> db2
```

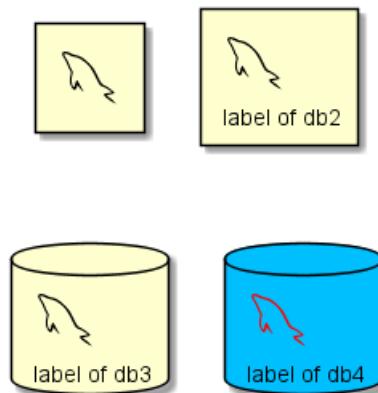
```
web1 <--> db1
web2 <--> db1
web3 <--> db1
web4 <--> db1
@enduml
```



```
@startuml
```

```
!include <tupadr3/common>
!include <tupadr3/devicons/mysql>
```

```
DEV_MYSQL(db1)
DEV_MYSQL(db2,label of db2)
DEV_MYSQL(db3,label of db3,database)
DEV_MYSQL(db4,label of db4,database,red) #DeepSkyBlue
@enduml
```



## Contents

<b>1 Sequence Diagram</b>	<b>1</b>
1.1 Basic examples . . . . .	1
1.2 Declaring participant . . . . .	1
1.3 Use non-letters in participants . . . . .	3
1.4 Message to Self . . . . .	3
1.5 Text alignment . . . . .	4
1.5.1 Text of response message below the arrow . . . . .	4
1.6 Change arrow style . . . . .	4
1.7 Change arrow color . . . . .	5
1.8 Message sequence numbering . . . . .	5
1.9 Page Title, Header and Footer . . . . .	7
1.10 Splitting diagrams . . . . .	8
1.11 Grouping message . . . . .	9
1.12 Secondary group label . . . . .	10
1.13 Notes on messages . . . . .	11
1.14 Some other notes . . . . .	11
1.15 Changing notes shape [hnote, rnote] . . . . .	12
1.16 Note over all participants [across] . . . . .	13
1.17 Several notes aligned at the same level [/] . . . . .	13
1.18 Creole and HTML . . . . .	14
1.19 Divider or separator . . . . .	15
1.20 Reference . . . . .	16
1.21 Delay . . . . .	16
1.22 Text wrapping . . . . .	17
1.23 Space . . . . .	17
1.24 Lifeline Activation and Destruction . . . . .	18
1.25 Return . . . . .	20
1.26 Participant creation . . . . .	20
1.27 Shortcut syntax for activation, deactivation, creation . . . . .	21
1.28 Incoming and outgoing messages . . . . .	23
1.29 Short arrows for incoming and outgoing messages . . . . .	24
1.30 Anchors and Duration . . . . .	25
1.31 Stereotypes and Spots . . . . .	25
1.32 More information on titles . . . . .	26
1.33 Participants encompass . . . . .	28
1.34 Removing Foot Boxes . . . . .	28
1.35 Skinparam . . . . .	29
1.36 Changing padding . . . . .	31
1.37 Appendix: Examples of all arrow type . . . . .	32
1.37.1 Normal arrow . . . . .	32
1.37.2 Incoming and outgoing messages (with '[, ']') . . . . .	33
1.37.3 Incoming messages (with '[') . . . . .	33
1.37.4 Outgoing messages (with ']') . . . . .	34
1.37.5 Short incoming and outgoing messages (with '?') . . . . .	35
1.37.6 Short incoming (with '?') . . . . .	35
1.37.7 Short outgoing (with '?') . . . . .	37
1.38 Specific SkinParameter . . . . .	38
1.38.1 By default . . . . .	38
1.38.2 LifelineStrategy . . . . .	39
1.38.3 style strictuml . . . . .	39
1.39 Hide unlinked participant . . . . .	40
<b>2 Use Case Diagram</b>	<b>41</b>
2.1 Usecases . . . . .	41
2.2 Actors . . . . .	41
2.3 Change Actor style . . . . .	42
2.3.1 Stick man ( <i>by default</i> ) . . . . .	42



2.3.2	Awesome man	42
2.3.3	Hollow man	42
2.4	Usecases description	43
2.5	Use package	43
2.6	Basic example	45
2.7	Extension	45
2.8	Using notes	46
2.9	Stereotypes	47
2.10	Changing arrows direction	47
2.11	Splitting diagrams	49
2.12	Left to right direction	49
2.13	Skipparam	50
2.14	Complete example	51
2.15	Business Use Case	51
2.15.1	Business Usecase	51
2.15.2	Business Actor	52
2.16	Change arrow color and style (inline style)	52
2.17	Change element color and style (inline style)	53
<b>3</b>	<b>Class Diagram</b>	<b>54</b>
3.1	Declaring element	54
3.2	Relations between classes	54
3.3	Label on relations	55
3.4	Adding methods	56
3.5	Defining visibility	57
3.6	Abstract and Static	58
3.7	Advanced class body	58
3.8	Notes and stereotypes	59
3.9	More on notes	60
3.10	Note on field (field, attribute, member) or method	61
3.10.1	Note on field or method	61
3.10.2	Note on method with the same name	61
3.11	Note on links	62
3.12	Abstract class and interface	62
3.13	Using non-letters	63
3.14	Hide attributes, methods...	64
3.15	Hide classes	65
3.16	Remove classes	65
3.17	Hide or Remove unlinked class	66
3.18	Use generics	67
3.19	Specific Spot	67
3.20	Packages	67
3.21	Packages style	68
3.22	Namespaces	69
3.23	Automatic namespace creation	70
3.24	Lollipop interface	71
3.25	Changing arrows direction	71
3.26	Association classes	72
3.27	Association on same classe	73
3.28	Skipparam	74
3.29	Skinned Stereotypes	74
3.30	Color gradient	75
3.31	Help on layout	76
3.32	Splitting large files	76
3.33	Extends and implements	77
3.34	Bracketed relations (linking or arrow) style	77
3.34.1	Line style	77
3.34.2	Line color	78



3.34.3 Line thickness . . . . .	79
3.34.4 Mix . . . . .	80
3.35 Change relation (linking or arrow) color and style (inline style) . . . . .	80
3.36 Change class color and style (inline style) . . . . .	81
3.37 Arrows from/to class members . . . . .	81
<b>4 Object Diagram</b>	<b>83</b>
4.1 Definition of objects . . . . .	83
4.2 Relations between objects . . . . .	83
4.3 Associations objects . . . . .	83
4.4 Adding fields . . . . .	84
4.5 Common features with class diagrams . . . . .	84
4.6 Map table or associative array . . . . .	85
<b>5 Activity Diagram (legacy)</b>	<b>87</b>
5.1 Simple Action . . . . .	87
5.2 Label on arrows . . . . .	87
5.3 Changing arrow direction . . . . .	87
5.4 Branches . . . . .	88
5.5 More on Branches . . . . .	89
5.6 Synchronization . . . . .	90
5.7 Long action description . . . . .	91
5.8 Notes . . . . .	91
5.9 Partition . . . . .	92
5.10 Skinparam . . . . .	93
5.11 Octagon . . . . .	94
5.12 Complete example . . . . .	94
<b>6 Activity Diagram (new)</b>	<b>97</b>
6.1 Simple action . . . . .	97
6.2 Start/Stop/End . . . . .	97
6.3 Conditional . . . . .	98
6.3.1 Several tests (horizontal mode) . . . . .	99
6.3.2 Several tests (vertical mode) . . . . .	100
6.4 Conditional with stop on an action [kill, detach] . . . . .	100
6.5 Repeat loop . . . . .	102
6.6 Break on a repeat loop [break] . . . . .	103
6.7 While loop . . . . .	104
6.8 Parallel processing . . . . .	105
6.9 Split processing . . . . .	106
6.9.1 Split . . . . .	106
6.9.2 Input split (multi-start) . . . . .	106
6.9.3 Output split (multi-end) . . . . .	107
6.10 Notes . . . . .	108
6.11 Colors . . . . .	109
6.12 Lines without arrows . . . . .	110
6.13 Arrows . . . . .	111
6.14 Connector . . . . .	111
6.15 Color on connector . . . . .	112
6.16 Grouping or partition . . . . .	112
6.17 Swimlanes . . . . .	113
6.18 Detach or kill [detach, kill] . . . . .	115
6.19 SDL (Specification and Description Language) . . . . .	117
6.20 Complete example . . . . .	118
6.21 Condition Style . . . . .	120
6.21.1 Inside style (by default) . . . . .	120
6.21.2 Diamond style . . . . .	121
6.21.3 InsideDiamond (or <i>Foo1</i> ) style . . . . .	122
6.22 Condition End Style . . . . .	123



6.22.1 Diamond style (by default) . . . . .	123
6.22.2 Horizontal line (hline) style . . . . .	124
<b>7 Component Diagram . . . . .</b>	<b>126</b>
7.1 Components . . . . .	126
7.2 Interfaces . . . . .	126
7.3 Basic example . . . . .	127
7.4 Using notes . . . . .	127
7.5 Grouping Components . . . . .	127
7.6 Changing arrows direction . . . . .	129
7.7 Use UML2 notation . . . . .	131
7.8 Use UML1 notation . . . . .	131
7.9 Use rectangle notation (remove UML notation) . . . . .	131
7.10 Long description . . . . .	132
7.11 Individual colors . . . . .	132
7.12 Using Sprite in Stereotype . . . . .	132
7.13 Skinparam . . . . .	133
7.14 Specific SkinParameter . . . . .	134
7.14.1 componentStyle . . . . .	134
7.15 Hide or Remove unlinked component . . . . .	136
<b>8 Deployment Diagram . . . . .</b>	<b>138</b>
8.1 Declaring element . . . . .	138
8.2 Declaring element (using short form) . . . . .	140
8.2.1 Actor . . . . .	140
8.2.2 Component . . . . .	141
8.2.3 Interface . . . . .	141
8.2.4 Usecase . . . . .	141
8.3 Linking or arrow . . . . .	141
8.4 Bracketed arrow style . . . . .	144
8.4.1 Line style . . . . .	144
8.4.2 Line color . . . . .	145
8.4.3 Line thickness . . . . .	145
8.4.4 Mix . . . . .	146
8.5 Change arrow color and style (inline style) . . . . .	146
8.6 Change element color and style (inline style) . . . . .	147
8.7 Nestable elements . . . . .	148
8.8 Packages and nested elements . . . . .	148
8.8.1 Example with one level . . . . .	148
8.8.2 Other example . . . . .	149
8.8.3 Full nesting . . . . .	150
8.9 Alias . . . . .	154
8.9.1 Simple alias with as . . . . .	154
8.9.2 Examples of long alias . . . . .	155
8.10 Round corner . . . . .	157
8.11 Specific SkinParameter . . . . .	157
8.11.1 roundCorner . . . . .	157
8.12 Appendix: All type of arrow line . . . . .	158
8.13 Appendix: All type of arrow head or '0' arrow . . . . .	159
8.13.1 Type of arrow head . . . . .	159
8.13.2 Type of '0' arrow or circle arrow . . . . .	160
8.14 Appendix: Test of inline style on all element . . . . .	161
8.14.1 Simple element . . . . .	161
8.14.2 Nested element . . . . .	162
8.14.3 Without sub-element . . . . .	162
8.14.4 With sub-element . . . . .	163
8.15 Appendix: Test of style on all element . . . . .	164
8.15.1 Simple element . . . . .	164
8.15.2 Global style (on componentDiagram) . . . . .	164



8.15.3 Style for each element . . . . .	165
8.15.4 Nested element (without level) . . . . .	168
8.15.5 Global style (on componentDiagram) . . . . .	168
8.15.6 Style for each nested element . . . . .	169
8.15.7 Nested element (with one level) . . . . .	171
8.15.8 Global style (on componentDiagram) . . . . .	171
8.15.9 Style for each nested element . . . . .	172
<b>9 State Diagram</b>	<b>175</b>
9.1 Simple State . . . . .	175
9.2 Change state rendering . . . . .	175
9.3 Composite state . . . . .	176
9.3.1 Internal sub-state . . . . .	176
9.3.2 Sub-state to sub-state . . . . .	177
9.4 Long name . . . . .	178
9.5 History [[H], [H*]] . . . . .	179
9.6 Fork [fork, join] . . . . .	180
9.7 Concurrent state [-,   ] . . . . .	181
9.7.1 Horizontal separator -- . . . . .	181
9.7.2 Vertical separator    . . . . .	182
9.8 Conditional [choice] . . . . .	183
9.9 Stereotypes full example [choice, fork, join, end] . . . . .	183
9.10 Point [entryPoint, exitPoint] . . . . .	184
9.11 Pin [inputPin, outputPin] . . . . .	185
9.12 Expansion [expansionInput, expansionOutput] . . . . .	186
9.13 Arrow direction . . . . .	187
9.14 Change line color and style . . . . .	188
9.15 Note . . . . .	188
9.16 Note on link . . . . .	189
9.17 More in notes . . . . .	189
9.18 Inline color . . . . .	190
9.19 Skinparam . . . . .	191
9.20 Changing style . . . . .	192
9.21 Change state color and style (inline style) . . . . .	193
<b>10 Timing Diagram</b>	<b>195</b>
10.1 Declaring participant . . . . .	195
10.2 Binary and Clock . . . . .	195
10.3 Adding message . . . . .	196
10.4 Relative time . . . . .	196
10.5 Anchor Points . . . . .	197
10.6 Participant oriented . . . . .	198
10.7 Setting scale . . . . .	198
10.8 Initial state . . . . .	198
10.9 Intricated state . . . . .	199
10.10 Hidden state . . . . .	200
10.11 Hide time axis . . . . .	200
10.12 Using Time and Date . . . . .	200
10.13 Adding constraint . . . . .	201
10.14 Highlighted period . . . . .	202
10.15 Adding texts . . . . .	203
10.16 Complete example . . . . .	203
10.17 Digital Example . . . . .	204
10.18 Adding color . . . . .	206
<b>11 Display JSON Data</b>	<b>207</b>
11.1 Complex example . . . . .	207
11.2 Highlight parts . . . . .	208
11.3 JSON basic element . . . . .	208



11.3.1	Synthesis of all JSON basic element . . . . .	208
11.4	JSON array or table . . . . .	209
11.4.1	Array type . . . . .	209
11.4.2	Minimal array or table . . . . .	210
11.4.3	Number array . . . . .	210
11.4.4	String array . . . . .	210
11.4.5	Boolean array . . . . .	210
11.5	JSON numbers . . . . .	210
11.6	JSON strings . . . . .	211
11.6.1	JSON Unicode . . . . .	211
11.6.2	JSON two-character escape sequence . . . . .	211
11.7	Minimal JSON examples . . . . .	212
11.8	Using (global) style . . . . .	213
11.8.1	Without style ( <i>by default</i> ) . . . . .	213
11.8.2	With style . . . . .	213
<b>12</b>	<b>Display YAML Data</b>	<b>215</b>
12.1	Complex example . . . . .	215
12.2	Specific key (with symbols or unicode) . . . . .	216
12.3	Highlight parts . . . . .	216
12.3.1	Normal style . . . . .	216
12.3.2	Customised style . . . . .	217
12.4	Using (global) style . . . . .	217
12.4.1	Without style ( <i>by default</i> ) . . . . .	217
12.4.2	With style . . . . .	218
<b>13</b>	<b>Network diagram (nwdiag)</b>	<b>220</b>
13.1	Simple diagram . . . . .	220
13.2	Define multiple addresses . . . . .	220
13.3	Grouping nodes . . . . .	221
13.3.1	Define group inside network definitions . . . . .	221
13.3.2	Define group outside of network definitions . . . . .	222
13.3.3	Define several groups on same network . . . . .	222
13.3.4	Example with 2 group . . . . .	222
13.3.5	Example with 3 groups . . . . .	223
13.4	Extended Syntax (for network or group) . . . . .	224
13.4.1	Network . . . . .	224
13.4.2	Group . . . . .	225
13.5	Using Sprites . . . . .	226
13.6	Using OpenIconic . . . . .	227
13.7	Same nodes on more than two networks . . . . .	228
13.8	Peer networks . . . . .	229
13.9	Peer networks and group . . . . .	229
13.9.1	Without group . . . . .	229
13.9.2	Group on first . . . . .	230
13.9.3	Group on second . . . . .	231
13.9.4	Group on third . . . . .	232
13.10	Add title, caption, header, footer or legend on network diagram . . . . .	233
13.11	Change width of the networks . . . . .	234
13.12	Other internal networks . . . . .	236
<b>14</b>	<b>Salt (Wireframe)</b>	<b>239</b>
14.1	Basic widgets . . . . .	239
14.2	Using grid [ ] . . . . .	239
14.3	Group box [ ] . . . . .	240
14.4	Using separator [., ==, ~~, -] . . . . .	240
14.5	Tree widget [T] . . . . .	241
14.6	Tree table [T] . . . . .	241
14.7	Enclosing brackets [{, }] . . . . .	243



14.8 Adding tabs [/] . . . . .	243
14.9 Using menu [*] . . . . .	244
14.10 Advanced table . . . . .	245
14.11 Scroll Bars [S, SI, S-] . . . . .	245
14.12 Colors . . . . .	246
14.13 Pseudo sprite [«, »] . . . . .	247
14.14 OpenIconic . . . . .	247
14.15 Include Salt "on activity diagram" . . . . .	248
14.16 Include salt "on while condition of activity diagram" . . . . .	250
<b>15 Archimate Diagram</b>	<b>252</b>
15.1 Archimate keyword . . . . .	252
15.2 Defining Junctions . . . . .	252
15.3 Example 1 . . . . .	253
15.4 Example 2 . . . . .	254
15.5 List possible sprites . . . . .	255
15.6 ArchiMate Macros . . . . .	255
15.6.1 Archimate Macros and Library . . . . .	255
15.6.2 Archimate elements . . . . .	255
15.6.3 Archimate relationships . . . . .	256
15.6.4 Appendix: Examples of all Archimate RelationTypes . . . . .	257
<b>16 Gantt Diagram</b>	<b>259</b>
16.1 Declaring tasks . . . . .	259
16.1.1 Duration . . . . .	259
16.1.2 Start . . . . .	259
16.1.3 End . . . . .	259
16.1.4 Start/End . . . . .	260
16.2 One-line declaration (with the and conjunction) . . . . .	260
16.3 Adding constraints . . . . .	260
16.4 Short names . . . . .	261
16.5 Customize colors . . . . .	261
16.6 Completion status . . . . .	261
16.7 Milestone . . . . .	261
16.7.1 Relative milestone (use of constraints) . . . . .	262
16.7.2 Absolute milestone (use of fixed date) . . . . .	262
16.7.3 Milestone of maximum end of tasks . . . . .	262
16.8 Hyperlinks . . . . .	262
16.9 Calendar . . . . .	263
16.10 Coloring days . . . . .	263
16.11 Changing scale . . . . .	263
16.11.1 Daily ( <i>by default</i> ) . . . . .	264
16.11.2 Weekly . . . . .	264
16.11.3 Monthly . . . . .	265
16.12 Close day . . . . .	265
16.13 Simplified task succession . . . . .	266
16.14 Separator . . . . .	266
16.15 Working with resources . . . . .	266
16.16 Complex example . . . . .	267
16.17 Comments . . . . .	268
16.18 Using style . . . . .	268
16.18.1 Without style ( <i>by default</i> ) . . . . .	268
16.18.2 With style . . . . .	268
16.19 Add notes . . . . .	270
16.20 Pause tasks . . . . .	272
16.21 Change link colors . . . . .	272
16.22 Tasks or Milestones on the same line . . . . .	273
16.23 Highlight today . . . . .	273
16.24 Task between two milestones . . . . .	273



16.25 Grammar and verbal form . . . . .	274
16.26 Add title, header, footer, caption or legend on gantt diagram . . . . .	274
16.27 Removing Foot Boxes . . . . .	274
<b>17 MindMap</b>	<b>277</b>
17.1 OrgMode syntax . . . . .	277
17.2 Markdown syntax . . . . .	277
17.3 Arithmetic notation . . . . .	278
17.4 Multilines . . . . .	278
17.5 Colors . . . . .	279
17.5.1 With inline color . . . . .	279
17.5.2 With style color . . . . .	280
17.6 Removing box . . . . .	282
17.7 Changing diagram direction . . . . .	283
17.8 Complete example . . . . .	283
17.9 Changing style . . . . .	284
17.9.1 node, depth . . . . .	284
17.9.2 boxless . . . . .	285
17.10 Word Wrap . . . . .	286
<b>18 Work Breakdown Structure (WBS)</b>	<b>288</b>
18.1 OrgMode syntax . . . . .	288
18.2 Change direction . . . . .	288
18.3 Arithmetic notation . . . . .	289
18.4 Removing box . . . . .	289
18.4.1 Boxless on Arithmetic notation . . . . .	290
18.4.2 Several boxless node . . . . .	290
18.4.3 All boxless node . . . . .	290
18.4.4 Boxless on OrgMode syntax . . . . .	291
18.4.5 Several boxless node . . . . .	291
18.4.6 All boxless node . . . . .	291
18.5 Colors (with inline or style color) . . . . .	292
18.6 Using style . . . . .	293
18.7 Word Wrap . . . . .	294
<b>19 Maths</b>	<b>297</b>
19.1 Standalone diagram . . . . .	297
19.2 How is this working? . . . . .	298
<b>20 Entity Relationship Diagram</b>	<b>299</b>
20.1 Information Engineering Relations . . . . .	299
20.2 Entities . . . . .	299
20.3 Complete Example . . . . .	300
<b>21 Common commands</b>	<b>302</b>
21.1 Comments . . . . .	302
21.2 Zoom . . . . .	302
21.3 Title . . . . .	302
21.4 Caption . . . . .	303
21.5 Footer and header . . . . .	304
21.6 Legend the diagram . . . . .	304
21.7 Appendix: Examples on all diagram . . . . .	305
21.7.1 Activity . . . . .	305
21.7.2 Archimate . . . . .	305
21.7.3 Class . . . . .	306
21.7.4 Component, Deployment, Use-Case . . . . .	307
21.7.5 Gantt project planning . . . . .	307
21.7.6 Object . . . . .	308
21.7.7 MindMap . . . . .	308



21.7.8 Network (nwdiag) . . . . .	309
21.7.9 Sequence . . . . .	310
21.7.10 State . . . . .	310
21.7.11 Timing . . . . .	311
21.7.12 Work Breakdown Structure (WBS) . . . . .	312
21.7.13 Wireframe (SALT) . . . . .	312
21.8 Appendix: Examples on all diagram with style . . . . .	313
21.8.1 Activity . . . . .	314
21.8.2 Archimate . . . . .	315
21.8.3 Class . . . . .	317
21.8.4 Component, Deployment, Use-Case . . . . .	318
21.8.5 Gantt project planning . . . . .	319
21.8.6 Object . . . . .	320
21.8.7 MindMap . . . . .	322
21.8.8 Network (nwdiag) . . . . .	323
21.8.9 Sequence . . . . .	324
21.8.10 State . . . . .	326
21.8.11 Timing . . . . .	327
21.8.12 Work Breakdown Structure (WBS) . . . . .	329
21.8.13 Wireframe (SALT) . . . . .	330
<b>22 Creole</b> . . . . .	<b>332</b>
22.1 Emphasized text . . . . .	332
22.2 Lists . . . . .	332
22.3 Escape character . . . . .	333
22.4 Horizontal lines . . . . .	333
22.5 Headings . . . . .	334
22.6 Legacy HTML . . . . .	334
22.7 Code . . . . .	335
22.8 Table . . . . .	336
22.8.1 Create a table . . . . .	336
22.8.2 Add color on rows or cells . . . . .	337
22.8.3 Add color on border and text . . . . .	337
22.8.4 No border or same color as the background . . . . .	337
22.8.5 Bold header or not . . . . .	338
22.9 Tree . . . . .	338
22.10 Special characters . . . . .	340
22.11 OpenIconic . . . . .	340
22.12 Appendix: Examples of "Creole List" on all diagrams . . . . .	341
22.12.1 Activity . . . . .	341
22.12.2 Class . . . . .	342
22.12.3 Component, Deployment, Use-Case . . . . .	343
22.12.4 Gantt project planning . . . . .	344
22.12.5 Object . . . . .	344
22.12.6 MindMap . . . . .	345
22.12.7 Network (nwdiag) . . . . .	345
22.12.8 Note . . . . .	345
22.12.9 Sequence . . . . .	346
22.12.10 State . . . . .	346
22.13 Appendix: Examples of "Creole horizontal lines" on all diagrams . . . . .	346
22.13.1 Activity . . . . .	346
22.13.2 Class . . . . .	347
22.13.3 Component, Deployment, Use-Case . . . . .	348
22.13.4 Gantt project planning . . . . .	349
22.13.5 Object . . . . .	349
22.13.6 MindMap . . . . .	349
22.13.7 Network (nwdiag) . . . . .	350
22.13.8 Note . . . . .	350



22.13.9 Sequence . . . . .	351
22.13.10 state . . . . .	351
22.14 Style equivalent (between Creole and HTML) . . . . .	351
<b>23 Defining and using sprites</b>	<b>353</b>
23.1 Changing colors . . . . .	354
23.2 Encoding Sprite . . . . .	354
23.3 Importing Sprite . . . . .	354
23.4 Examples . . . . .	354
23.5 StdLib . . . . .	355
23.6 Listing Sprites . . . . .	355
<b>24 Skinparam command</b>	<b>357</b>
24.1 Usage . . . . .	357
24.2 Nested . . . . .	357
24.3 Black and White . . . . .	357
24.4 Shadowing . . . . .	358
24.5 Reverse colors . . . . .	358
24.6 Colors . . . . .	359
24.7 Font color, name and size . . . . .	360
24.8 Text Alignment . . . . .	360
24.9 Examples . . . . .	361
24.10 List of all skinparam parameters . . . . .	364
<b>25 Preprocessing</b>	<b>368</b>
25.1 Migration notes . . . . .	368
25.2 Variable definition . . . . .	368
25.3 Boolean expression . . . . .	369
25.3.1 Boolean representation [0 is false] . . . . .	369
25.3.2 Boolean operation and operator [&&,   , ()] . . . . .	369
25.3.3 Boolean builtin functions [%false(), %true(), %not(<exp>)] . . . . .	369
25.4 Conditions [!if, !else, !elseif, !endif] . . . . .	369
25.5 While loop [!while, !endwhile] . . . . .	370
25.6 Procedure [!procedure, !endprocedure] . . . . .	371
25.7 Return function [!function, !endfunction] . . . . .	372
25.8 Default argument value . . . . .	373
25.9 Unquoted procedure or function [!unquoted] . . . . .	374
25.10 Keywords arguments . . . . .	374
25.11 Including files or URL [!include, !include_many, !include_once] . . . . .	375
25.12 Including Subpart [!startsub, !endsub, !includesub] . . . . .	376
25.13 Builtin functions [%] . . . . .	376
25.14 Logging [!log] . . . . .	377
25.15 Memory dump [!memory_dump] . . . . .	377
25.16 Assertion [!assert] . . . . .	378
25.17 Building custom library [!import, !include] . . . . .	378
25.18 Search path . . . . .	379
25.19 Argument concatenation [##] . . . . .	379
25.20 Dynamic invocation [%invoke_procedure(), %call_user_func()] . . . . .	379
25.21 Evaluation of addition depending of data types [+] . . . . .	380
25.22 Preprocessing JSON . . . . .	381
<b>26 Unicode</b>	<b>382</b>
26.1 Examples . . . . .	382
26.2 Charset . . . . .	384
<b>27 Standard Library</b>	<b>385</b>
27.1 List of Standard Library . . . . .	385
27.2 ArchiMate [archimate] . . . . .	386
27.3 AWS library [aws] . . . . .	387



27.4 Amazon Labs AWS Library [awslib]	388
27.5 Azure library [azure]	389
27.6 C4 Library [C4]	390
27.7 Cloud Insight [cloudinsight]	391
27.8 Cloudogu [cloudogu]	392
27.9 Elastic library [elastic]	393
27.10 Google Material Icons [material]	395
27.11 Kubernetes [kubernetes]	396
27.12 Logos [logos]	397
27.13 Office [office]	399
27.14 Open Security Architecture (OSA) [osa]	401
27.15 Tupadr3 library [tupadr3]	403

