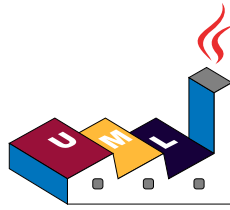


# Drawing UML with PlantUML



## PlantUML Language Reference Guide

(Version 1.2020.23)

**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
- Network diagram (nwdiag)
- Wireframe graphical interface
- Archimate diagram
- Specification and Description Language (SDL)
- Dita 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

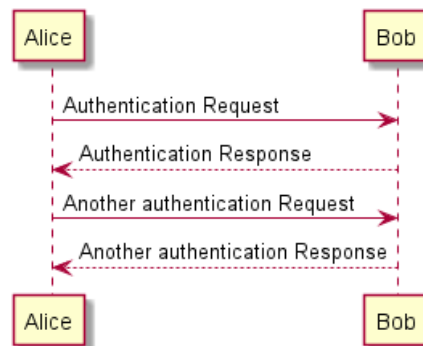
The sequence `->` is used to draw a message between two participants. Participants do not have to be explicitly declared.

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**.

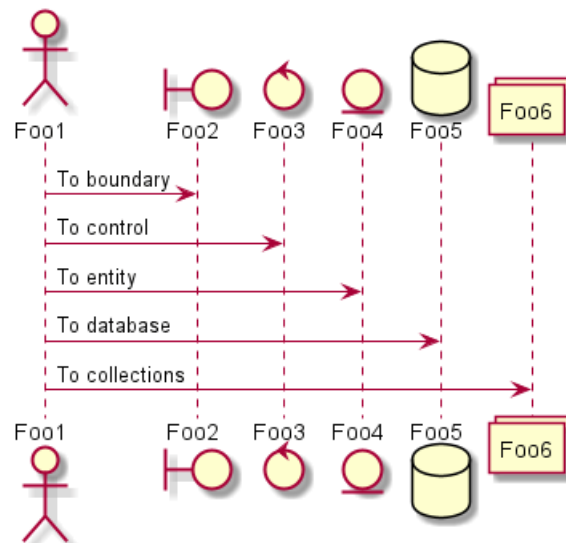
Using these other keywords to declare participants will **change the shape** of the participant representation:

- actor
- boundary
- control
- entity
- database
- collections

```
@startuml
actor Foo1
boundary Foo2
control Foo3
entity Foo4
database Foo5
collections Foo6
Foo1 -> Foo2 : To boundary
Foo1 -> Foo3 : To control
Foo1 -> Foo4 : To entity
Foo1 -> Foo5 : To database
Foo1 -> Foo6 : To collections
```



```
@enduml
```

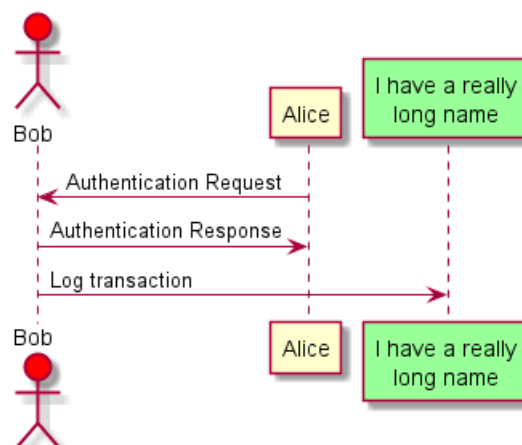


Rename a participant using the `as` keyword.

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
  '/
```

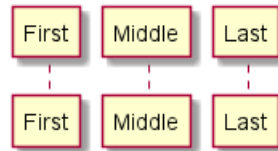
```
Alice->Bob: Authentication Request
Bob->Alice: Authentication Response
Bob->L: Log transaction
@enduml
```



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
```

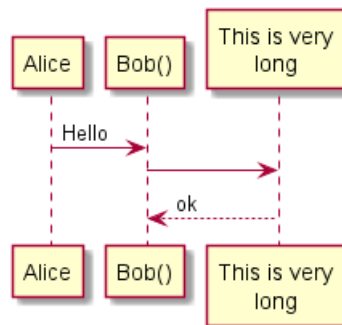




### 1.3 Use non-letters in participants

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

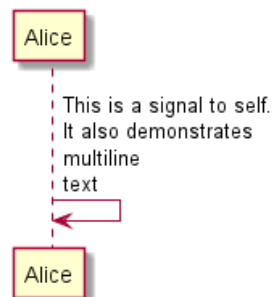


### 1.4 Message to Self

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

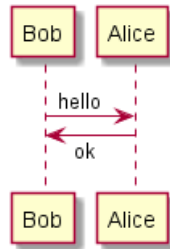
You can put the text of the response message below the arrow, with the skinparam `responseMessageBelowArrow true` command.



```

@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 <->

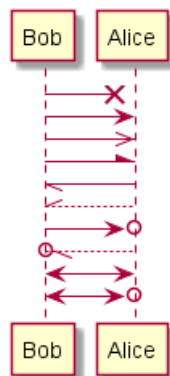
```

@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

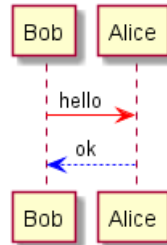
```



## 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
```



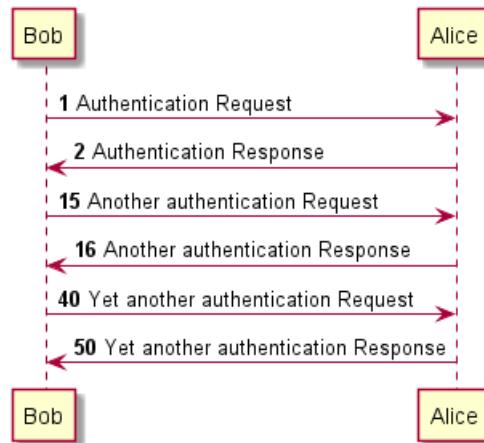
You can specify a startnumber with `autonumber //start//`, and also an increment with `autonumber //start// //increment//`.

```
@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.

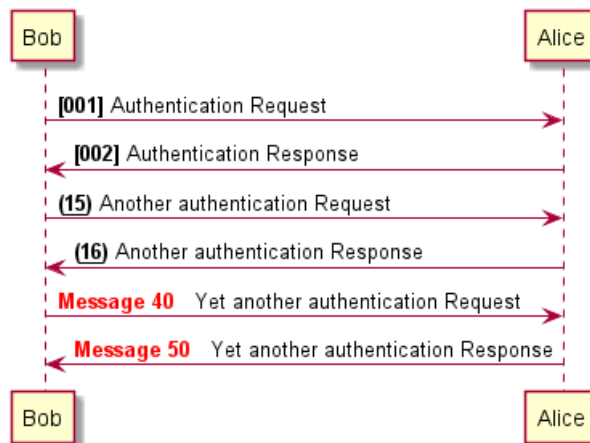
```

@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]"
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response

autonumber stop
Bob -> Alice : dummy
  
```



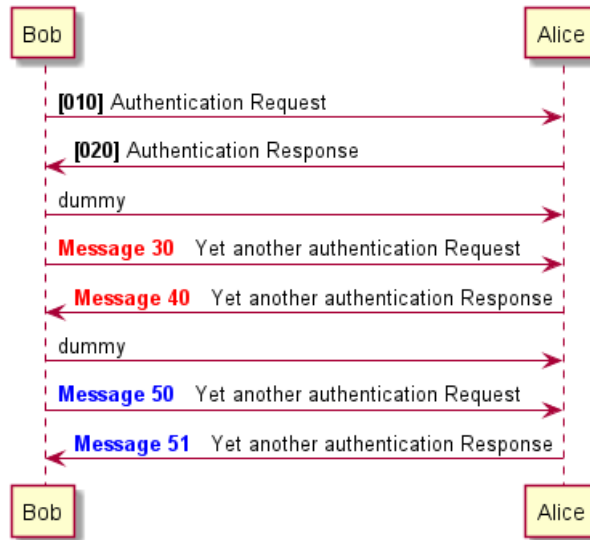
```

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

```



## 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%

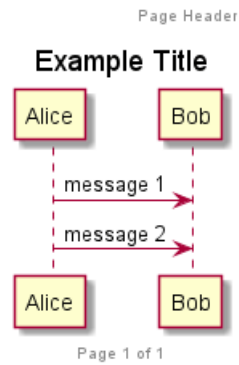
title Example Title

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

@enduml

```





## 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
```

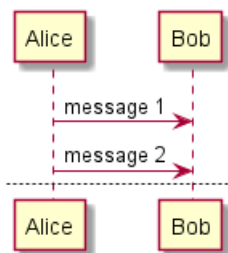
```
Alice -> Bob : message 4
```

```
newpage A title for the\nlast 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
```

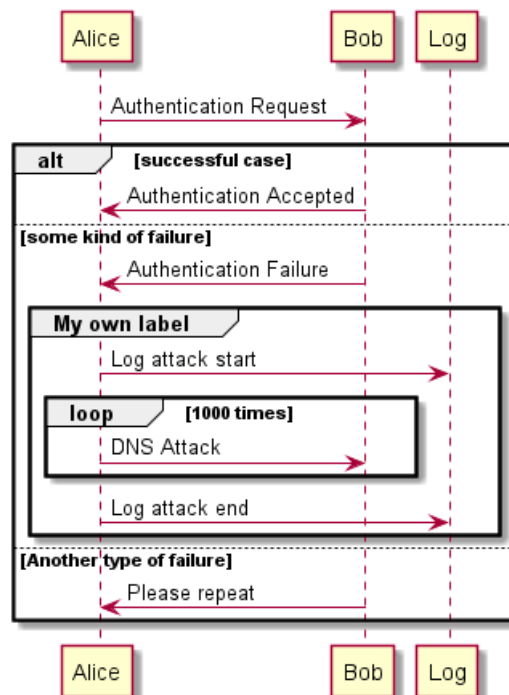
```
    end
```

```
else Another type of failure
```

```
    Bob -> Alice: Please repeat
```

```
end
```

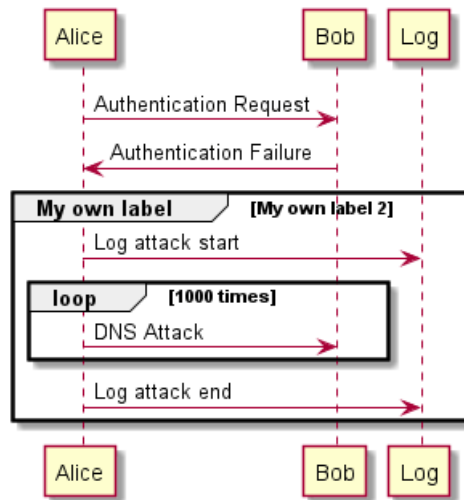
```
@enduml
```



## 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

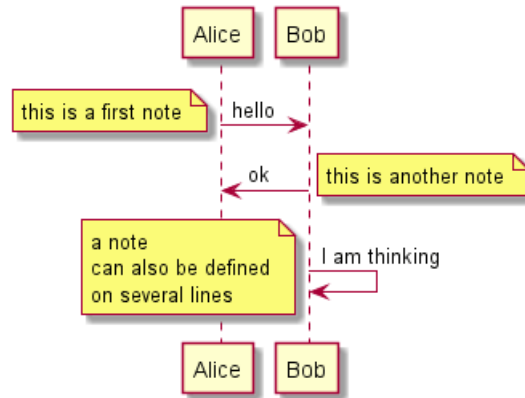
It is possible to put notes on message using the note left or note right keywords *just after the message*.

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.
  
```

```

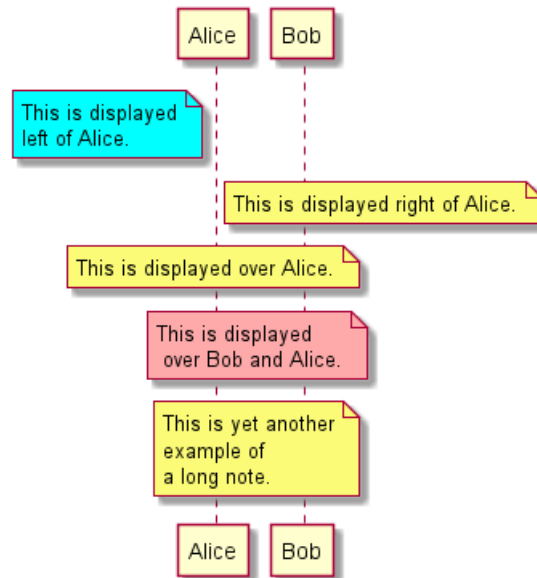
note over Alice: This is displayed over Alice.
  
```

```

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

```

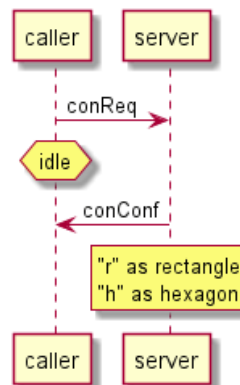
note over Bob, Alice
This is yet another
example of
a long note.
end note
@enduml
  
```



## 1.15 Changing notes shape

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

```
@startuml
caller -> server : conReq
hnote over caller : idle
caller <- server : conConf
rnote over server
  "r" as rectangle
  "h" as hexagon
endnote
@enduml
```



## 1.16 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
```

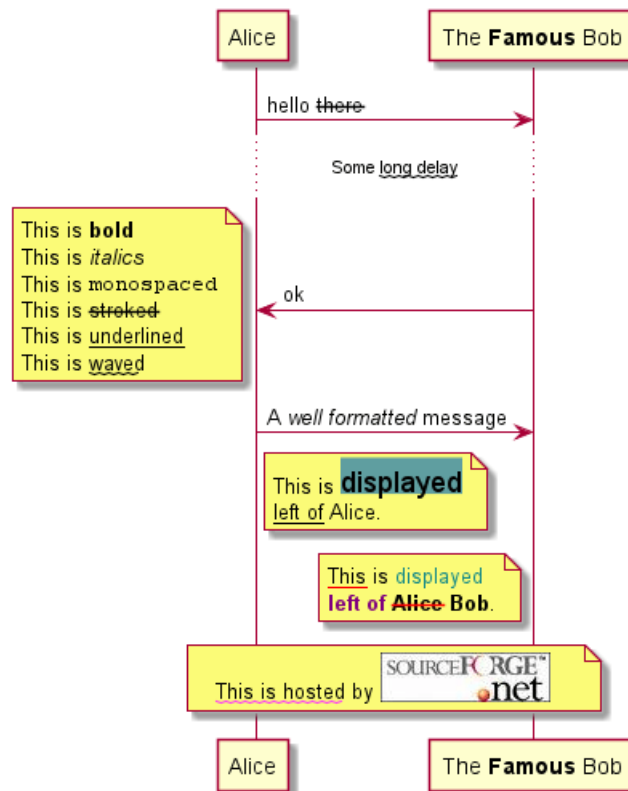


```

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

```



## 1.17 Divider

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

```

@startuml
== Initialization ==

```

```

Alice -> Bob: Authentication Request

```



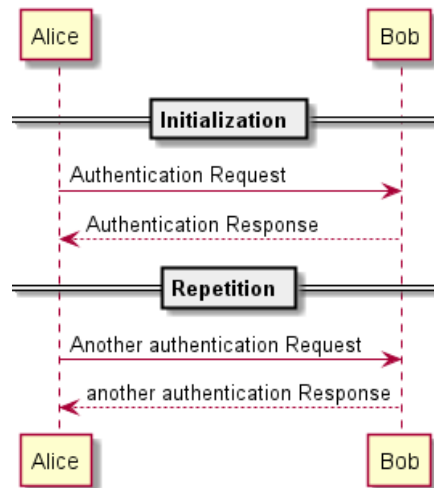
```
Bob --> Alice: Authentication Response
```

```
== Repetition ==
```

```
Alice -> Bob: Another authentication Request
```

```
Alice <-- Bob: another authentication Response
```

```
@enduml
```



## 1.18 Reference

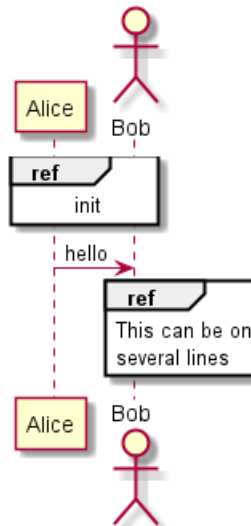
You can use reference in a diagram, using the keyword `ref` over.

```
@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
```



## 1.19 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
```

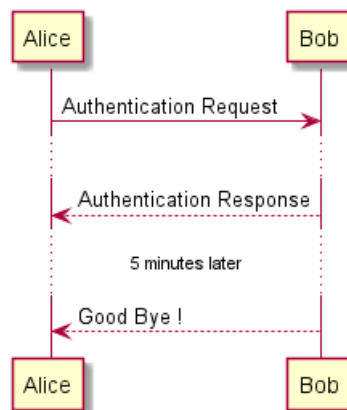
```
...
```

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

```
...5 minutes later...
```

```
Bob --> Alice: Good Bye !
```

```
@enduml
```



## 1.20 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
```

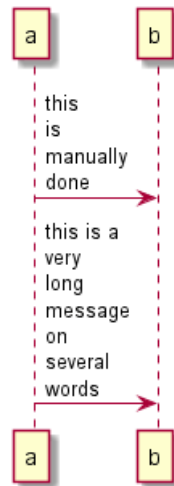
```
a -> b :this\nis\nmanually\ndone
```

```
a -> b :this is a very long message on several words
```

```
@enduml
```







## 1.21 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
```



## 1.22 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

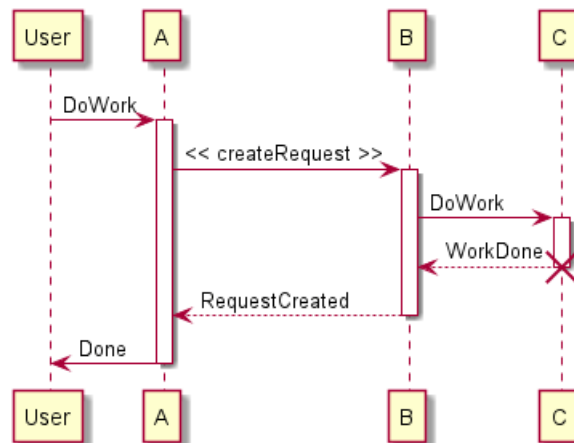
B -> C: DoWork
activate C
C --> B: WorkDone
destroy C

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

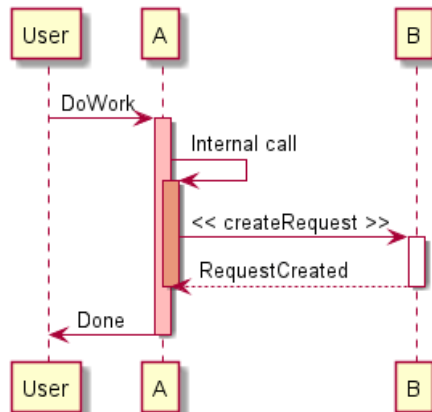
```



```

deactivate A
@enduml

```

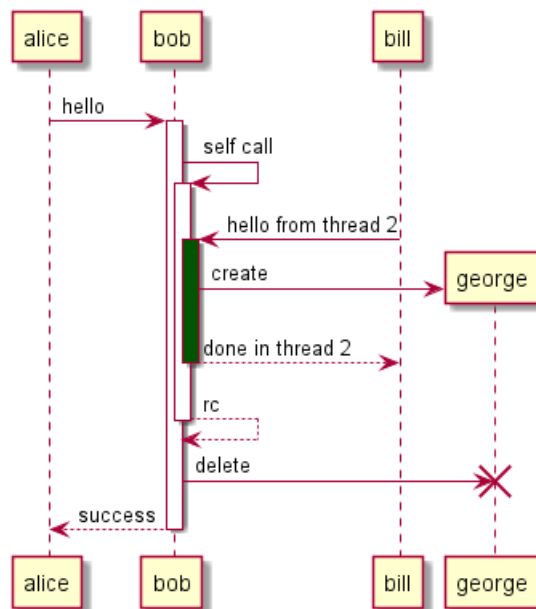


Autoactivation is possible and works with the return keywords:

```

@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

```



## 1.23 Return

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

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

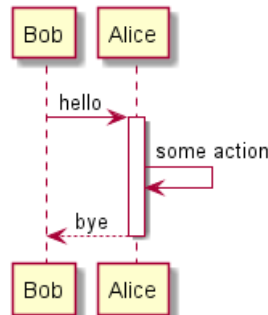
The syntax is `return label` where `label` if provided is any string acceptable for conventional messages.



```

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

```



## 1.24 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.

```

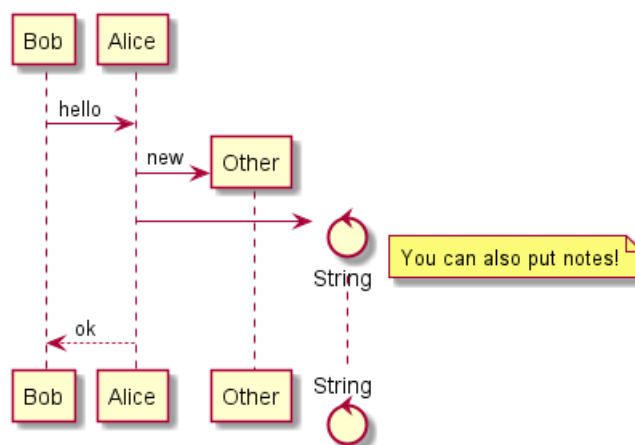
@startuml
Bob -> Alice : hello

create Other
Alice -> Other : new

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

Alice --> Bob : ok
@enduml

```



## 1.25 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)

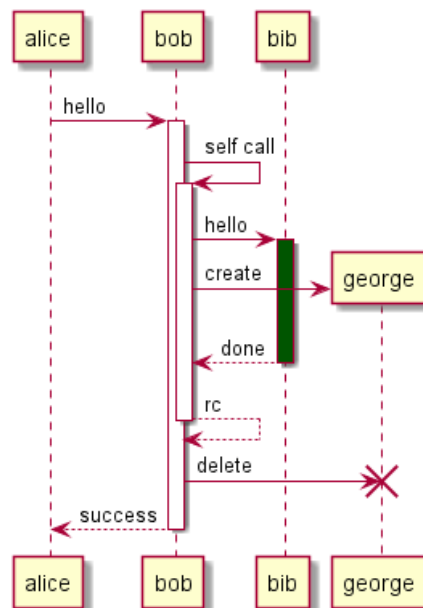


- -- Deactivate the source
- \*\* Create an instance of the target
- !! Destroy an instance of the target

```

@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

```



## 1.26 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

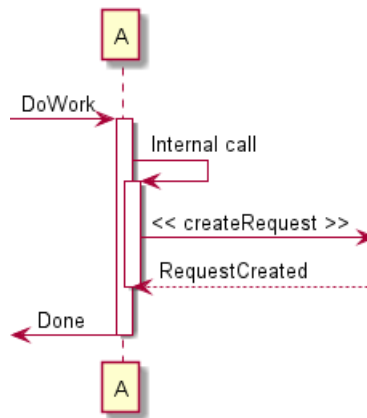
A -> A: Internal call
activate A

A ->] : << createRequest >>

A<--] : RequestCreated
deactivate A
[<- A: Done
deactivate A
@enduml

```

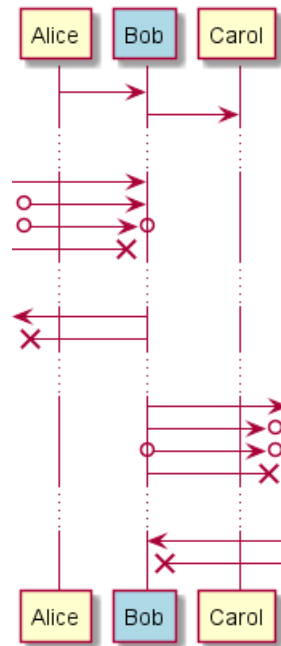




You can also have the following syntax:

```

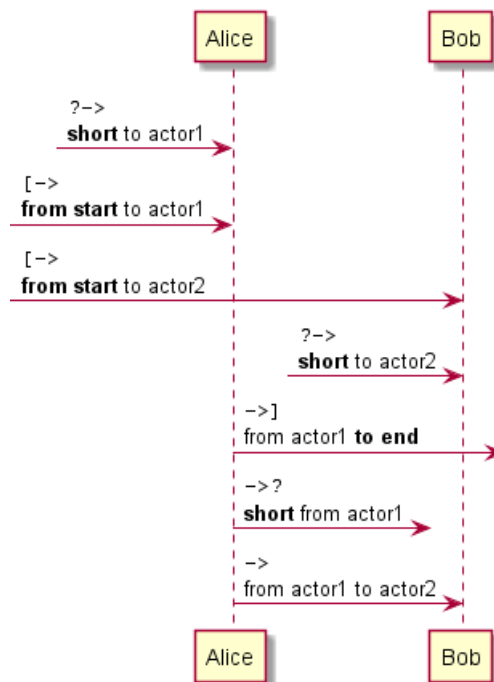
@startuml
participant Alice
participant Bob #lightblue
Alice -> Bob
Bob -> Carol
...
[-> Bob
[o-> Bob
[o->o Bob
[x-> Bob
...
[<- Bob
[x<- Bob
...
Bob ->]
Bob ->o]
Bob o->o]
Bob ->x]
...
Bob <-]
Bob x<-]
@enduml
  
```



## 1.27 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]

## 1.28 Anchors and Duration

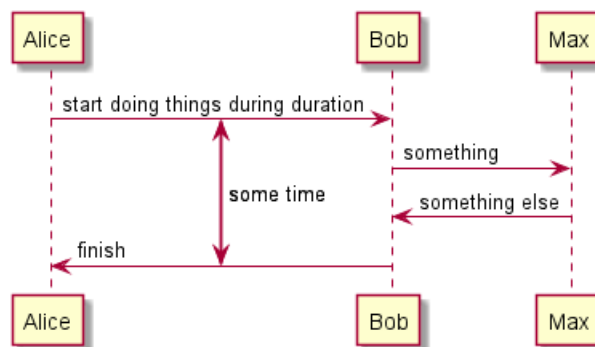
With teoz usage it is possible to add anchors to the diagram and use the anchors to specify duration time.

```
@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
```



## 1.29 Stereotypes and Spots

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

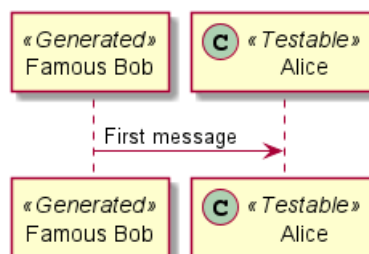
In the stereotype, you can add a spotted character in a colored circle using the syntax (X,color).

```
@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 >>

```

```

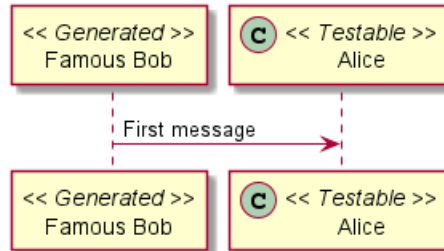
Bob->Alice: First message

```

```

@enduml

```



```

@startuml

```

```

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

```

```

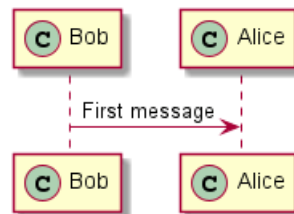
Bob->Alice: First message

```

```

@enduml

```



### 1.30 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

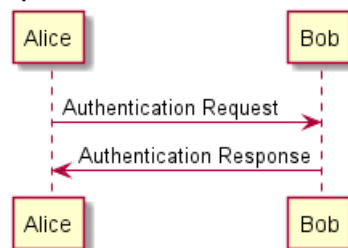
```

```

@enduml

```

#### Simple communication example



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

```

@startuml

```

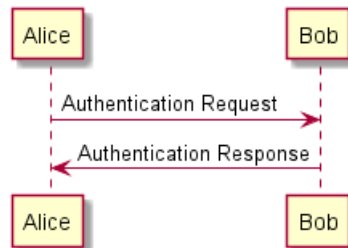


```
title __Simple__ communication example\non several lines
```

```
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
```

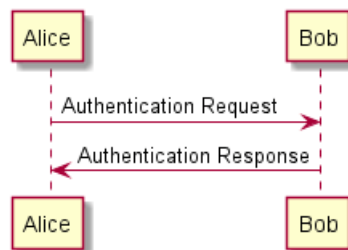
```
title
  <u>Simple</u> communication example
  on <i>several</i> lines and using <font color=red>html</font>
  This is hosted by <img:sourceforge.jpg>
end title
```

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

```
@enduml
```

### Simple communication example on several lines and using html

This is hosted by 



## 1.31 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
```

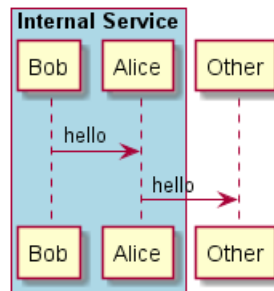


```

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

@enduml

```



### 1.32 Removing Foot Boxes

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

```

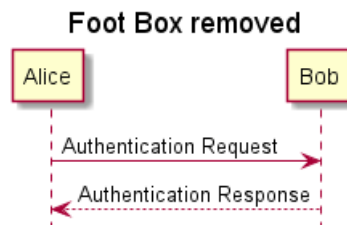
@startuml

hide footbox
title Foot Box removed

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

@enduml

```



### 1.33 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

```

```

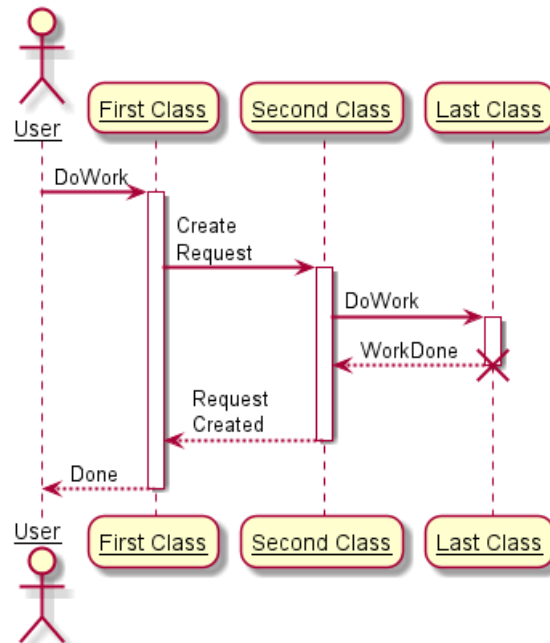
A --> User: Done
deactivate A

```

```

@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

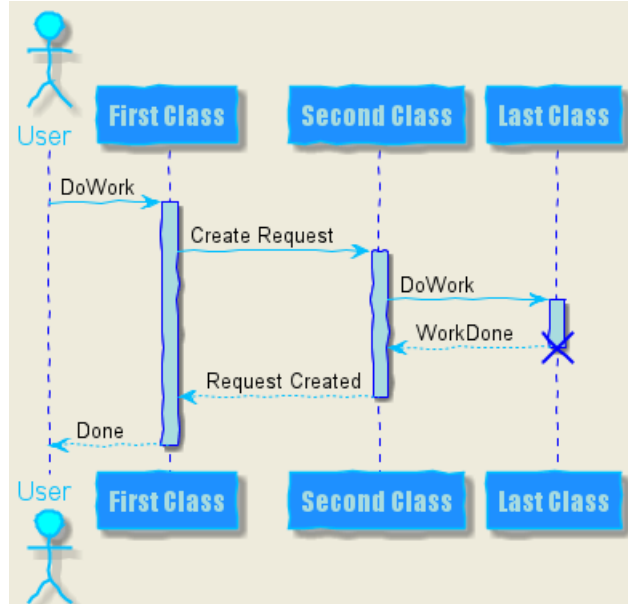
B -> C: DoWork
activate C
C --> B: WorkDone
destroy C

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml

```



### 1.34 Changing padding

It is possible to tune some padding settings.

```

@startuml
skinparam ParticipantPadding 20
skinparam BoxPadding 10

box "Foo1"

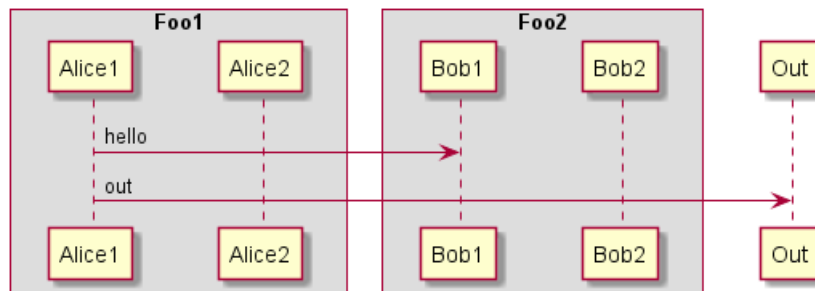
```



```

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

```



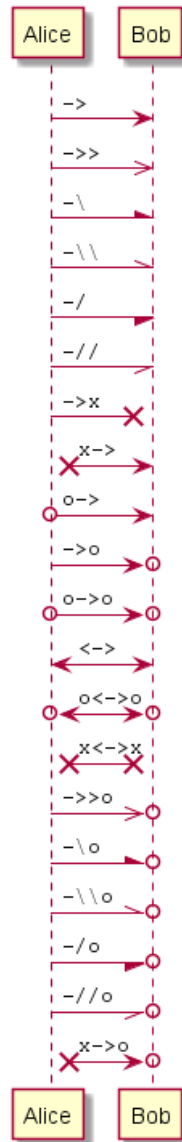
## 1.35 Appendice: Examples of all arrow type

### 1.35.1 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

```



### 1.35.2 Incoming and outgoing 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 ""

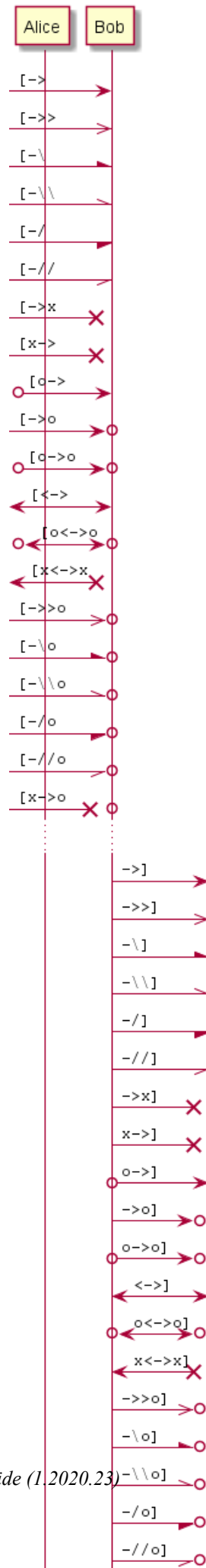
```

```

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

```



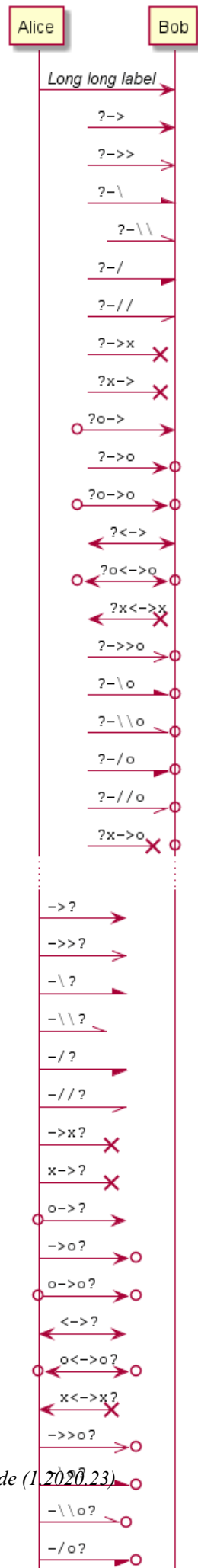


**1.35.3 Short incoming and outgoing messages (with '?')**

```

@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  "
...
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

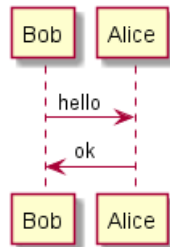
```



## 1.36 Specific SkinParameter

### 1.36.1 By default

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

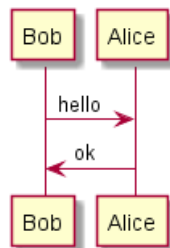


### 1.36.2 lifelineStrategy solid

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
```



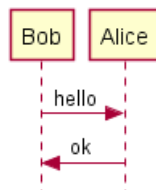
[Ref. QA-2794]

### 1.36.3 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]

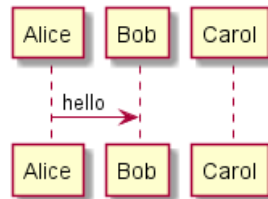


### 1.37 Hide unlinked participant

By default, all participants are displayed.

```
@startuml
participant Alice
participant Bob
participant Carol

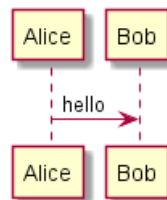
Alice -> Bob : hello
@enduml
```



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

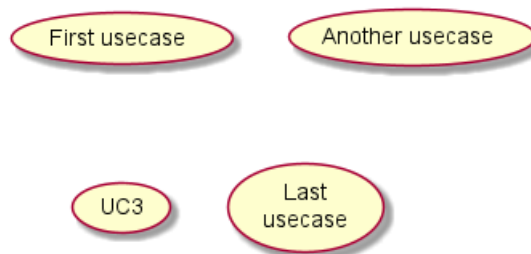
Let's have a few examples:

### 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\nusecase) as UC4
@enduml
```



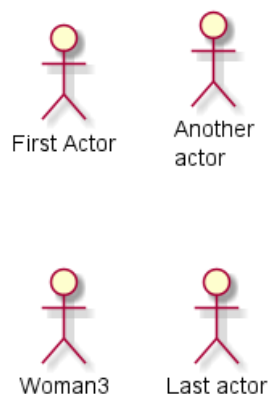
### 2.2 Actors

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\nactor: as Man2
actor Woman3
actor :Last actor: as Person1
@enduml
```



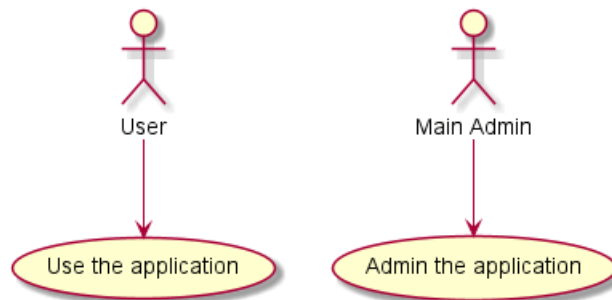
## 2.3 Change Actor style

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.

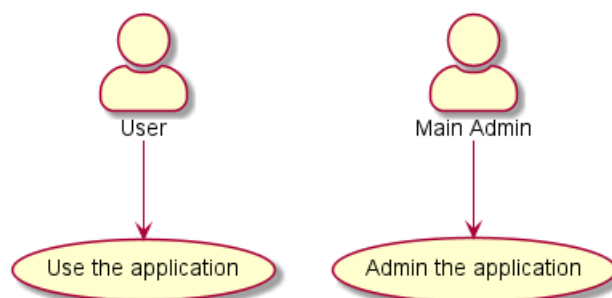
### 2.3.1 Stick man (*by default*)

```
@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
```

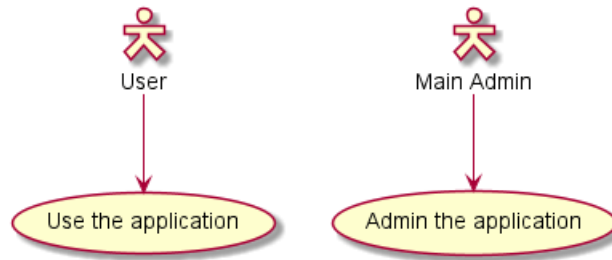


[Ref. QA-10493]

### 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.
```

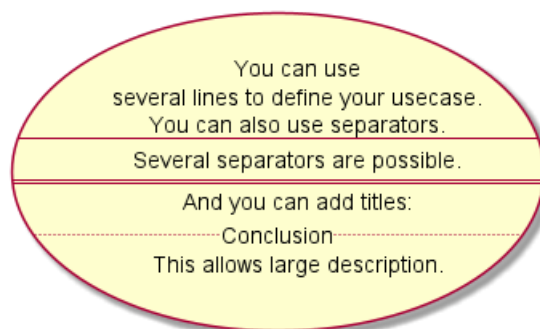
```
==
```

```
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
```

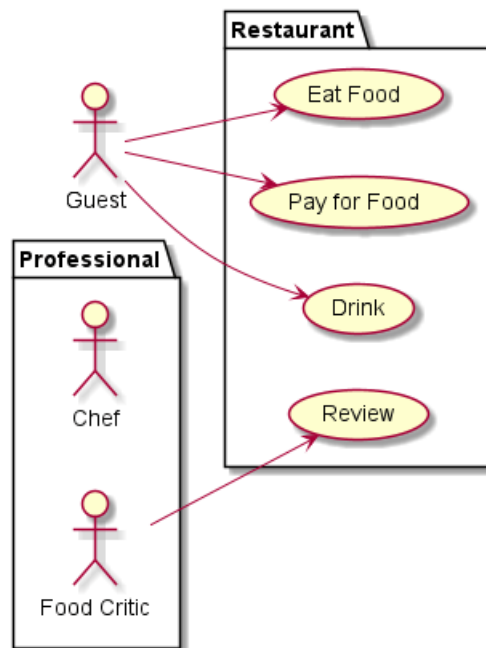




```

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

```

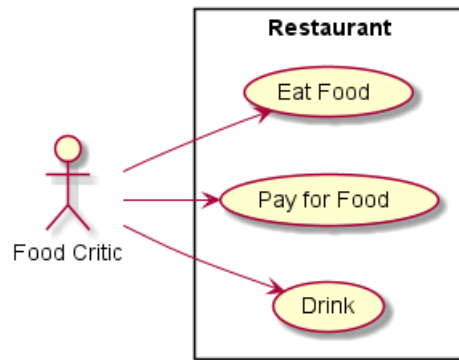


You can use `rectangle` to change the display of the package.

```

@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

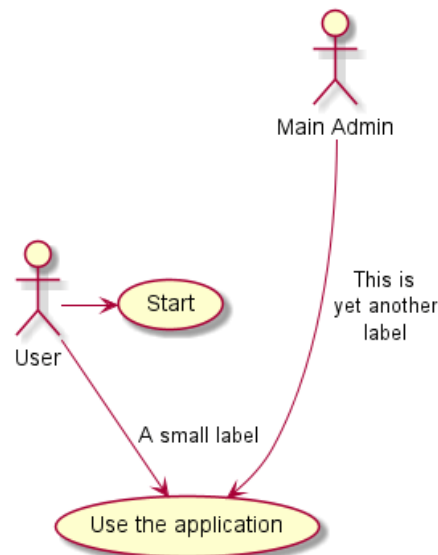
To link actors and use cases, the arrow --> is used.

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

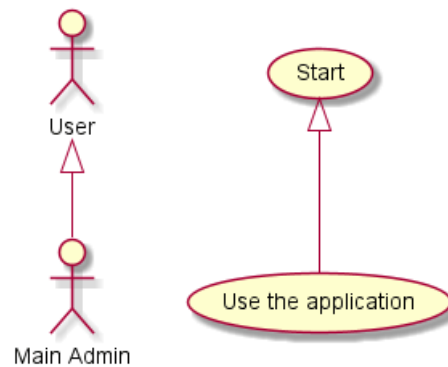
If one actor/use case extends another one, you can use the symbol <|--.

```
@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 keywords, 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)
```

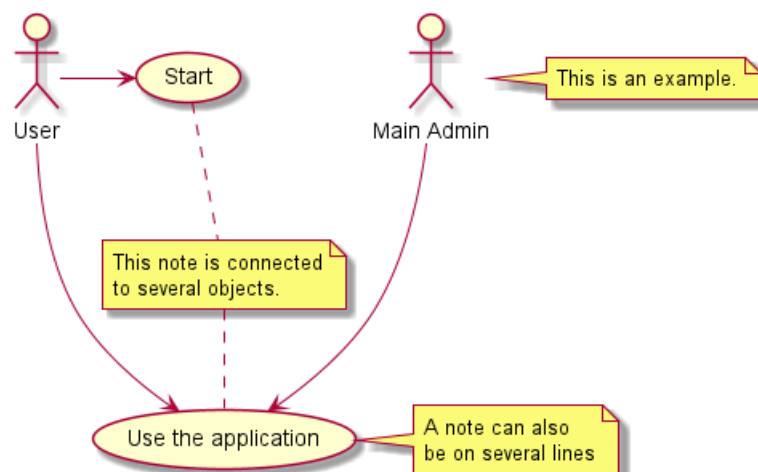
```
  A note can also
  be on several lines
end note
```

```
note "This note is connected\nto several objects." as N2
```

```
(Start) .. N2
```

```
N2 .. (Use)
```

```
@enduml
```



## 2.9 Stereotypes

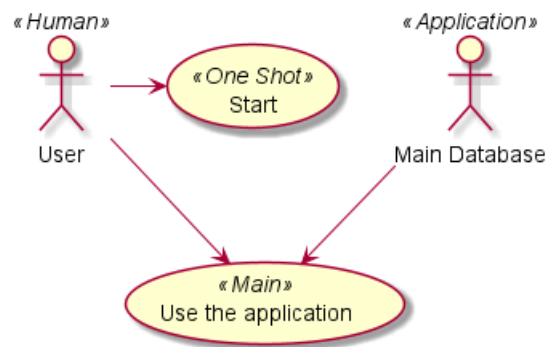
You can add stereotypes while defining actors and use cases using << and >>.

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

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

MySQL --> (Use)

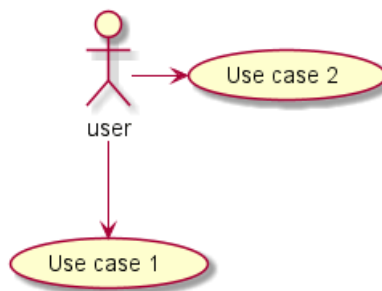
@enduml
```



## 2.10 Changing arrows direction

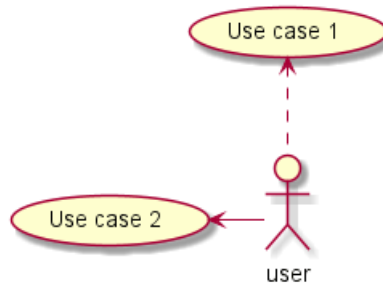
By default, links between classes have two dashes -- and are vertically oriented. It is possible to use horizontal link by putting a single dash (or dot) like this:

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



You can also change directions by reversing the link:

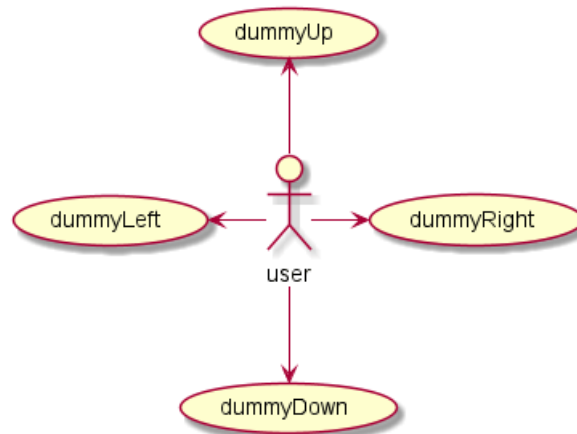
```
@startuml
(Use case 1) <.. :user:
(Use case 2) <- :user:
@enduml
```



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
  
```



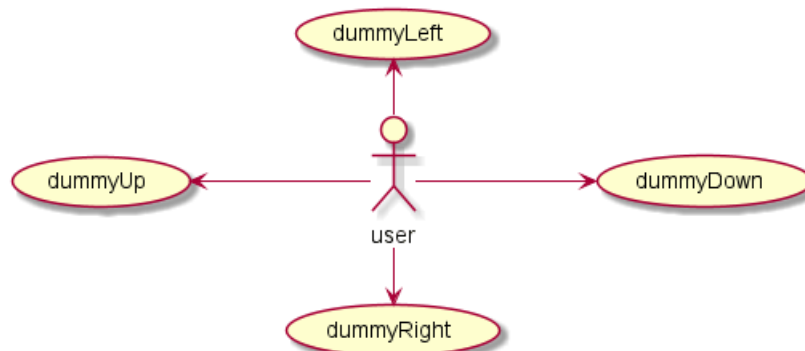
You can shorten the arrow by using only the first character of the direction (for example, -d- instead of -down-) or the two first characters (-do-).

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

And with the left to right direction parameter:

```

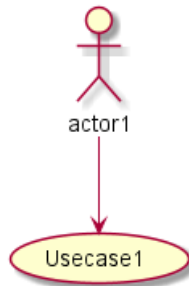
@startuml
left to right direction
:user: -left-> (dummyLeft)
:user: -right-> (dummyRight)
:user: -up-> (dummyUp)
:user: -down-> (dummyDown)
@enduml
  
```



## 2.11 Splitting diagrams

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

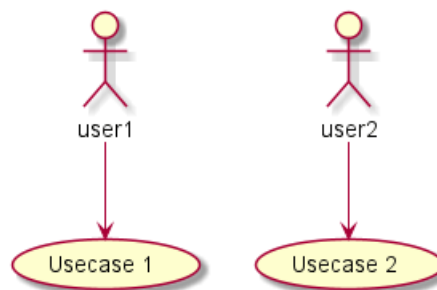
```
@startuml
:actor1: --> (Usecase1)
newpage
:actor2: --> (Usecase2)
@enduml
```



## 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
```

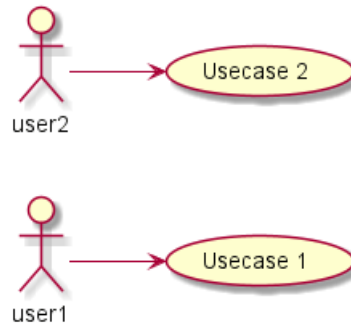


You may change to **left to right** using the left to right direction command. The result is often better with this direction.

```
@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

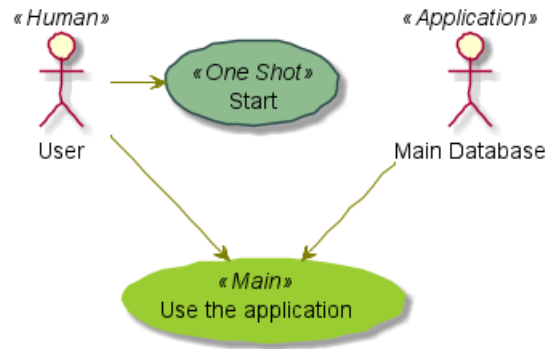
ActorBackgroundColor<< Human >> Gold
}

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

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

MySql --> (Use)

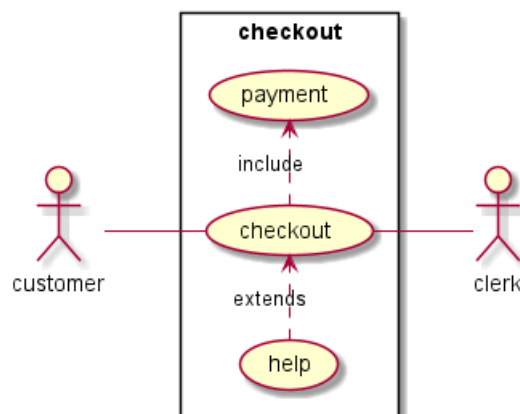
@enduml
```



## 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
  
```





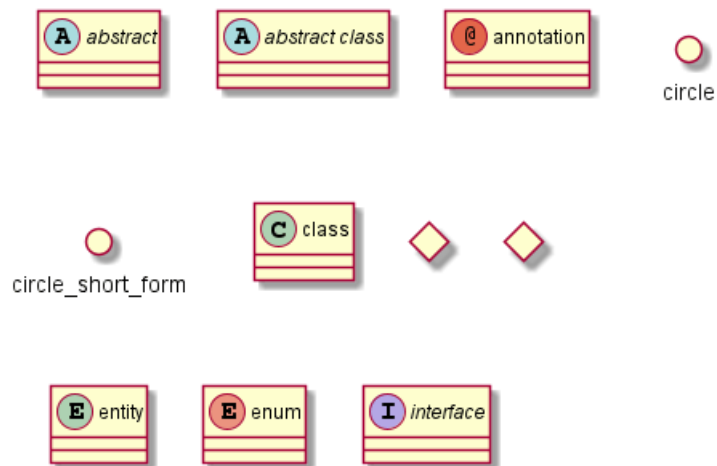
## 3 Class Diagram

### 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.

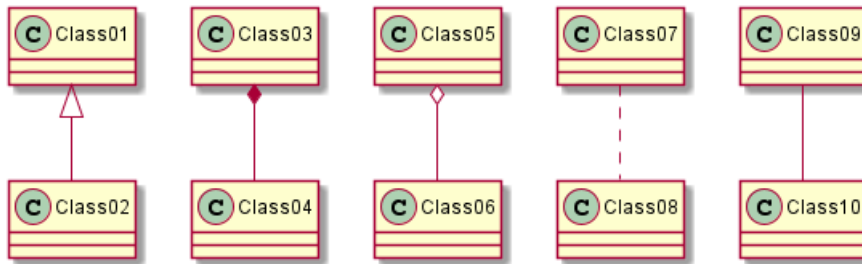
Knowing those rules, it is possible to draw the following drawings:

```

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

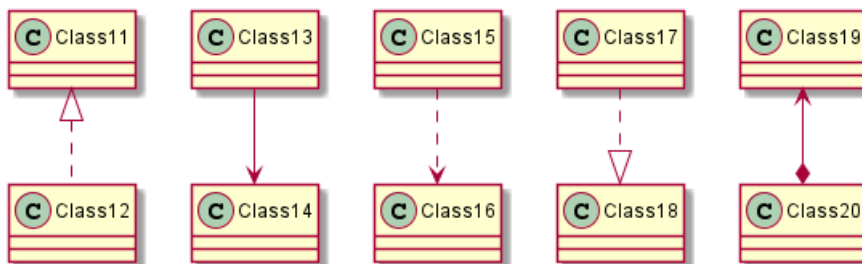
```





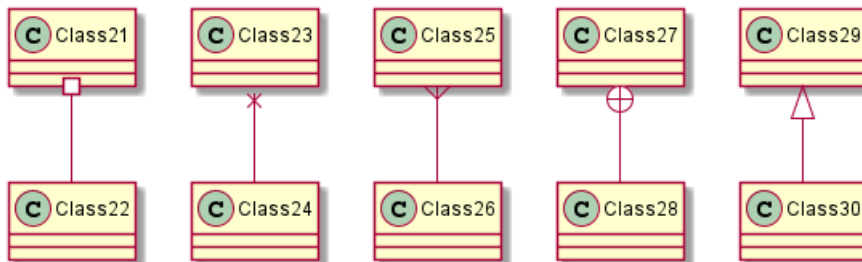
```

@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

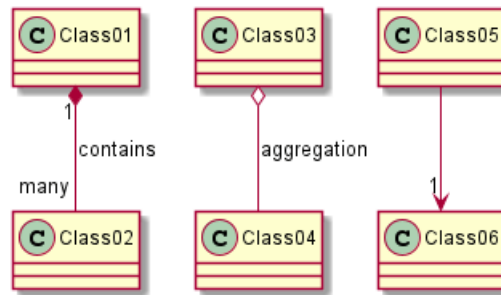
It is possible to add a label on the relation, using `:`, followed by the text of the label.

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
  
```





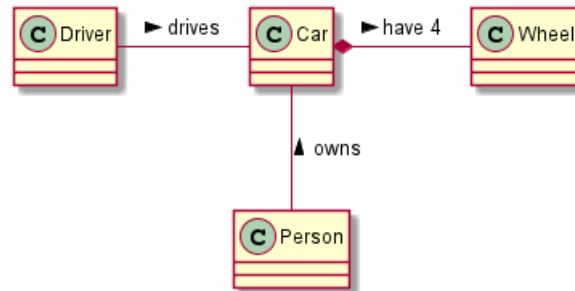
You can add an extra arrow pointing at one object showing which object acts on the other object, using < or > at the begin or at the end of the label.

```

@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.

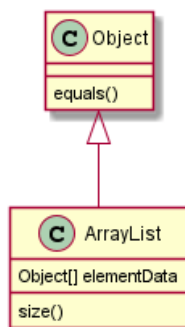
The system checks for parenthesis to choose between methods and fields.

```

@startuml
Object <|-- ArrayList

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

@enduml
  
```



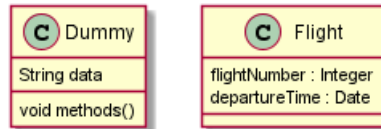
It is also possible to group between brackets {} all fields and methods.



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

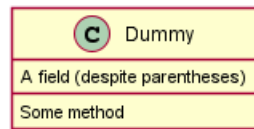
```
@startuml
class Dummy {
    String data
    void methods()
}

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
```

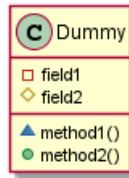


### 3.5 Defining visibility

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
```

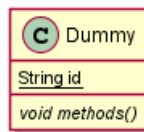


### 3.6 Abstract and Static

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
}
```



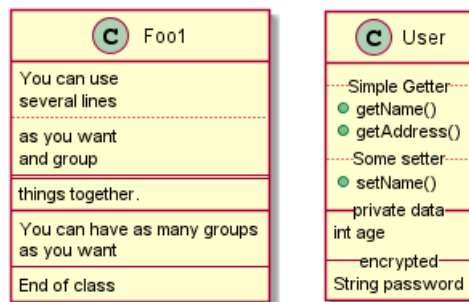
```

==
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.

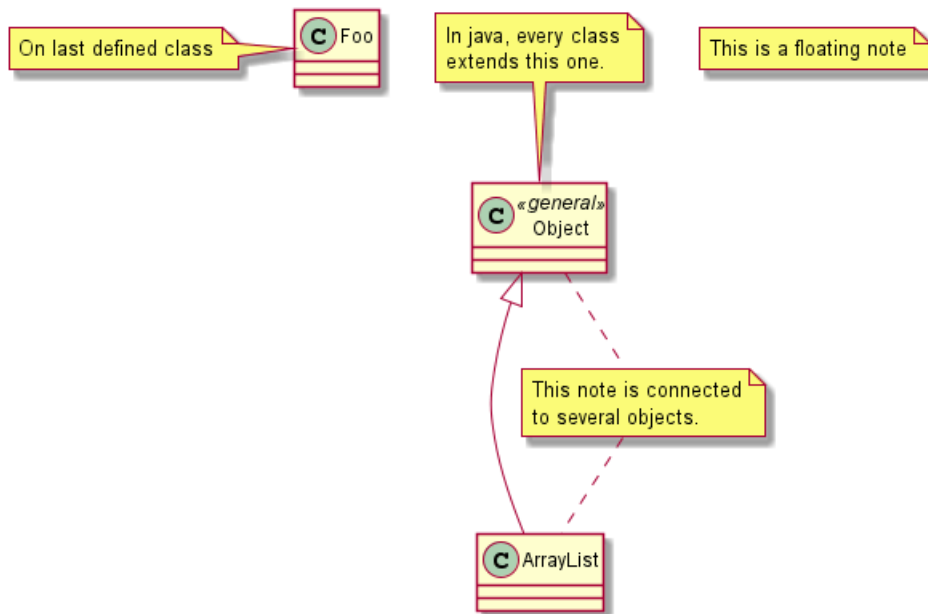
note "This is a floating note" as N1
note "This note is connected\nto several objects." as N2
Object .. N2
N2 .. ArrayList

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

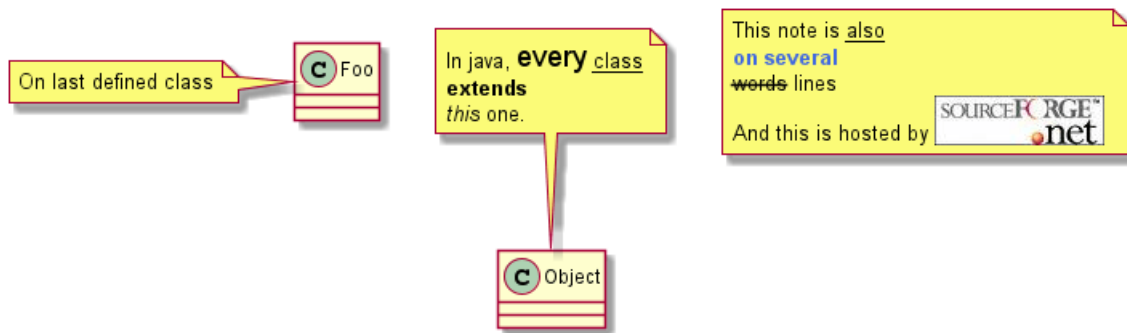
note top of Object
  In java, <size:18>every</size> <u>class</u>
  <b>extends</b>
  <i>this</i> one.
end note

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

```



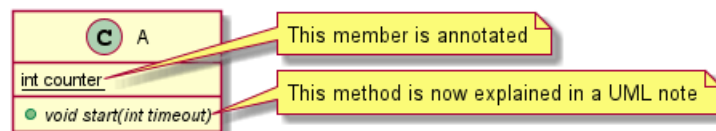


### 3.10 Note on field (field, attribut, member) or method

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

#### 3.10.1 Note on field or method

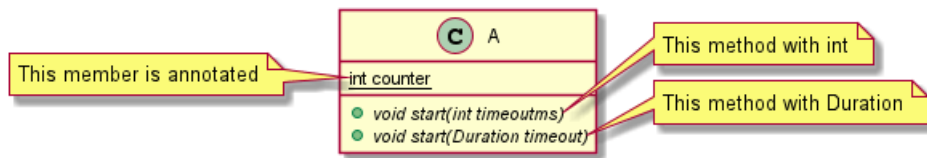
```
@startuml
class A {
{static} int counter
+void {abstract} start(int timeout)
}
note right of A::counter
  This member is annotated
end note
note right of A::start
  This method is now explained in a UML note
end note
@enduml
```



#### 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`.

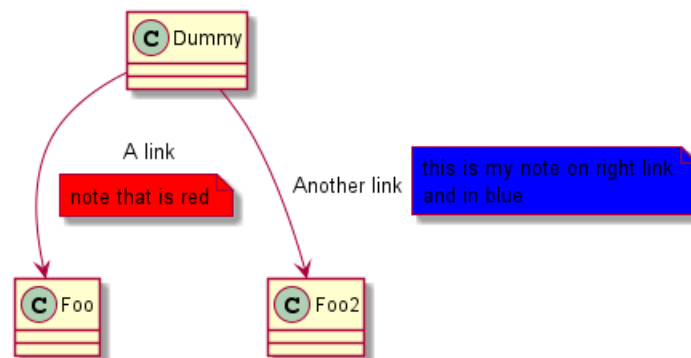
You can also use `note left on link`, `note right on link`, `note top on link`, `note bottom on link` if you want to change the relative position of the note with the label.

```
@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

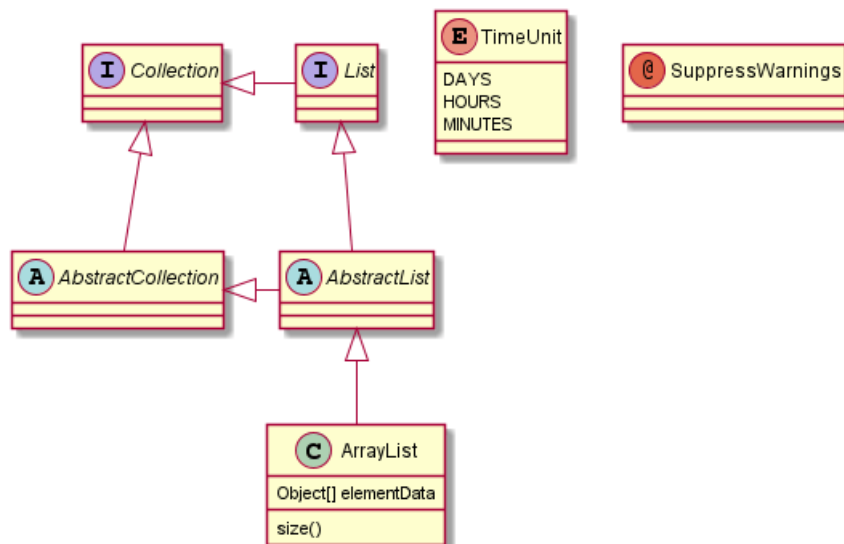
class ArrayList {
    Object[] elementData
    size()
}

enum TimeUnit {
    DAYS
    HOURS
    MINUTES
}

annotation SuppressWarnings

@enduml

```



### 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

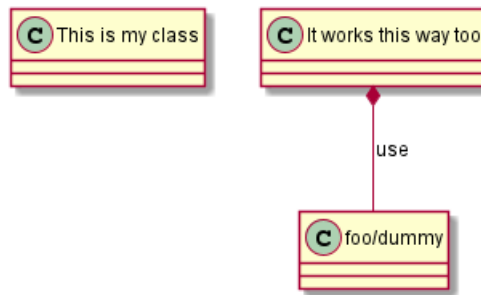
```

@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()
}
```

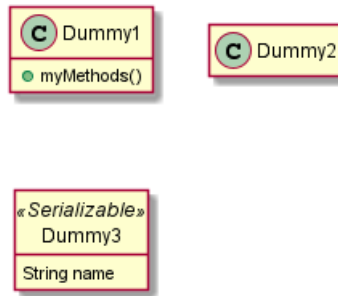
```
class Dummy2 {
  +hiddenMethod()
}
```

```
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.

```

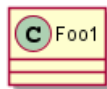
@startuml

class Foo1
class Foo2

Foo2 *-- Foo1

hide Foo2

@enduml
  
```



### 3.16 Use generics

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

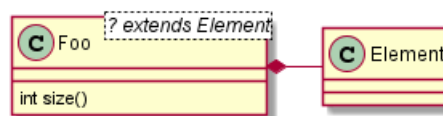
```

@startuml

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

Foo *-- Element

@enduml
  
```



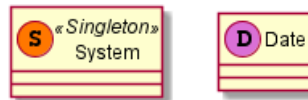
It is possible to disable this drawing using skinparam genericDisplay old command.

### 3.17 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
```



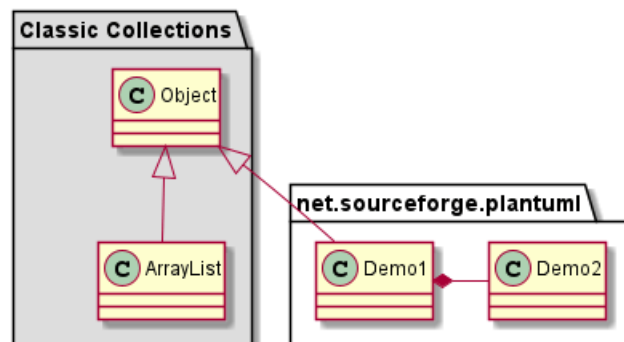
### 3.18 Packages

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
```



### 3.19 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
}

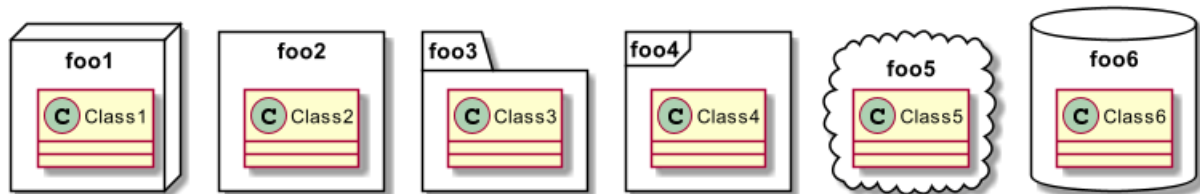
package foo4 <<Frame>> {
  class Class4
}

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

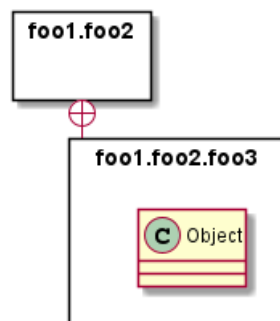
package foo1.foo2 {
}

package foo1.foo2.foo3 {
  class Object
}

foo1.foo2 +-- foo1.foo2.foo3

@enduml

```



### 3.20 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

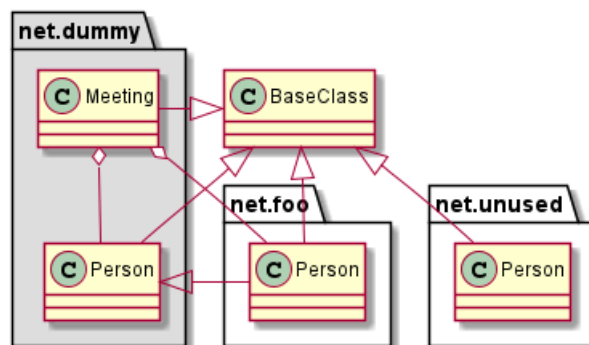
    .BaseClass <|-- Meeting
}

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

    net.dummy.Meeting o-- Person
}

BaseClass <|-- net.unused.Person

@enduml
```



### 3.21 Automatic namespace creation

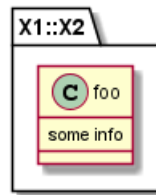
You can define another separator (other than the dot) using the command : set namespaceSeparator ???.

```
@startuml

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

@enduml
```

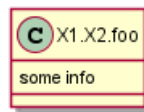




You can disable automatic package creation using the command set namespaceSeparator none.

```

@startuml
set namespaceSeparator none
class X1.X2.foo {
    some info
}
@enduml
  
```



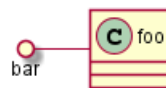
### 3.22 Lollipop interface

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

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

```

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

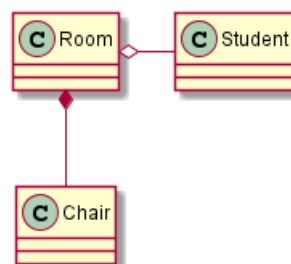


### 3.23 Changing arrows direction

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:

```

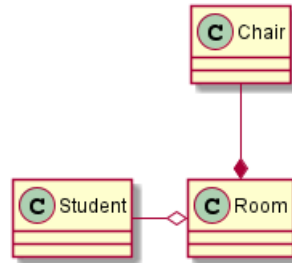
@startuml
Room o- Student
Room *-- Chair
@enduml
  
```





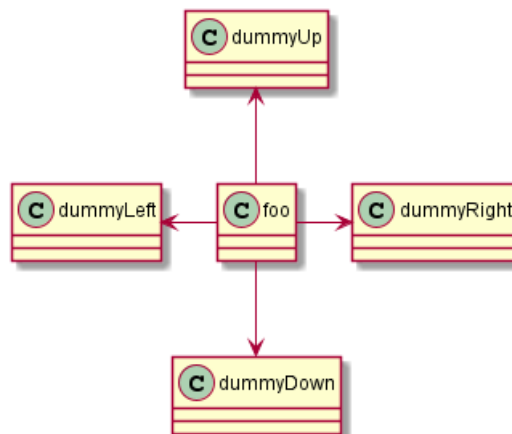
You can also change directions by reversing the link:

```
@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
```

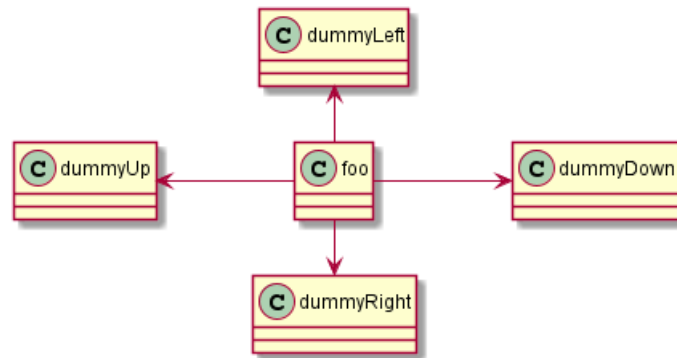


You can shorten the arrow by using only the first character of the direction (for example, -d- instead of -down-) or the two first characters (-do-).

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
```



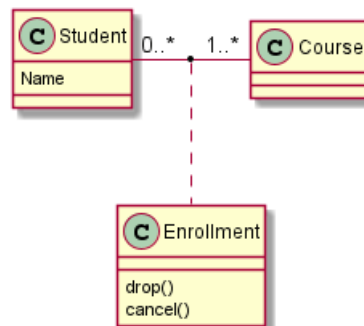
### 3.24 Association classes

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
  
```

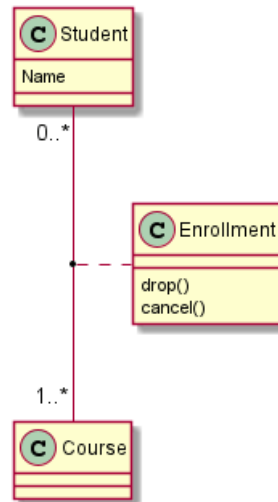


You can define it in another direction:

```

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

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



### 3.25 Association on same classe

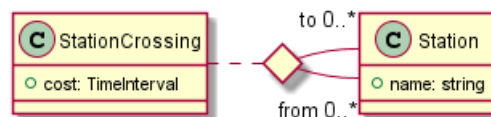
```

@startuml
class Station {
    +name: string
}

class StationCrossing {
    +cost: TimeInterval
}

<> diamond

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



[Ref. Incubation: Associations]

### 3.26 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
}
  
```



```

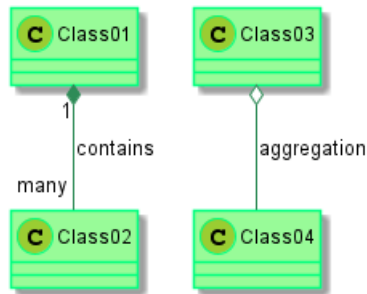
}
skinparam stereotypeCBackgroundColor YellowGreen

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

Class03 o-- Class04 : aggregation

@enduml

```



### 3.27 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
}

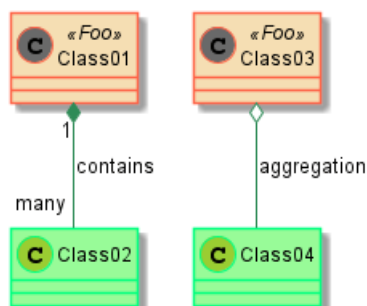
skinparam stereotypeCBackgroundColor YellowGreen
skinparam stereotypeCBackgroundColor<< Foo >> DimGray

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

Class03 o-- Class04 : aggregation

@enduml

```



### 3.28 Color gradient

It's possible to declare individual color for classes or note using the # notation.



You can use either standard color name or RGB code.

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

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

depending the direction of the gradient.

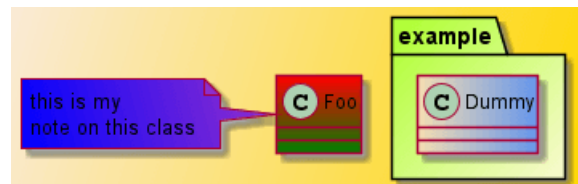
For example, you could have:

```
@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
```



### 3.29 Help on layout

Sometimes, the default layout is not perfect...

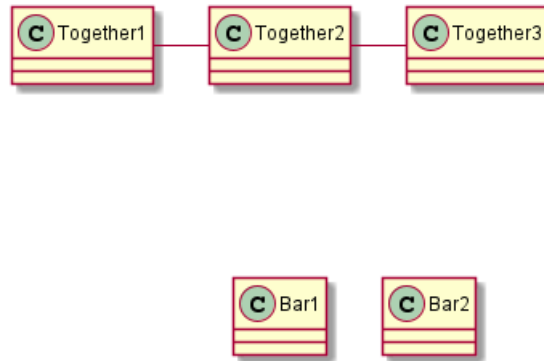
You can use together keyword to group some classes together : the layout engine will try to group them (as if they were in the same package).

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
```





### 3.30 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

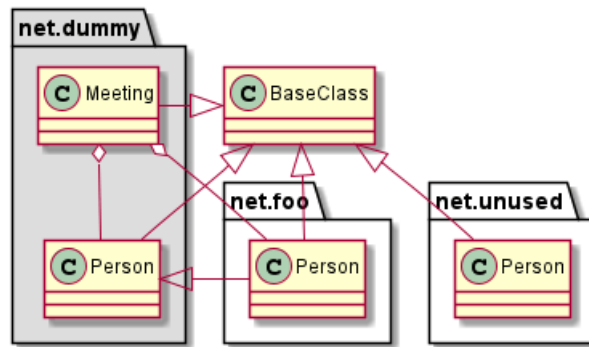
  .BaseClass <|-- Meeting
}

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

  net.dummy.Meeting o-- Person
}

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

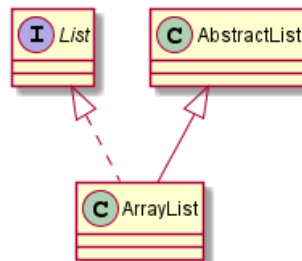
```



### 3.31 Extends and implements

It is also possible to use extends and implements keywords.

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

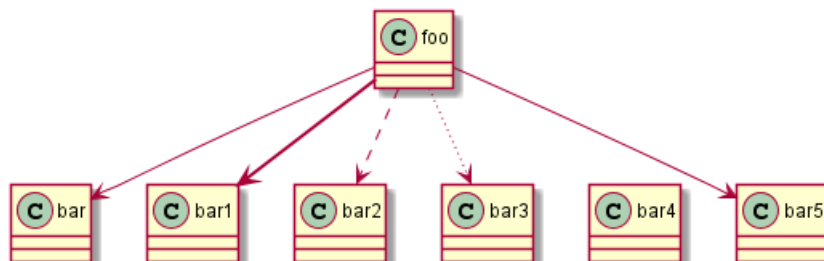


### 3.32 Inline style of relations (Linking or arrow)

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

- without label

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

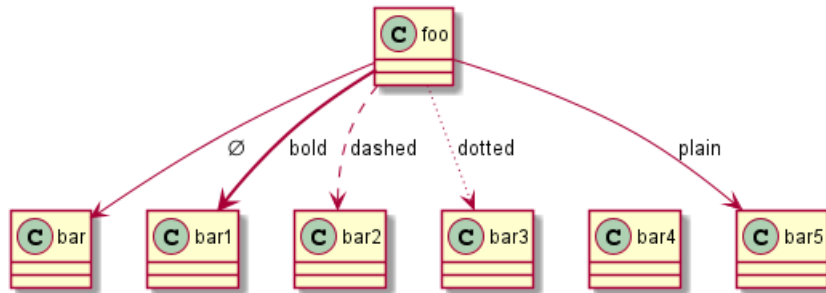


- with label

```

@startuml
class 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]

### 3.33 Change relation, linking or arrow color and style

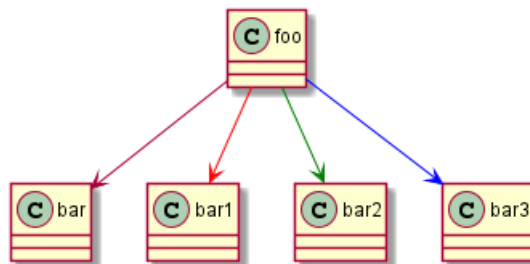
You can change the color of individual relation or arrows using the following notation: [#color] or #color;line. [bold|dashed|dotted|dashed]

- old method

```

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

```



- new method

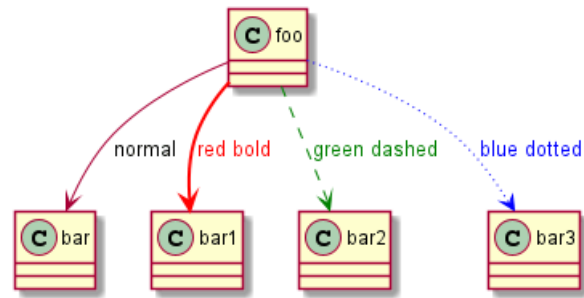
```

@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]

### 3.34 Arrows from/to class members

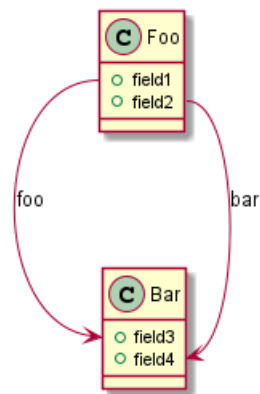
```

@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
}

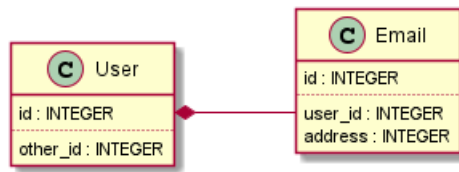
class Email {
id : INTEGER
..
user_id : INTEGER
address : INTEGER
}

```



```
}
```

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



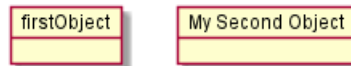
[Ref. QA-5261]

## 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.

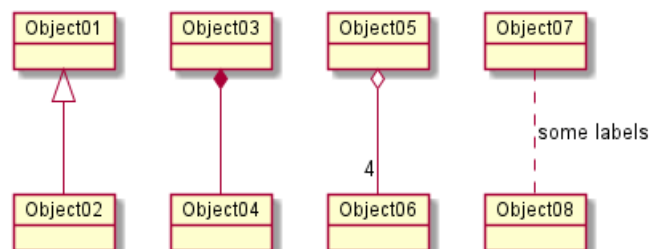
Knowing those rules, it is possible to draw the following drawings.

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 "" 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.

```

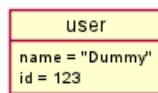
@startuml

object user

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

@enduml

```



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

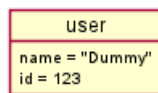
```

@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 **Contry => CapitalCity**" as CC {
  UK => London
  USA => Washington
  Germany => Berlin
}
@enduml
```

Map Contry => 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
```





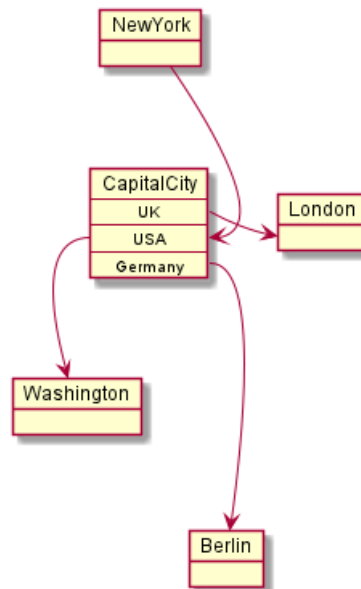
```

@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
```

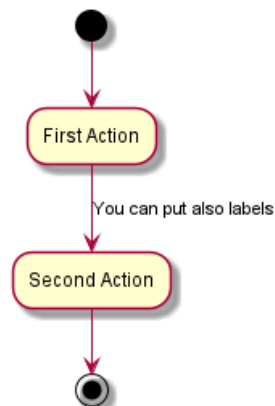


### 5.2 Label on arrows

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
```



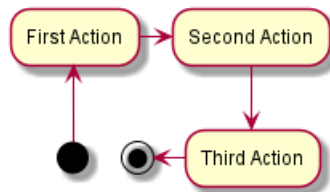
### 5.3 Changing arrow direction

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
```

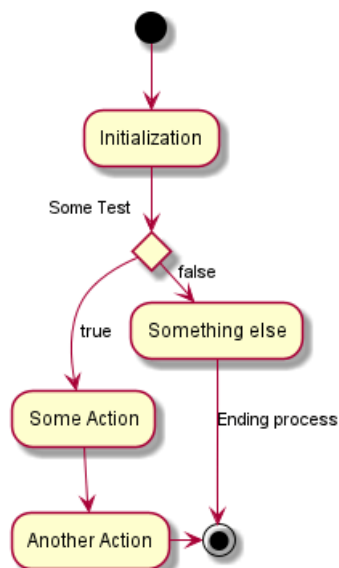


## 5.4 Branches

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

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

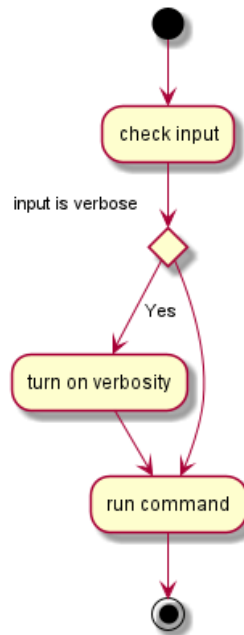
if "Some Test" then
  -->[true] "Some Action"
  --> "Another Action"
-right-> (*)
else
  ->[false] "Something else"
  -->[Ending process] (*)
endif
@enduml
```





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
endif
else
```

```

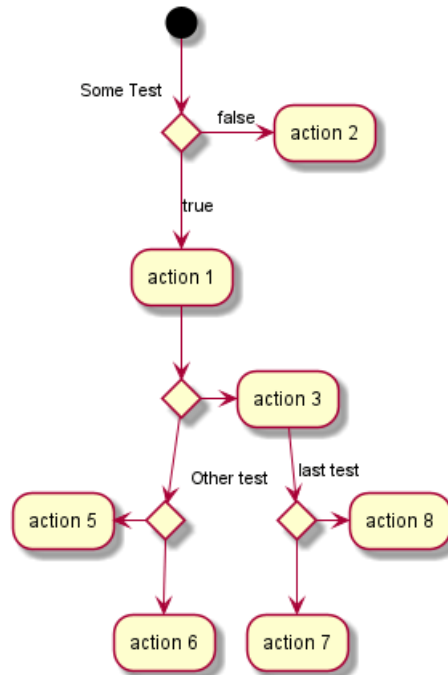
->[false] "action 2"

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===

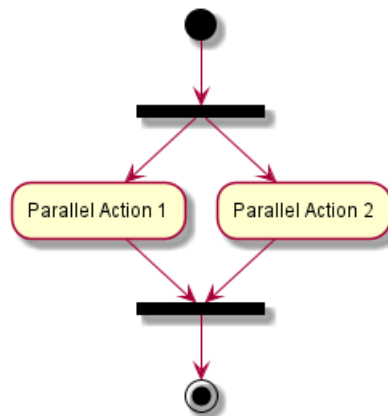
===B1=== --> "Parallel Action 2"
--> ===B2===

--> (*)

@enduml

```





### 5.6.1 # Long action description

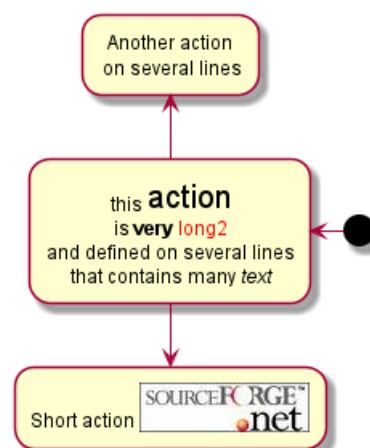
When you declare activities, you can span on several lines the description text. You can also add in the description. You can also give a short code to the activity with the `as` keyword. This code can be used later in the diagram description.

```

@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
  
```



## 5.7 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"
  
```

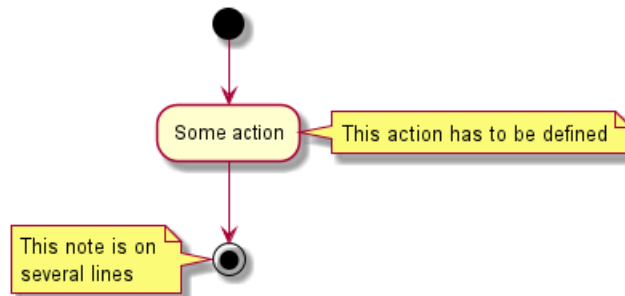


```

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

@enduml

```



## 5.8 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
}

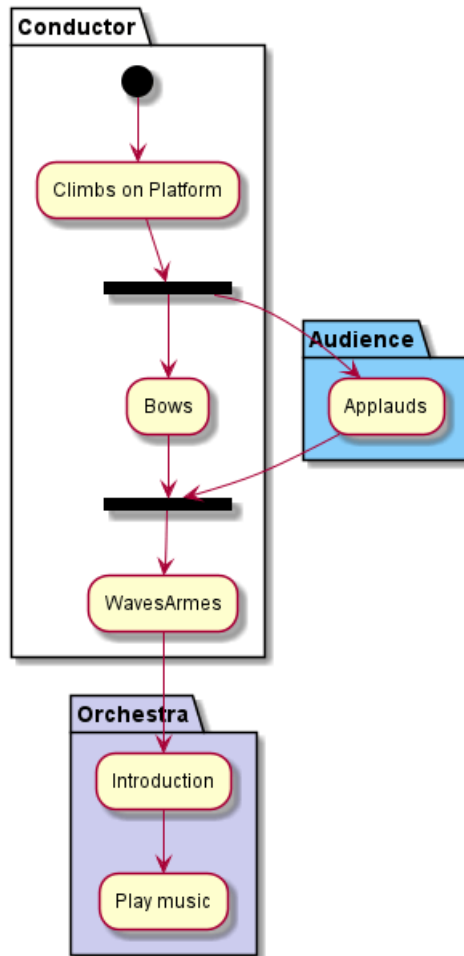
partition Audience #LightSkyBlue {
  === S1 === --> Applauds
}

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

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

@enduml

```



## 5.9 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
```

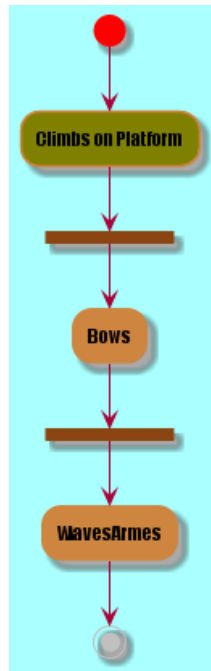
```
skinparam backgroundColor #AFFFFFFF
skinparam activity {
  StartColor red
  BarColor SaddleBrown
  EndColor Silver
  BackgroundColor Peru
  BackgroundColor<< Begin >> Olive
  BorderColor Peru
  FontName Impact
}
```

```
(* ) --> "Climbs on Platform" << Begin >>
--> === S1 ===
--> Bows
```



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

@enduml
```



## 5.10 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
```



## 5.11 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

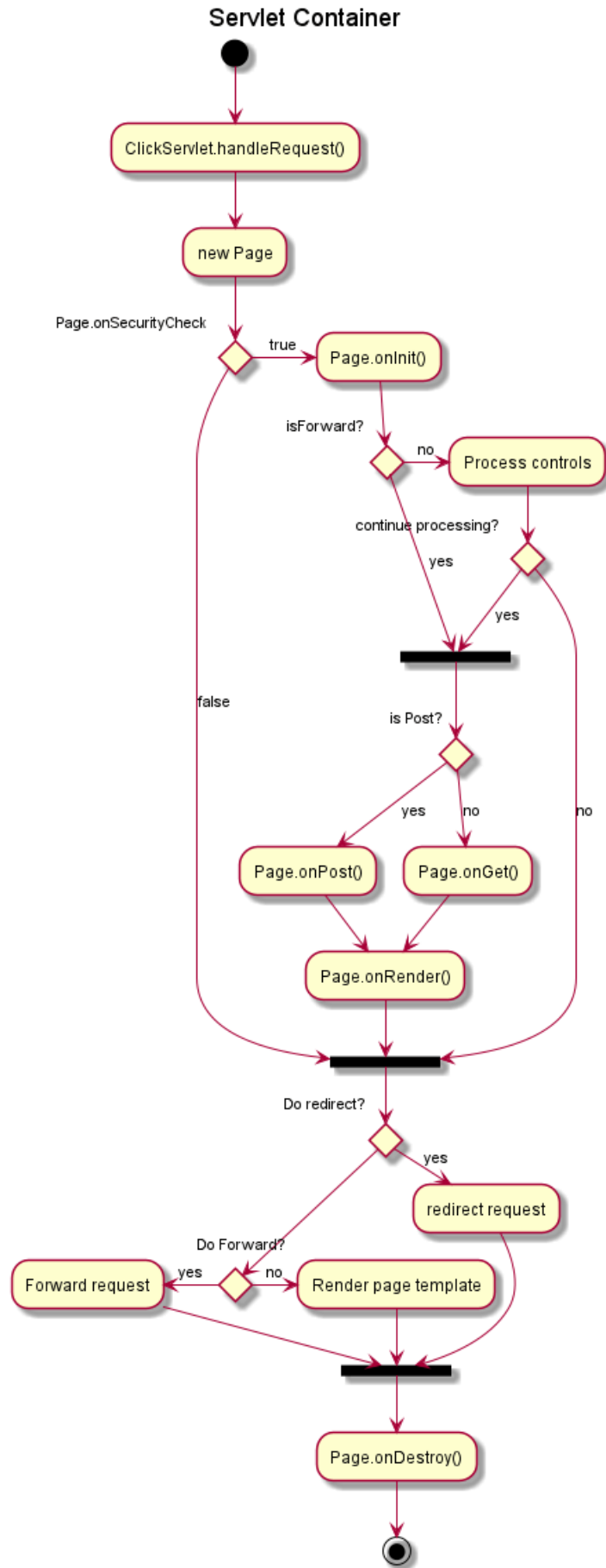
  if "is Post?" then
    -->[yes] "Page.onPost()"
    --> "Page.onRender()" as render
    --> ===REDIRECT_CHECK===
  else
    -->[no] "Page.onGet()"
    --> render
  endif

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.

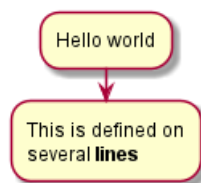
### 6.1 Simple action

Activities label starts with : and ends with ;.

Text formatting can be done using creole wiki syntax.

They are implicitly linked in their definition order.

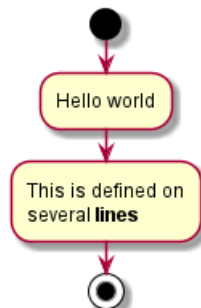
```
@startuml
:Hello world;
:This is defined on
several lines;
@enduml
```



### 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.

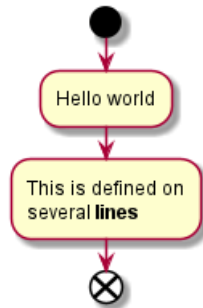
```
@startuml
start
:Hello world;
```



```

:This is defined on
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

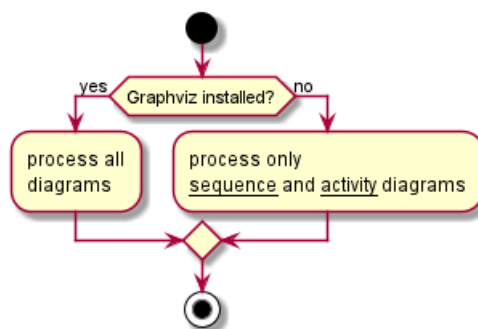
start

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

stop

@enduml

```



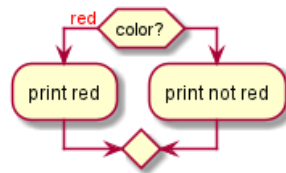
- `if (...) is (...) then`

```

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

```

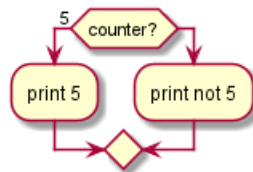




- if (...) equals (...) then

```

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



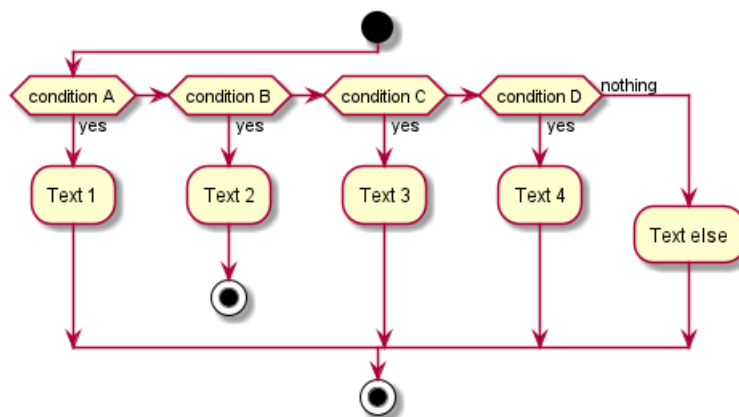
[Ref. QA-301]

### 6.3.1 Several tests (horizontal mode)

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

```

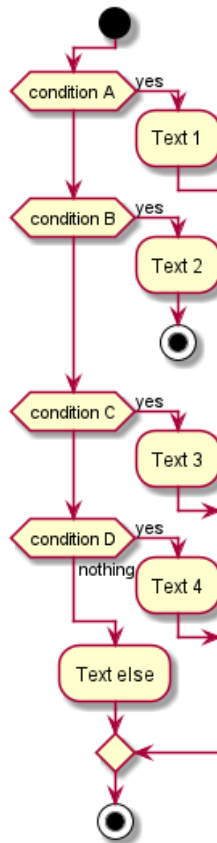
@startuml
start
if (condition A) then (yes)
:Text 1;
elseif (condition B) then (yes)
:Text 2;
stop
elseif (condition C) then (yes)
:Text 3;
elseif (condition D) then (yes)
:Text 4;
else (nothing)
:Text else;
endif
stop
@enduml
  
```



### 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.

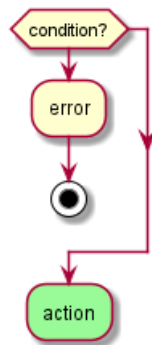
```
@startuml
if (condition?) then
  :error;
```



```

stop
endif
#palegreen:action;
@enduml

```



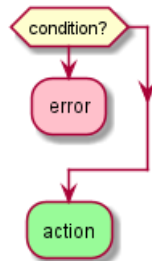
But if you want to stop at an precise action, you can use the kill or detach keyword:

- kill

```

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

```



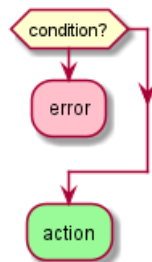
[Ref. QA-265]

- detach

```

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

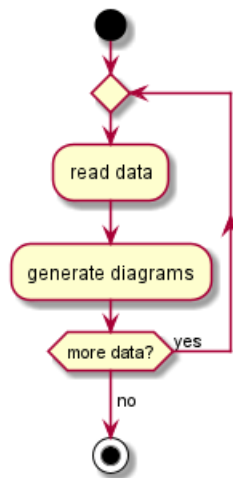
```



## 6.5 Repeat loop

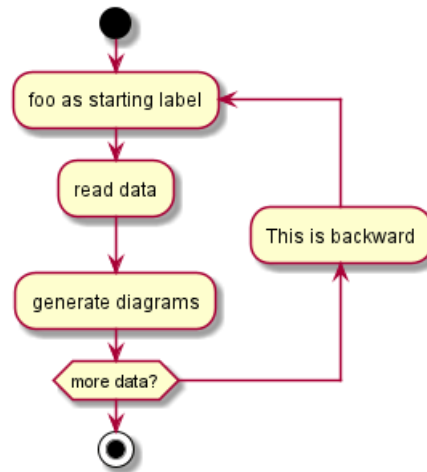
You can use `repeat` and `repeatwhile` keywords to have repeat loops.

```
@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
```

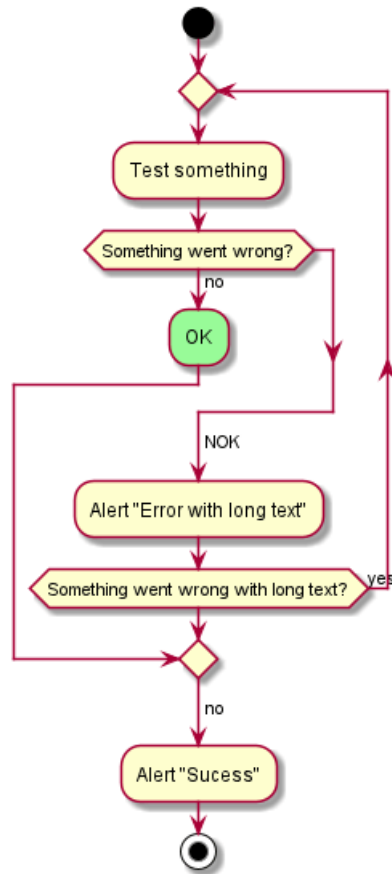


## 6.6 Break on a repeat loop [break]

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)
->no;
:Alert "Sucess";
stop
@enduml
  
```



[Ref. QA-6105]

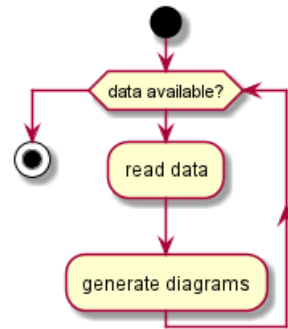
## 6.7 While loop

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

```

@startuml
start
while (data available?)
  :read data;
  :generate diagrams;
endwhile
stop
@enduml
  
```

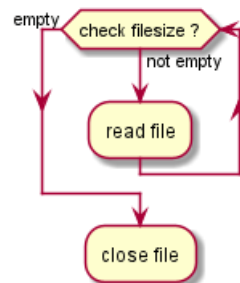




It is possible to provide a label after the endwhile keyword, or using the is keyword.

```

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



## 6.8 Parallel processing

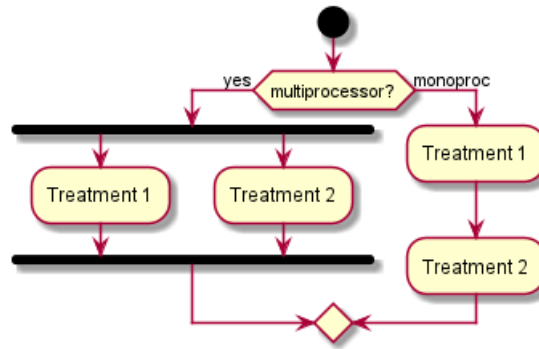
You can use fork, fork again and end fork keywords to denote parallel processing.

```

@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 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>
```

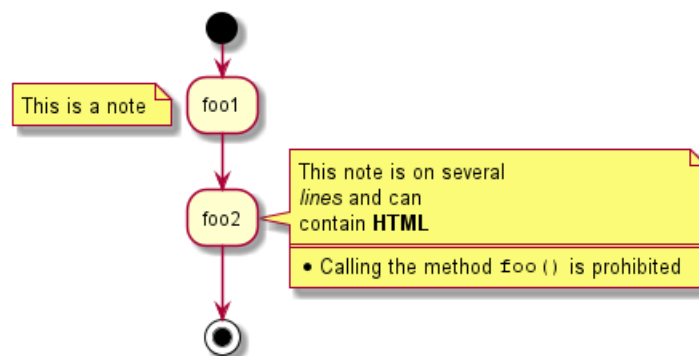
```
  ====
```

```
  * 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;
```

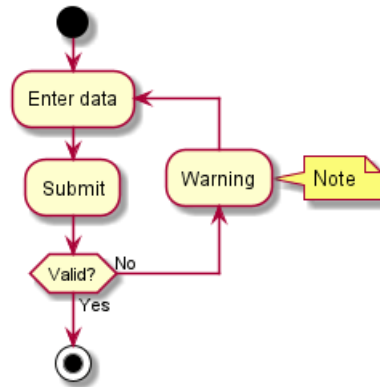
```
note right: Note
```

```
repeat while (Valid?) is (No) not (Yes)
```

```
stop
```

```
@enduml
```





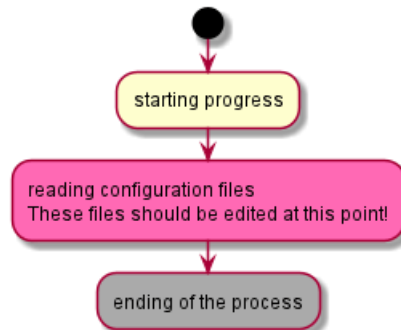
[Ref. QA-11788]

## 6.10 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
  
```

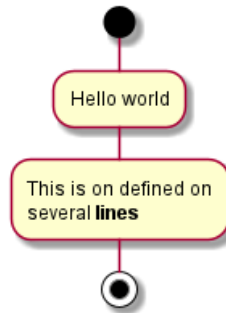


## 6.11 Lines without arrows

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

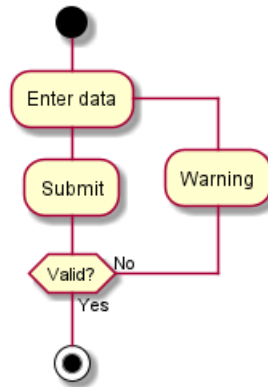
```

@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
  
```



## 6.12 Arrows

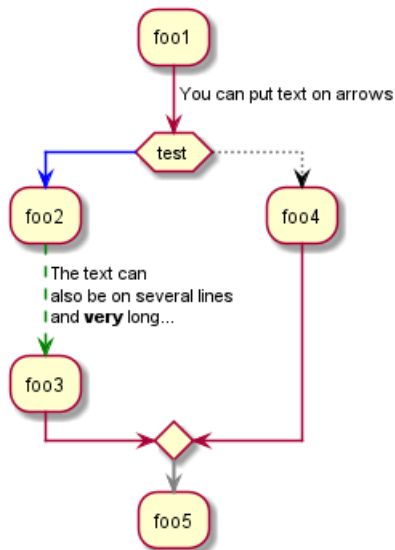
Using the `->` notation, you can add texts to arrow, and change their color.

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
  
```

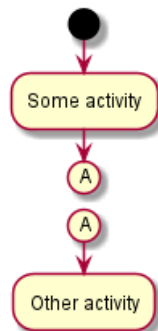




### 6.13 Connector

You can use parentheses to denote connector.

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



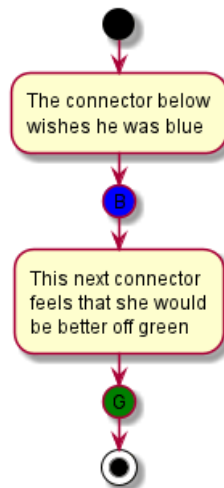
### 6.14 Color on connector

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
```

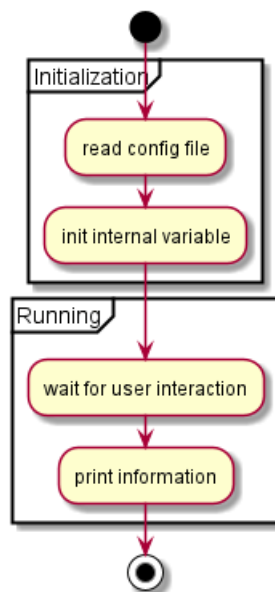


[Ref. QA-10077]

## 6.15 Grouping

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
```

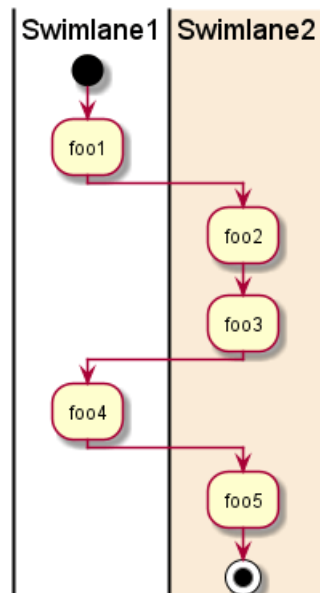


## 6.16 Swimlanes

Using pipe |, you can define swimlanes.

It's also possible to change swimlanes color.

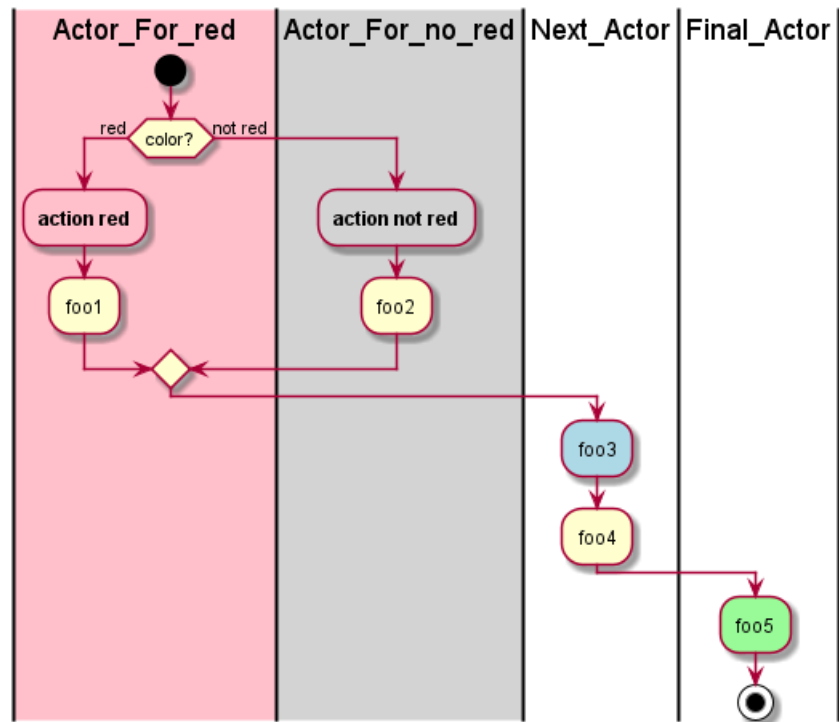
```
@startuml
|Swimlane1|
start
:foo1;
|#AntiqueWhite|Swimlane2|
:foo2;
:foo3;
|Swimlane1|
:foo4;
|Swimlane2|
:foo5;
stop
@enduml
```



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
```





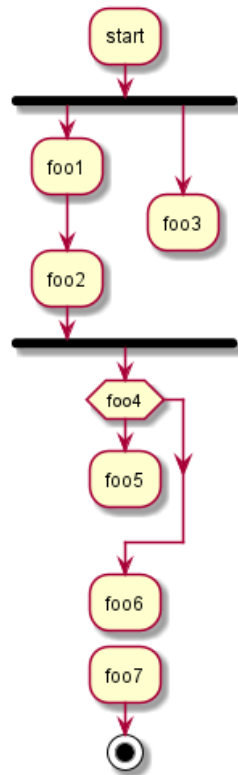
## 6.17 Detach or kill [detach, kill]

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
```



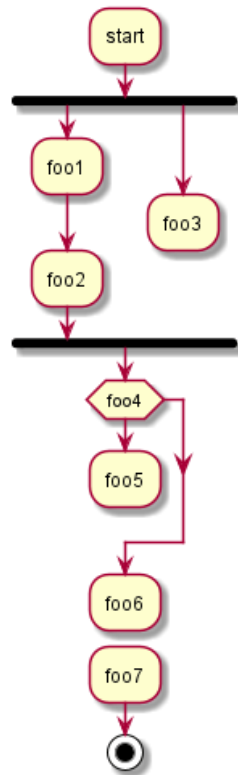


- kill

```

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

```



## 6.18 SDL (Specification and Description Language)

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

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

```

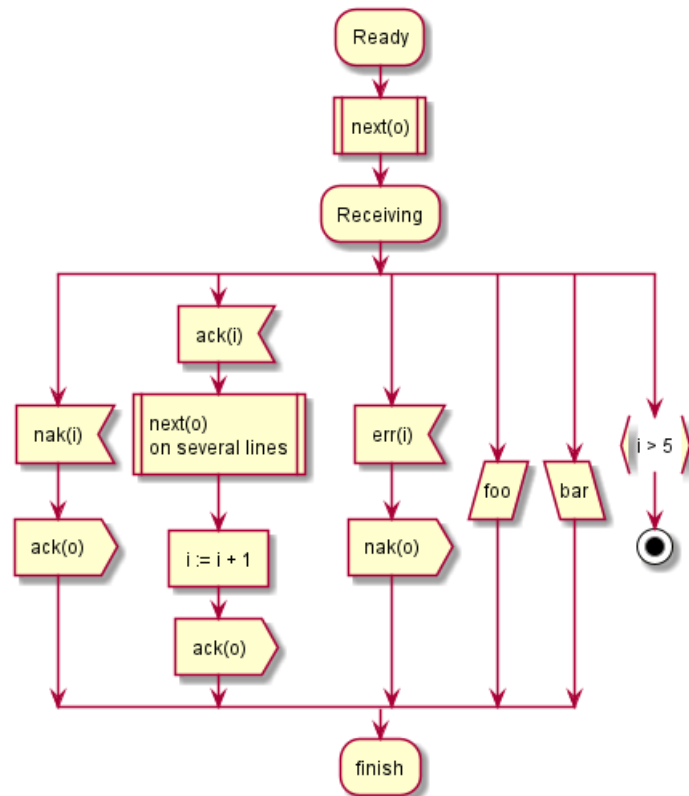
@startuml
:Ready;
:next(o)|
:Receiving;
split
:nak(i)<
:ack(o)>
split again
:ack(i)<
:next(o)
on several lines|
:i := i + 1]
:ack(o)>
split again
:err(i)<
:nak(o)>
split again
:foo/
  
```



```

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

```



## 6.19 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
endif

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

```

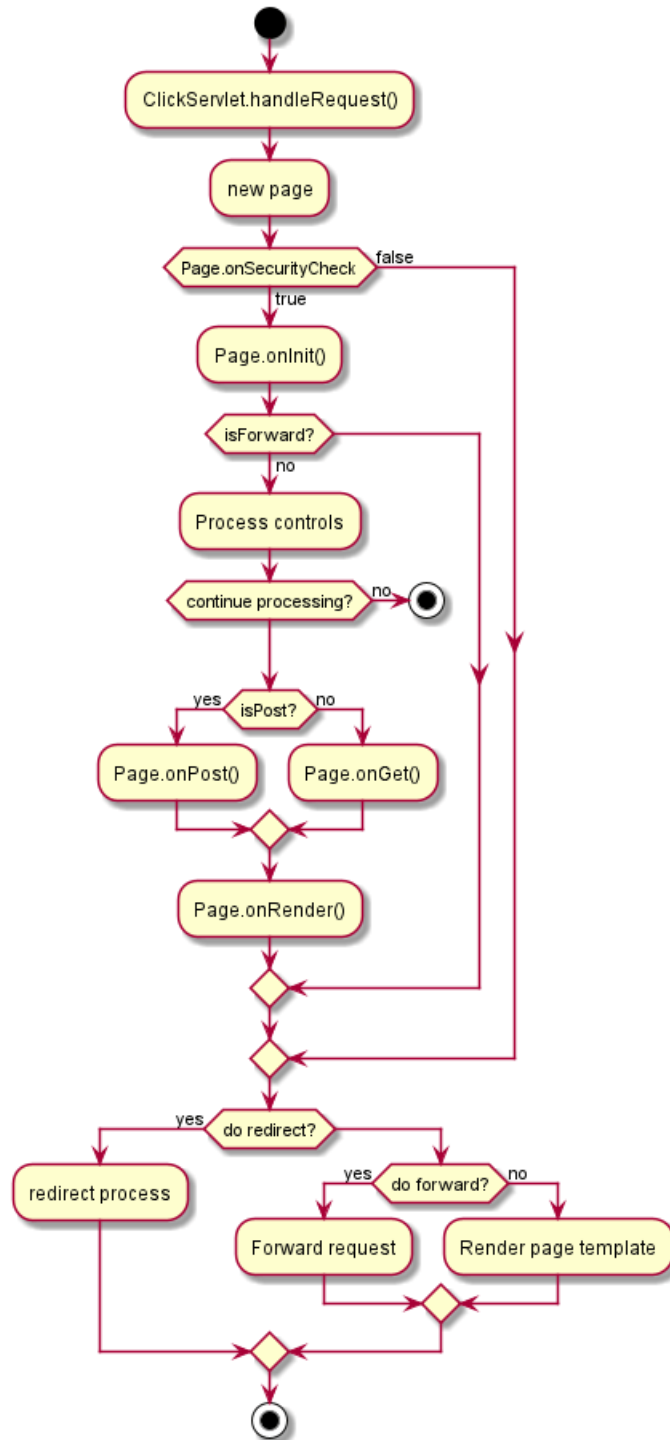


```
else (false)
endif

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

stop

@enduml
```



## 6.20 Condition Style

### 6.20.1 Inside style (by default)

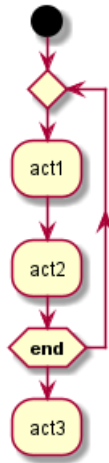
```

@startuml
skinparam conditionStyle inside
start
repeat
:act1;
:act2;

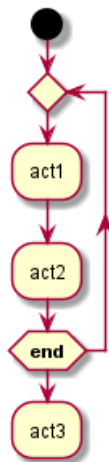
```



```
repeatwhile (<b>end)
:act3;
@enduml
```



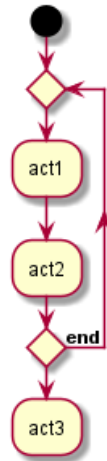
```
@startuml
start
repeat
:act1;
:act2;
repeatwhile (<b>end)
:act3;
@enduml
```



### 6.20.2 Diamond style

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

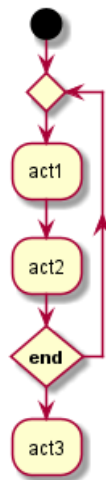




### 6.20.3 InsideDiamond (or *FooI*) style

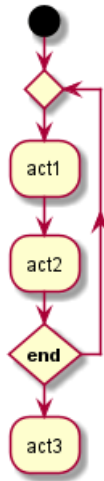
```

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



```

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



[Ref. QA-1290 and #400]

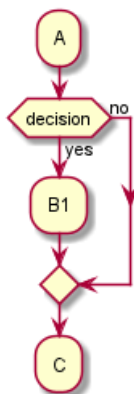
## 6.21 Condition End Style

### 6.21.1 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)

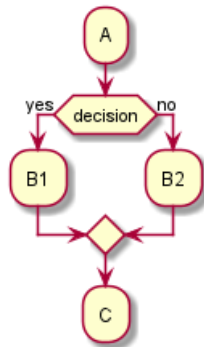
```

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





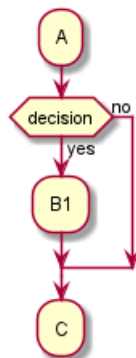
```
@enduml
@enduml
```



### 6.21.2 Horizontal line (hline) style

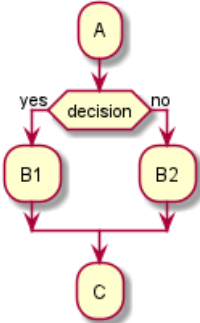
- With one branch

```
@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

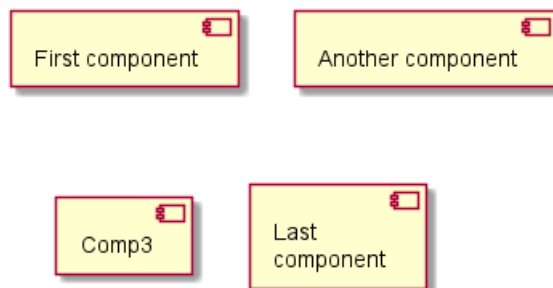
Let's have few examples.

### 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

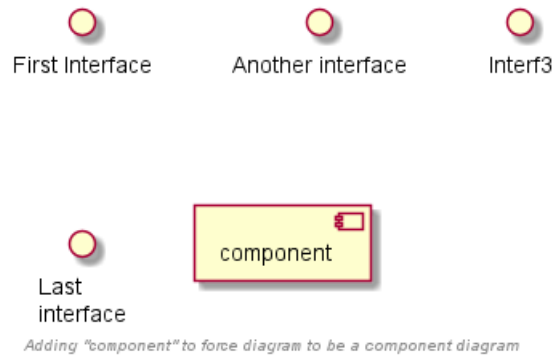
Interface can be defined using the () symbol (because this looks like a circle).

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
```



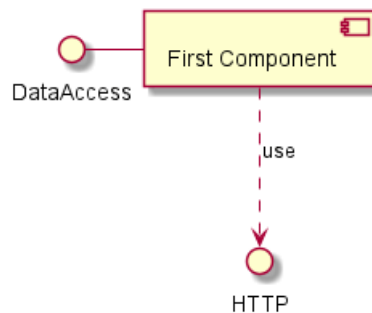
### 7.3 Basic example

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
```

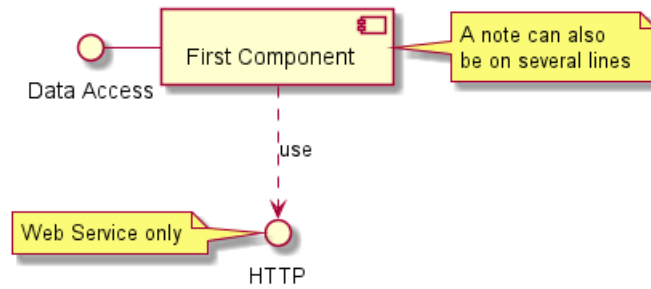
```
DA - [First Component]
[First Component] ..> HTTP : use
```

```
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
}
```

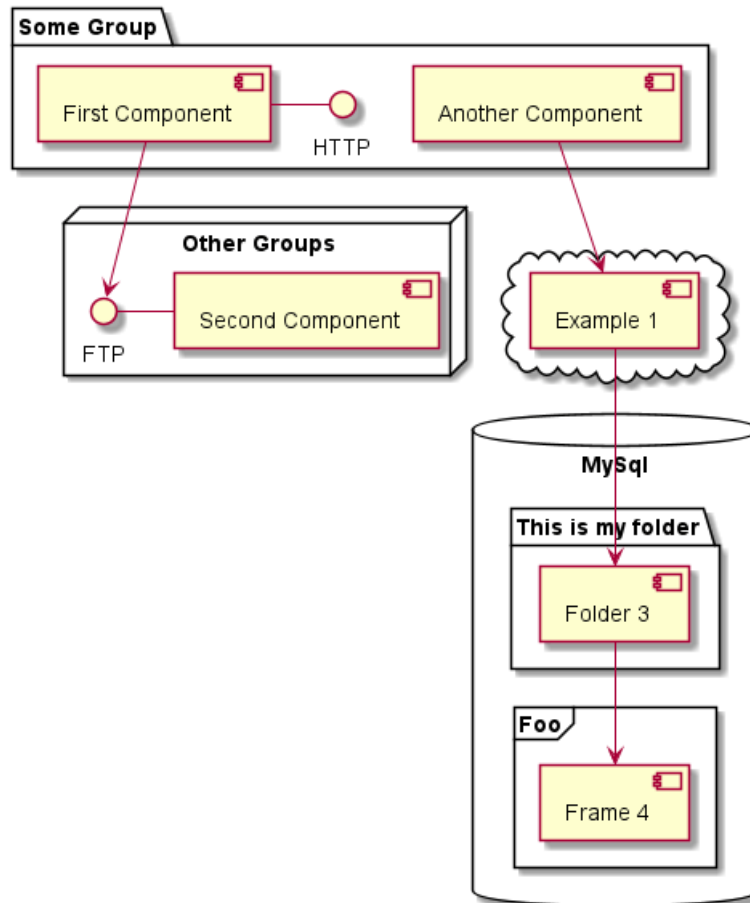
```
cloud {
  [Example 1]
}
```

```
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
```

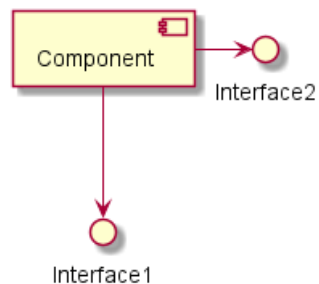




## 7.6 Changing arrows direction

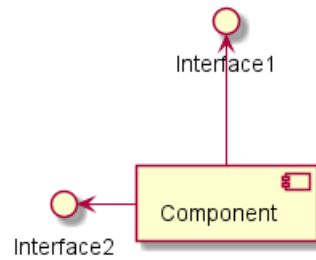
By default, links between classes have two dashes -- and are vertically oriented. It is possible to use horizontal link by putting a single dash (or dot) like this:

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



You can also change directions by reversing the link:

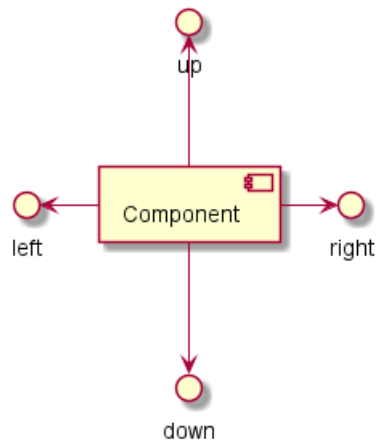
```
@startuml
Interface1 <-- [Component]
Interface2 <- [Component]
@enduml
```



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
  
```



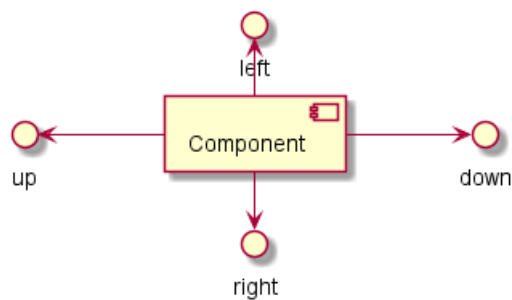
You can shorten the arrow by using only the first character of the direction (for example, `-d-` instead of `-down-`) or the two first characters (`-do-`).

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

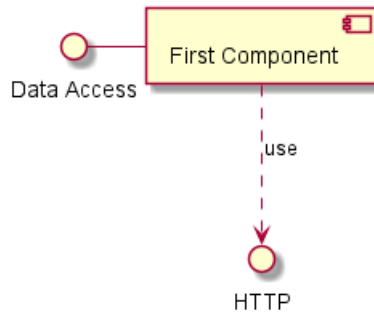
By default (from v1.2020.13-14), UML2 notation is used.



```

@startuml
interface "Data Access" as DA
DA - [First Component]
[First Component] ..> HTTP : use
@enduml

```



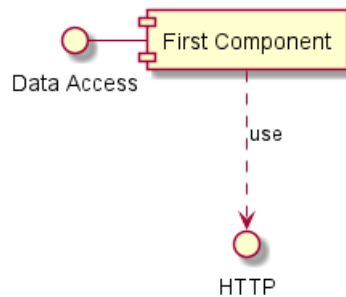
## 7.8 Use UML1 notation

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*).

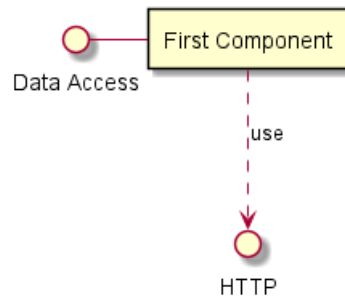
```

@startuml
skinparam componentStyle rectangle
interface "Data Access" as DA
DA - [First Component]
[First Component] ..> HTTP : use
@enduml

```



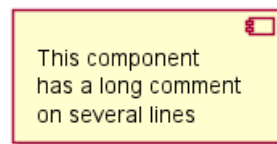




## 7.10 Long description

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] {
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFOFFFFF
FFFFFFFFFOFFFFF
FF0000000000FFF
FF0000000000FFF
FF0000000000FFF
FFFFFFFFFOFFFFF
FFFFFFFFFOFFFFF
FFFFFFFFFFFFFFFF
}
```



```

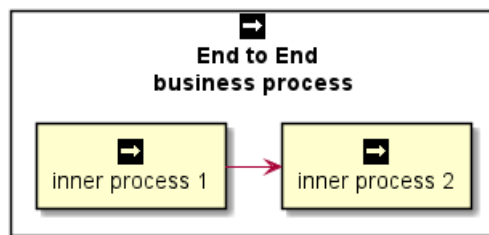
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
}

```

```

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
```

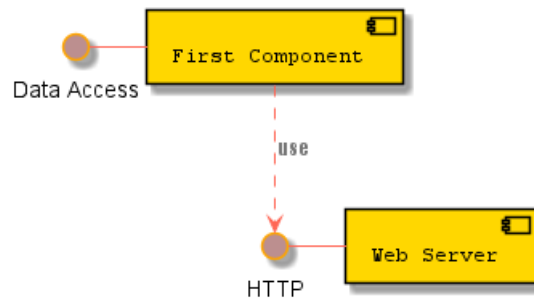
```
DA - [First Component]
```

```
[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>>
```

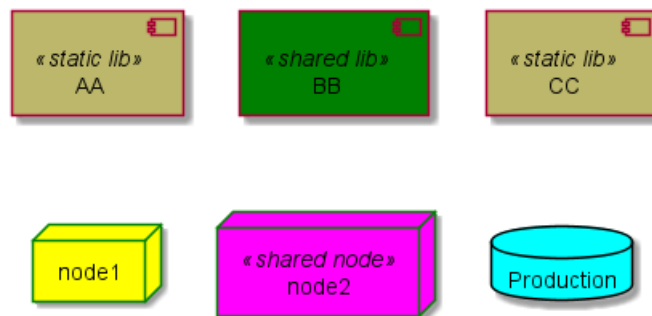
```
database Production
```

```
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" {
```

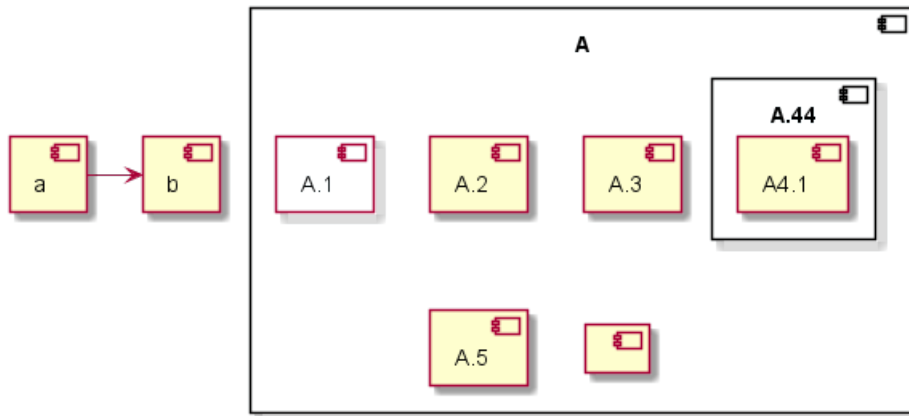
```
}
```

```
    component A.44 {
```

```

    [A4.1]
}
component "A.2"
  [A.3]
  component A.5 [
A.5]
    component A.6 [
]
}
[a]->[b]
@enduml

```

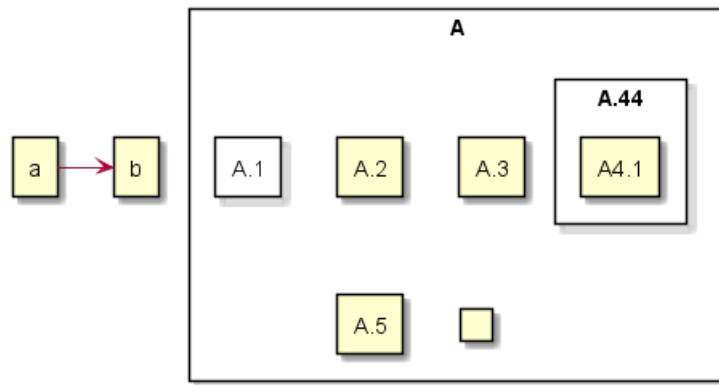


- If you want to suppress it, and to have only the rectangle, you can use `skinparam componentStyle rectangle`

```

@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]

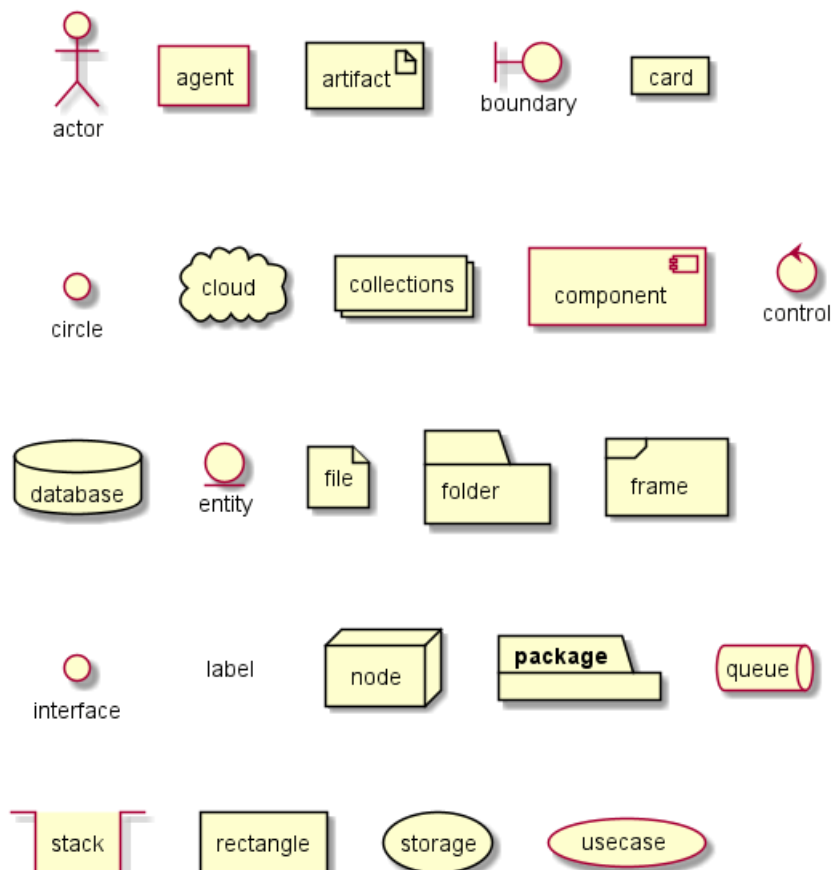
## 8 Deployment Diagram

### 8.1 Declaring element

```

@startuml
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
interface interface
label label
node node
package package
queue queue
stack stack
rectangle rectangle
storage storage
usecase usecase
@enduml

```



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

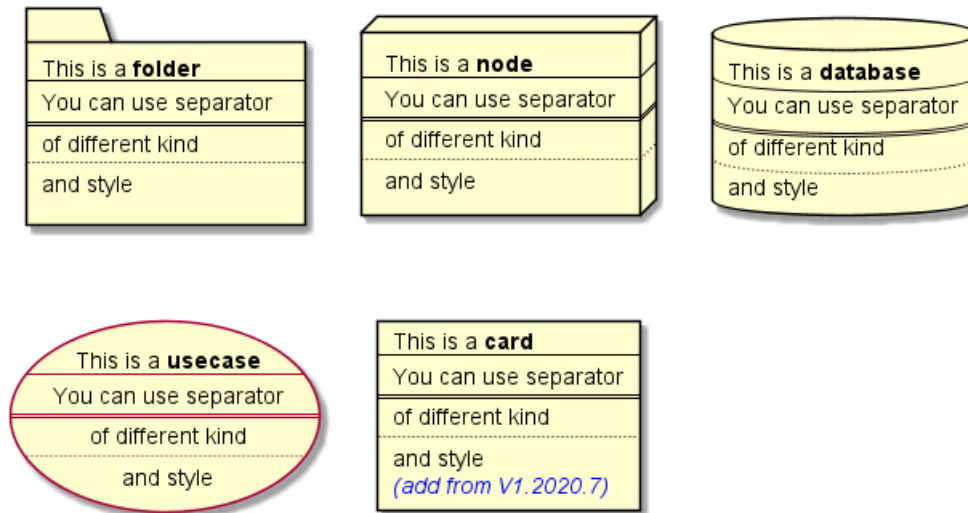
```
@startuml
folder folder [
This is a <b>folder
----
You can use separator
====
of different kind
....
and style
]

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.

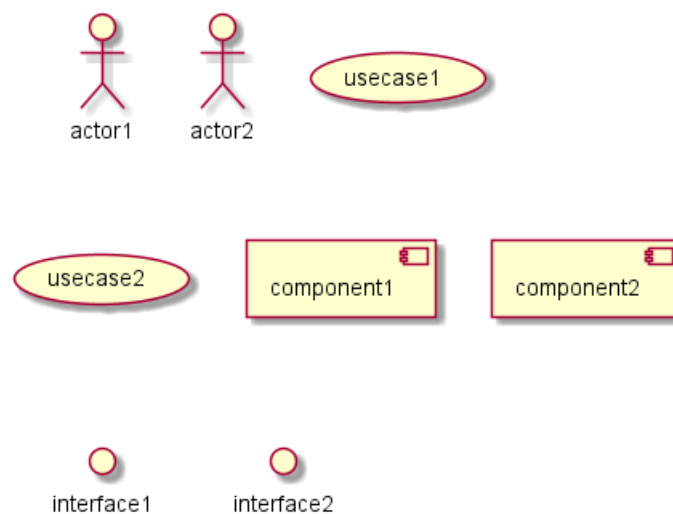
Long form Keyword	Short form Keyword	Long form example	Short form example	Ref.
actor	: <i>a</i> :	actor actor1	:actor2:	Actors
usecase	( <i>u</i> )	usecase usecase1	(usecase2)	Usecases
component	[ <i>c</i> ]	component component1	[component2]	Components
interface	( ) <i>i</i>	interface interface1	( )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

```
@startuml
actor actor1
:actor2:

usecase usecase1
(usecase2)

component component1
[component2]

interface interface1
() "interface2"
@enduml
```



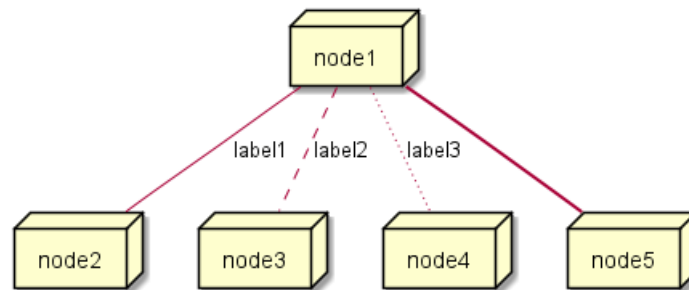


**NB:** There is an old syntax for actor with guillemet which is now deprecated and will be removed some days. Please do not use in your diagram.

### 8.3 Linking or arrow

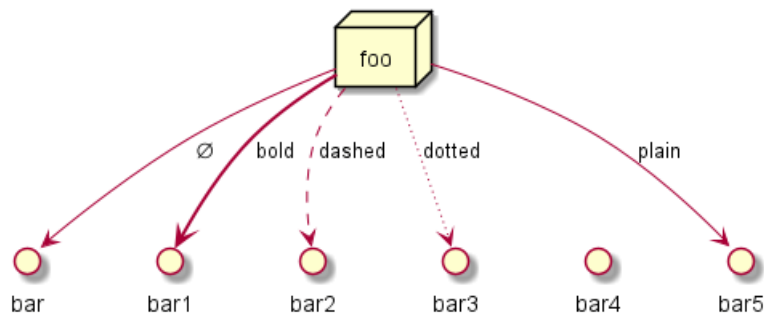
You can create simple links between elements with or without labels:

```
@startuml
node node1
node node2
node node3
node node4
node node5
node1 -- node2 : label1
node1 .. node3 : label2
node1 ~- node4 : label3
node1 == node5
@enduml
```



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

```
@startuml
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
```



It is possible to use several types of links:

```
@startuml
artifact artifact1
artifact artifact2
artifact artifact3
```

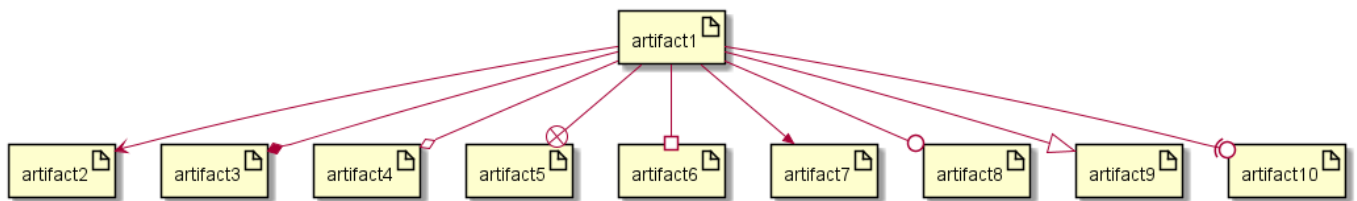


```

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:

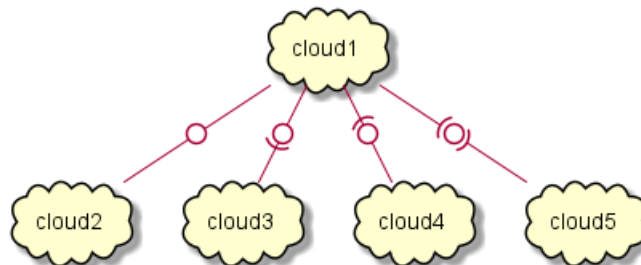
```

@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

```

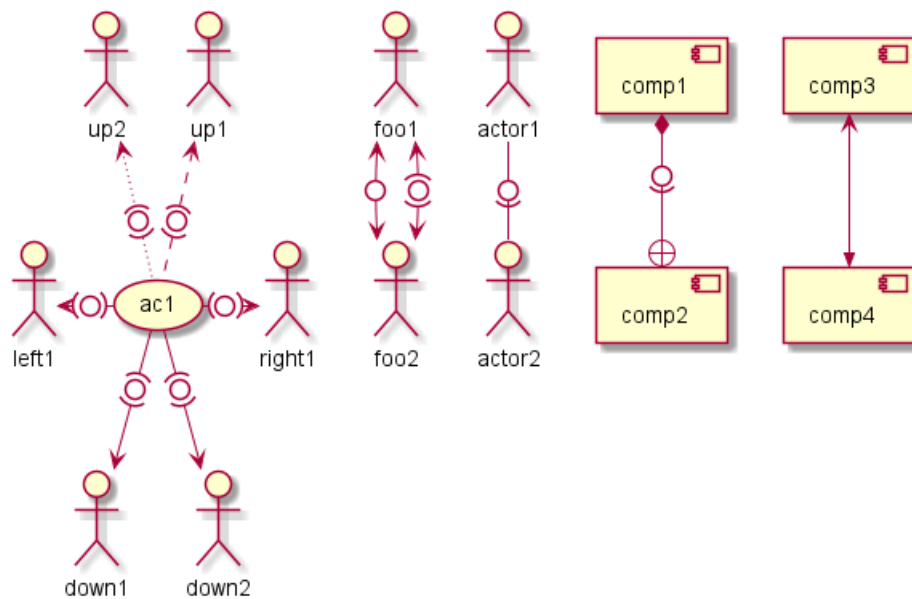
```

(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]

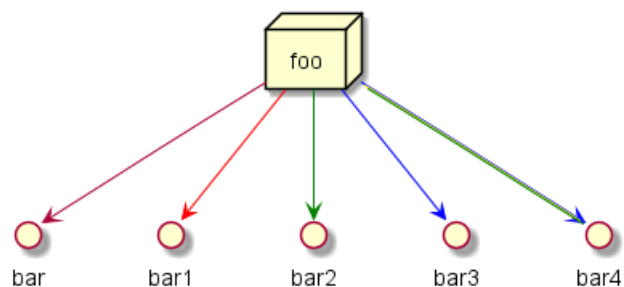
## 8.4 Change arrow color and style

You can change the color of individual arrows using the following notation: [#color].

```

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

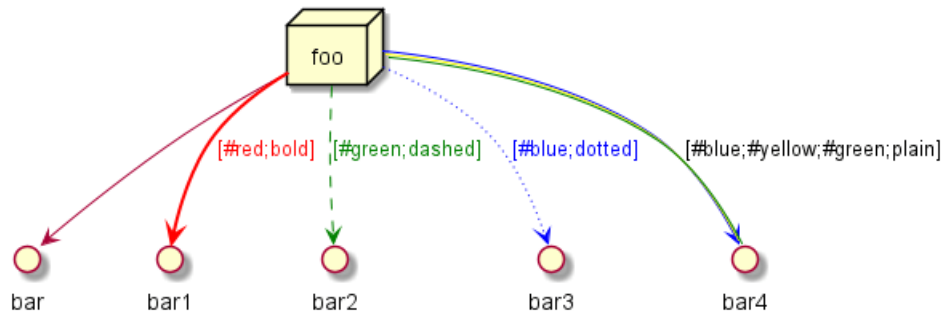
```



Then you can change color and style of individual arrows using the following notation:

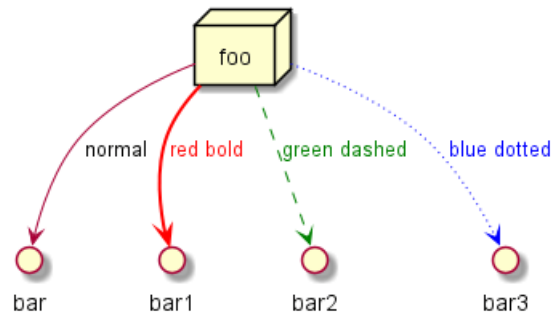
- old method [#color;style]

```
@startuml
node foo
foo --> bar
foo -[#red;bold]-> bar1           : <color:red>[#red;bold]
foo -[#green;dashed]-> bar2       : <color:green>[#green;dashed]
foo -[#blue;dotted]-> bar3        : <color:blue>[#blue;dotted]
foo -[#blue;#yellow;#green;plain]-> bar4 : [#blue;#yellow;#green;plain]
@enduml
```



- new method #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
```



[See similar feature on class diagram]

## 8.5 Nestable elements

Here are the nestable elements:

```
@startuml
artifact artifact {
}
card card {
}
cloud cloud {
}
component component {
}
database database {
}
```



```

file file {
}
folder folder {
}
frame frame {
}
node node {
}
package package {
}
queue queue {
}
rectangle rectangle {
}
stack stack {
}
storage storage {
}
@enduml

```



## 8.6 Packages and nested elements

### 8.6.1 Example with one level

```

@startuml
artifact      artifactVeryL00000000000000000000g    as "artifact" {
file f1
}
card          cardVeryL00000000000000000000g      as "card" {
file f2
}
cloud         cloudVeryL00000000000000000000g    as "cloud" {
file f3
}
component     componentVeryL00000000000000000000g as "component" {
file f4
}
database      databaseVeryL00000000000000000000g as "database" {
file f5
}
file          fileVeryL00000000000000000000g     as "file" {
file f6
}
folder        folderVeryL00000000000000000000g   as "folder" {
file f7
}
frame         frameVeryL00000000000000000000g    as "frame" {
file f8
}
node          nodeVeryL00000000000000000000g     as "node" {
file f9
}
package       packageVeryL00000000000000000000g  as "package" {
file f10
}

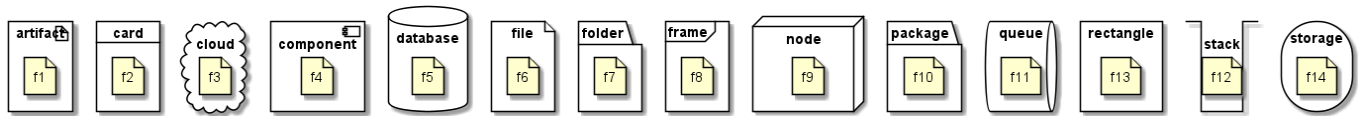
```



```

}
queue      queueVeryL0000000000000000000g    as "queue" {
file f11
}
rectangle  rectangleVeryL0000000000000000000g  as "rectangle" {
file f13
}
stack      stackVeryL00000000000000000000g    as "stack" {
file f12
}
storage    storageVeryL00000000000000000000g  as "storage" {
file f14
}
}
@enduml

```



### 8.6.2 Other example

```

@startuml
artifact Foo1 {
  folder Foo2
}

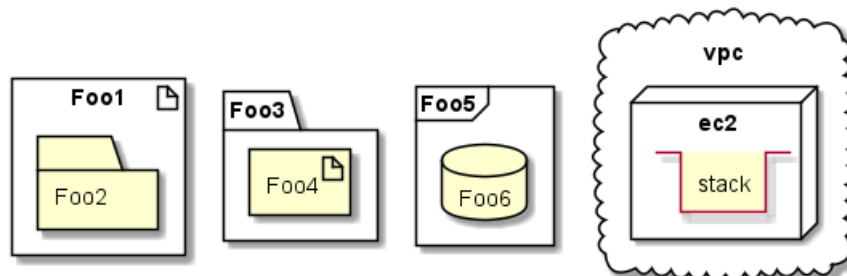
folder Foo3 {
  artifact Foo4
}

frame Foo5 {
  database Foo6
}

cloud vpc {
  node ec2 {
    stack stack
  }
}

@enduml

```



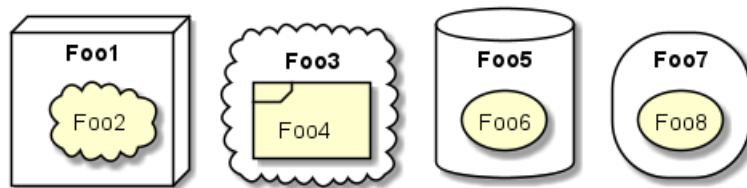
```

@startuml
node Foo1 {
  cloud Foo2
}

```



```
cloud Foo3 {  
    frame Foo4  
}  
  
database Foo5 {  
    storage Foo6  
}  
  
storage Foo7 {  
    storage Foo8  
}  
@enduml
```

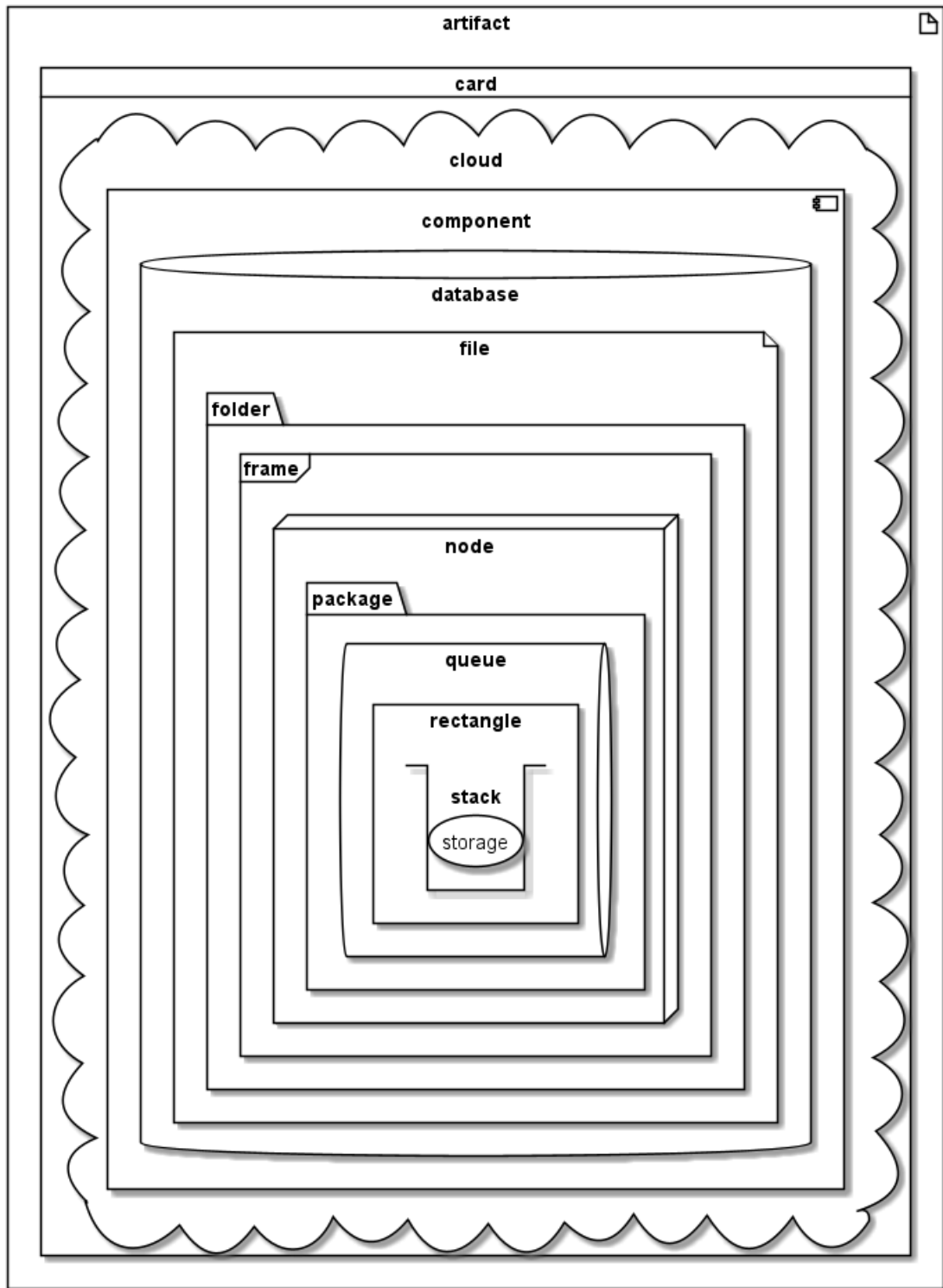


### 8.6.3 Full nesting

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 {  
                                node node {  
                                    package package {  
                                        queue queue {  
                                            rectangle rectangle {  
                                                stack stack {  
                                                    storage storage {  
                                                        }  
                                                    }  
                                                }  
                                            }  
                                        }  
                                    }  
                                }  
                            }  
                        }  
                    }  
                }  
            }  
        }  
    }  
}  
@enduml
```



- or reverse alphabetical order

```

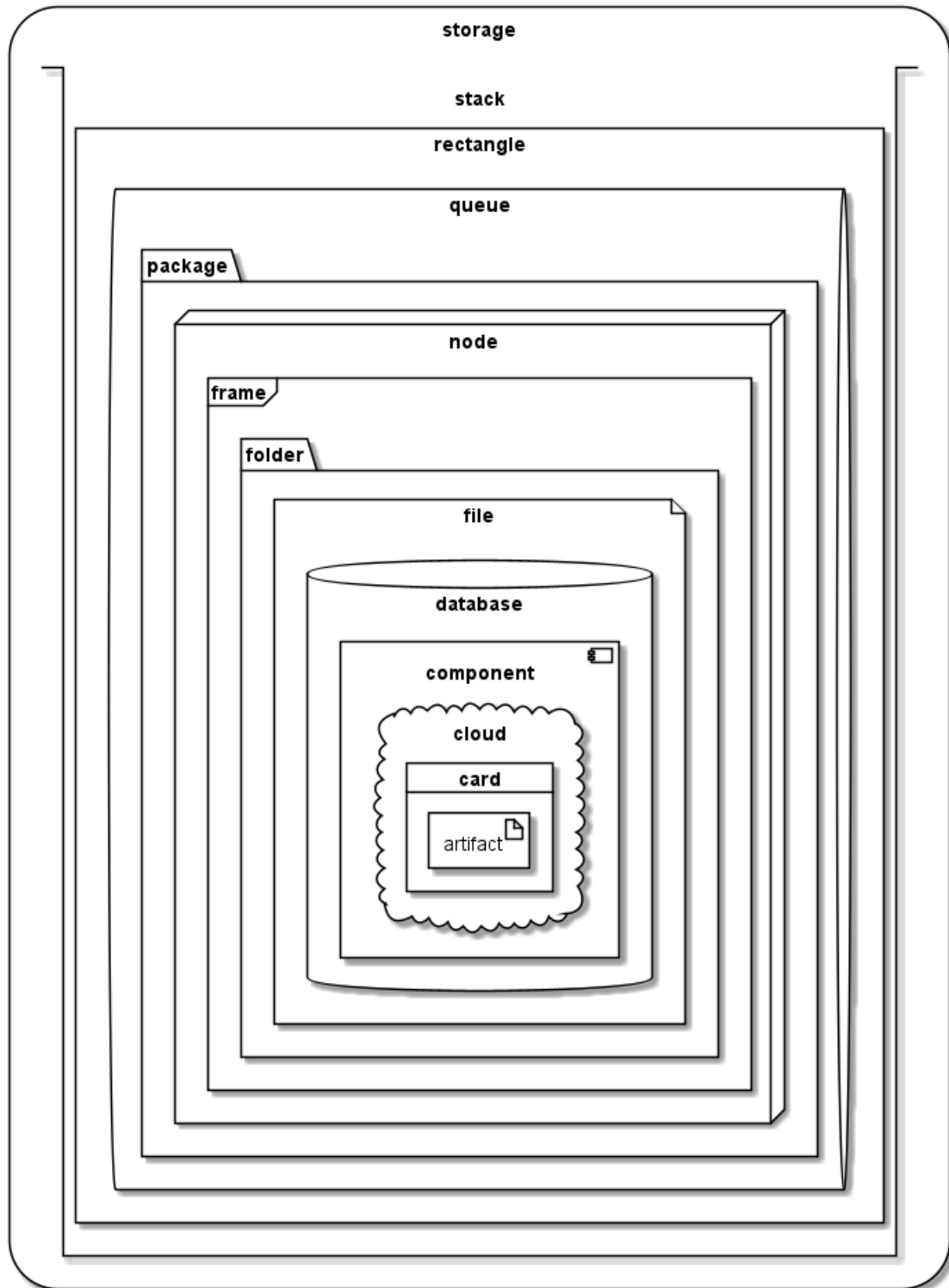
@startuml
storage storage {
stack stack {
rectangle rectangle {
queue queue {
package package {
node node {
frame frame {
folder folder {

```



```
file file {  
  database database {  
    component component {  
      cloud cloud {  
        card card {  
          artifact artifact {  
            }  
          }  
        }  
      }  
    }  
  }  
}
```

@enduml



## 8.7 Round corner

```

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

rectangle "Concept Model" <<Concept>> {
    rectangle "Example 1" <<Concept>> as ex1

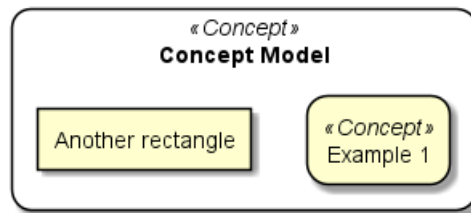
```



```

rectangle "Another rectangle"
}
@enduml

```



## 8.8 Alias

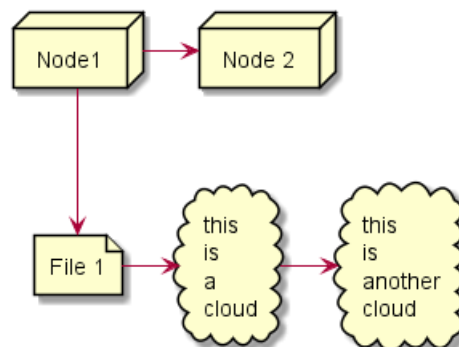
### 8.8.1 Simple alias with as

```

@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

```



### 8.8.2 Examples of long alias

```

@startuml
actor      "actor"      as actorVeryL00000000000000000000g
agent      "agent"      as agentVeryL00000000000000000000g
artifact   "artifact"   as artifactVeryL00000000000000000000g
boundary   "boundary"   as boundaryVeryL00000000000000000000g
card       "card"       as cardVeryL00000000000000000000g
cloud      "cloud"      as cloudVeryL00000000000000000000g
collections "collections" as collectionsVeryL00000000000000000000g
component  "component"  as componentVeryL00000000000000000000g

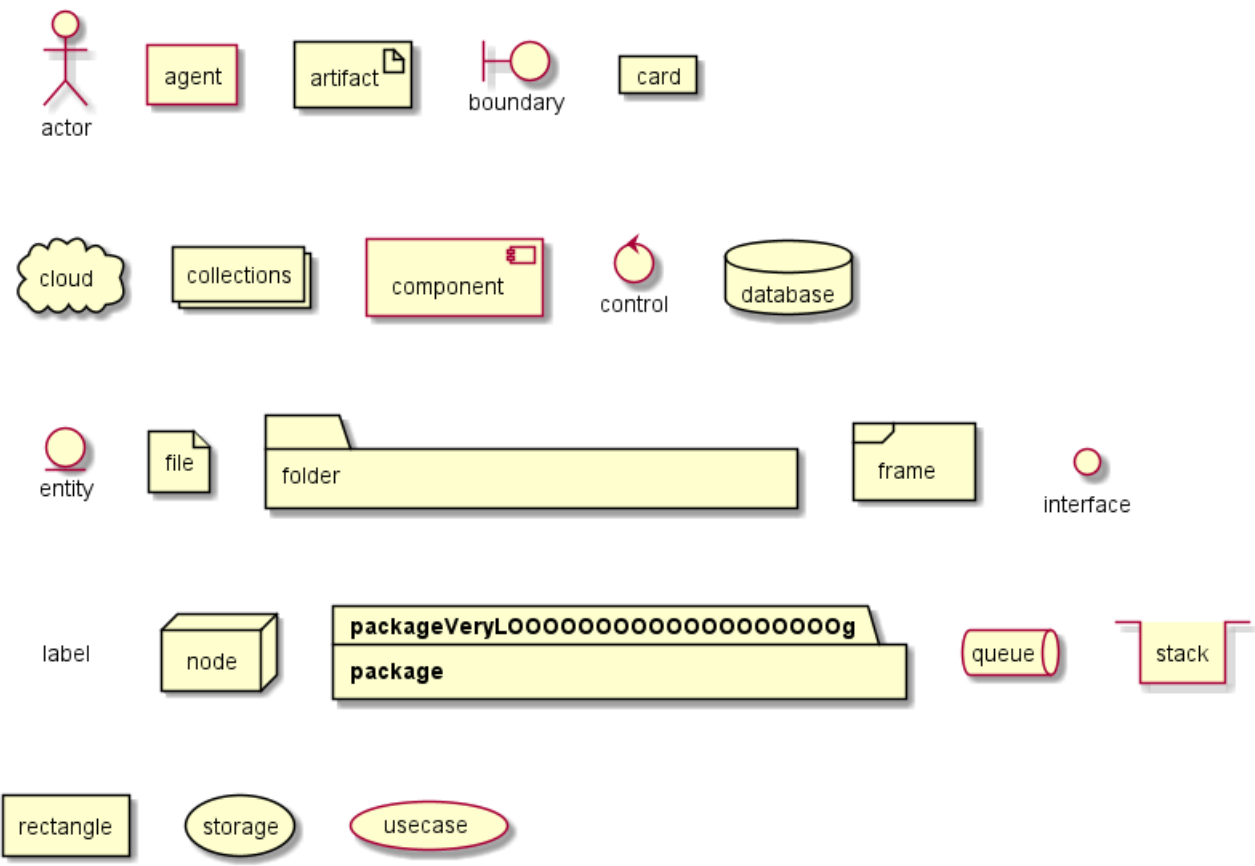
```



```

control      "control"      as controlVeryL00000000000000000000g
database     "database"     as databaseVeryL00000000000000000000g
entity       "entity"       as entityVeryL00000000000000000000g
file         "file"         as fileVeryL00000000000000000000g
folder       "folder"       as folderVeryL00000000000000000000g
frame        "frame"        as frameVeryL00000000000000000000g
interface    "interface"    as interfaceVeryL00000000000000000000g
label        "label"        as labelVeryL00000000000000000000g
node         "node"         as nodeVeryL00000000000000000000g
package      "package"      as packageVeryL00000000000000000000g
queue        "queue"        as queueVeryL00000000000000000000g
stack        "stack"        as stackVeryL00000000000000000000g
rectangle    "rectangle"    as rectangleVeryL00000000000000000000g
storage      "storage"      as storageVeryL00000000000000000000g
usecase      "usecase"      as usecaseVeryL00000000000000000000g
@enduml

```



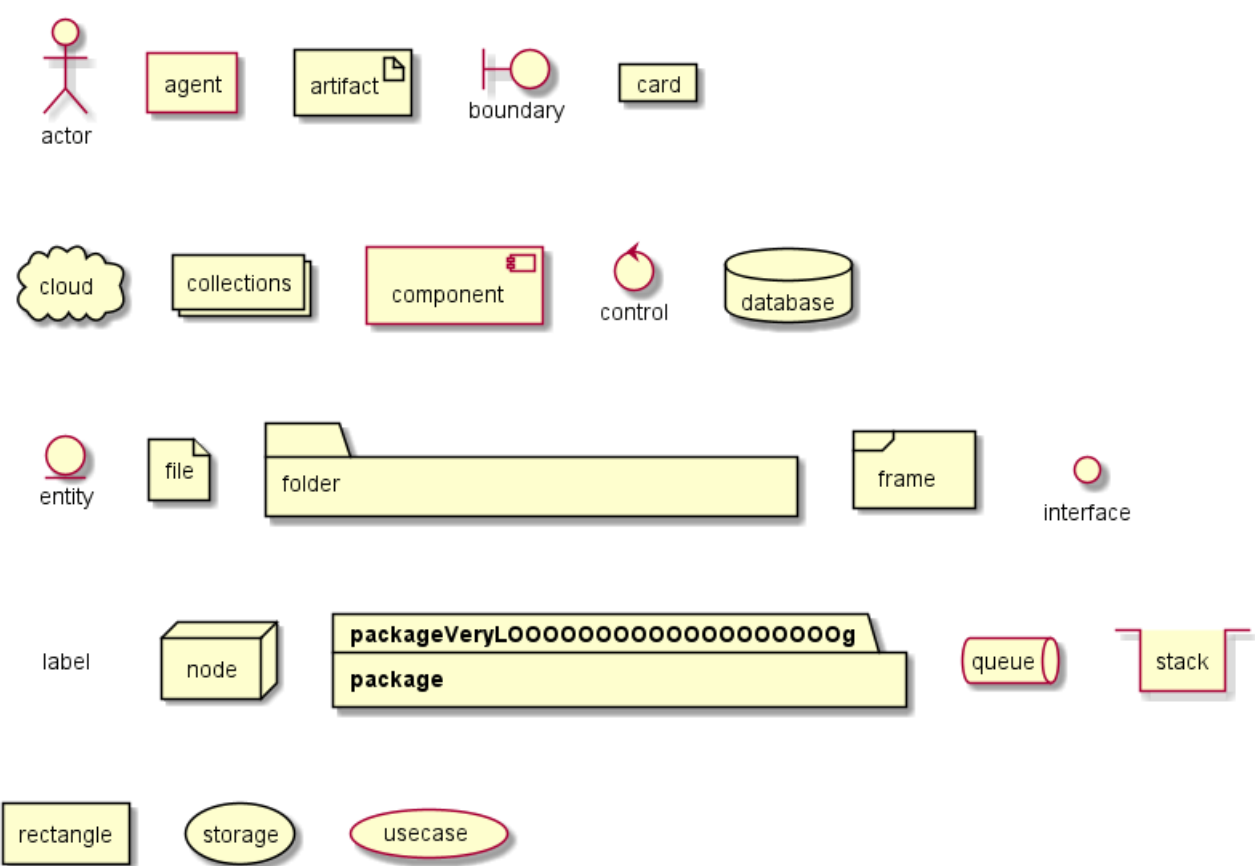
```

@startuml
actor      actorVeryL00000000000000000000g      as "actor"
agent      agentVeryL00000000000000000000g      as "agent"
artifact   artifactVeryL00000000000000000000g  as "artifact"
boundary   boundaryVeryL00000000000000000000g  as "boundary"
card       cardVeryL00000000000000000000g      as "card"
cloud      cloudVeryL00000000000000000000g     as "cloud"
collections collectionsVeryL00000000000000000000g as "collections"
component  componentVeryL00000000000000000000g as "component"
control    controlVeryL00000000000000000000g   as "control"
database   databaseVeryL00000000000000000000g  as "database"
entity     entityVeryL00000000000000000000g    as "entity"
file       fileVeryL00000000000000000000g      as "file"

```

```

folder      folderVeryL00000000000000000000g      as "folder"
frame       frameVeryL00000000000000000000g      as "frame"
interface   interfaceVeryL00000000000000000000g    as "interface"
label       labelVeryL00000000000000000000g      as "label"
node        nodeVeryL00000000000000000000g      as "node"
package     packageVeryL00000000000000000000g    as "package"
queue       queueVeryL00000000000000000000g      as "queue"
stack       stackVeryL00000000000000000000g     as "stack"
rectangle   rectangleVeryL00000000000000000000g  as "rectangle"
storage     storageVeryL00000000000000000000g    as "storage"
usecase     usecaseVeryL00000000000000000000g    as "usecase"
@enduml
    
```



[Ref. QA-12082]

## 8.9 Type of arrow head or '0' arrow

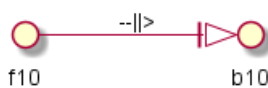
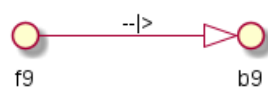
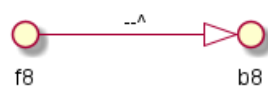
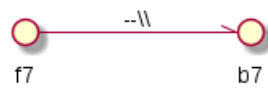
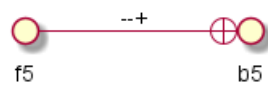
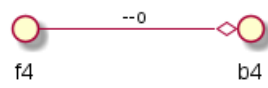
### 8.9.1 Type of arrow head

```

@startuml
left to right direction

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
```

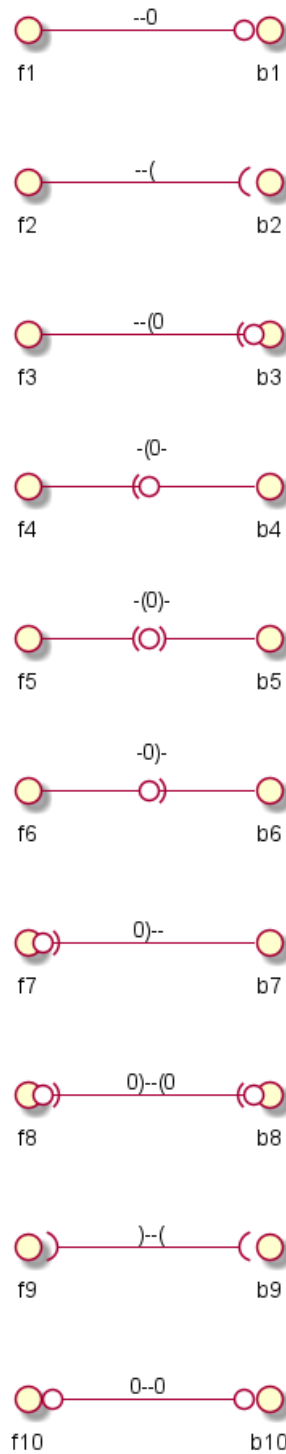


**8.9.2 Type of '0' arrow or circle arrow**

```
@startuml
left to right direction

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
```





## 8.10 Specific SkinParameter

### 8.10.1 roundCorner

```

@startuml
skinparam roundCorner 15
actor actor
agent agent
artifact artifact
boundary boundary

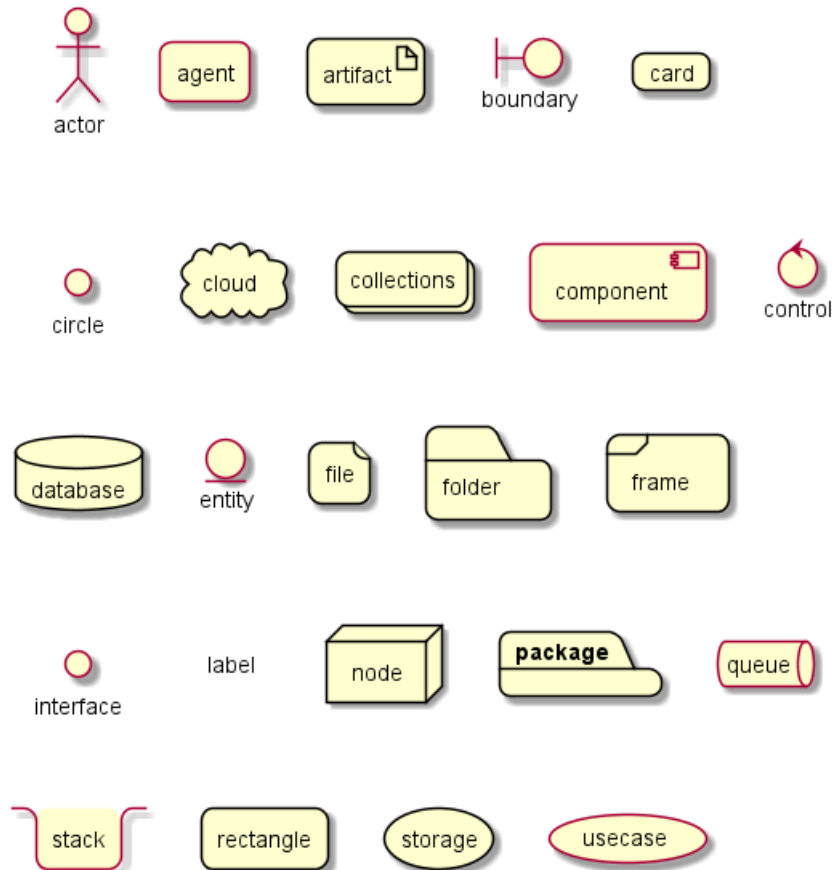
```



```

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

```



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

## 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

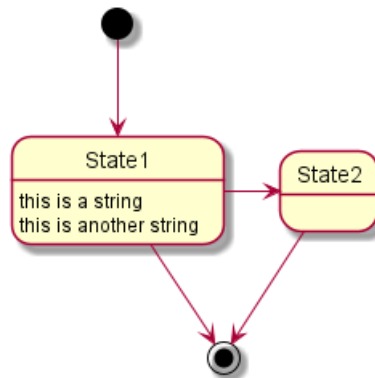
You can use [\*] for the starting point and ending point of the state diagram.

Use --> for arrows.

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

State1 -> State2
State2 --> [*]

@enduml
```



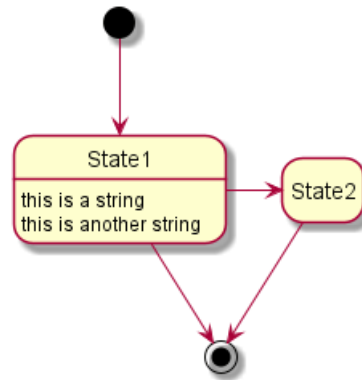
### 9.2 Change state rendering

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

```

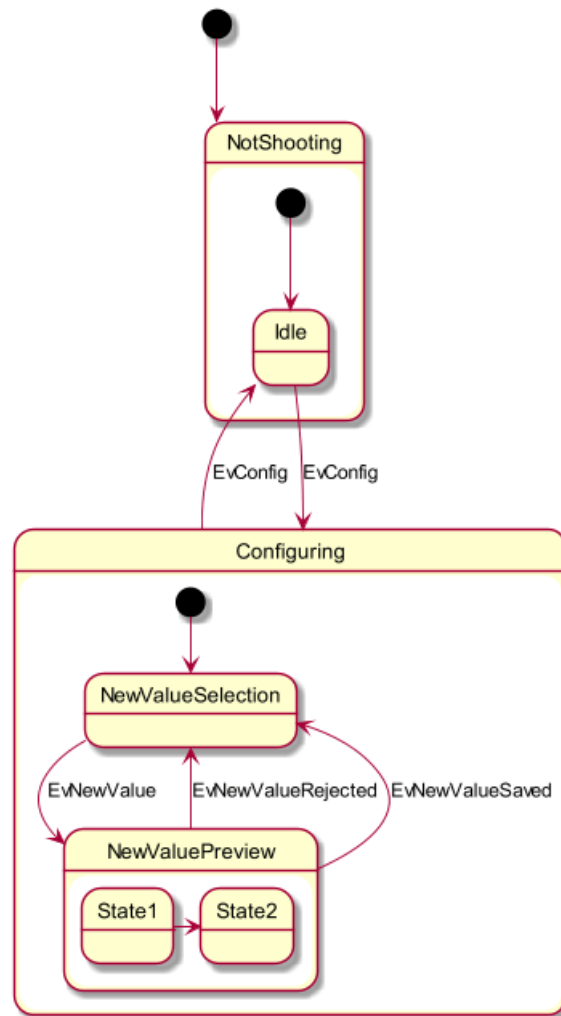
@startuml
scale 350 width
[*] --> NotShooting

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
  
```



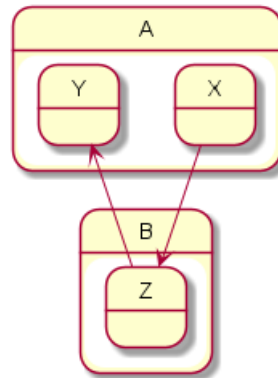
### 9.3.2 Sub-state to sub-state

```

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

state B {
  state Z {
  }
}

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



[Ref. QA-3300]

## 9.4 Long name

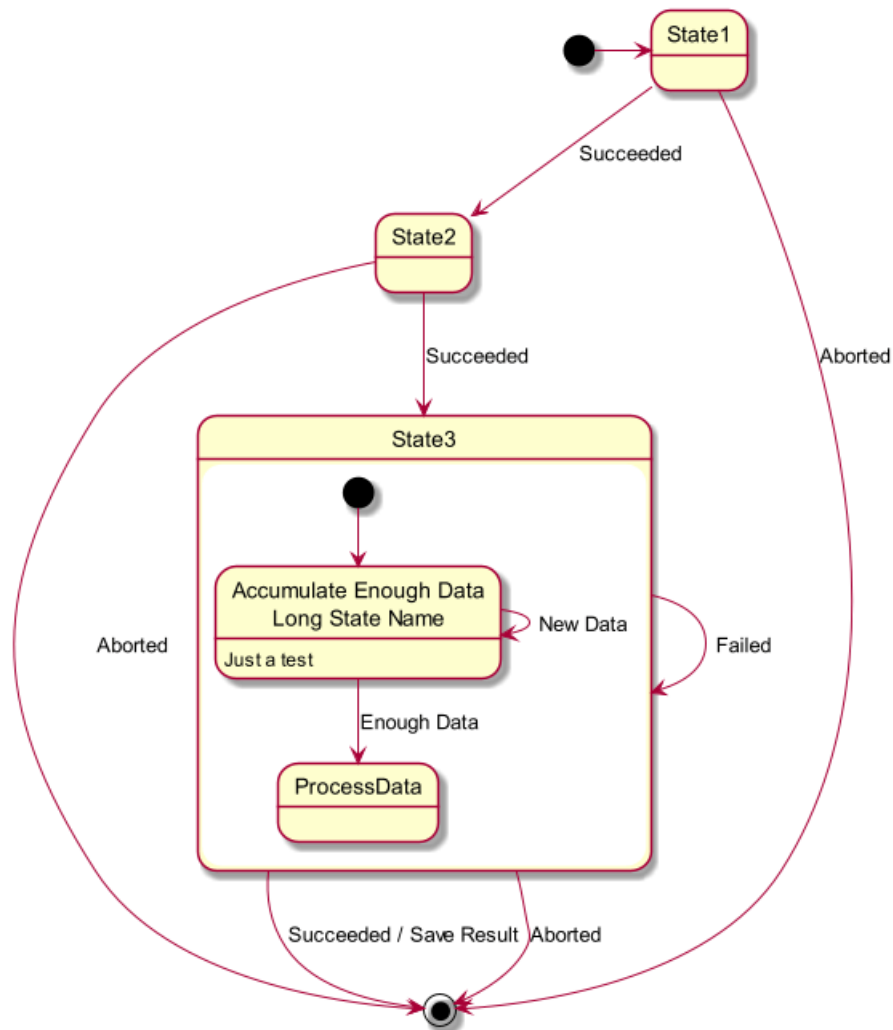
You can also use the state keyword to use long description for states.

```

@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
  
```



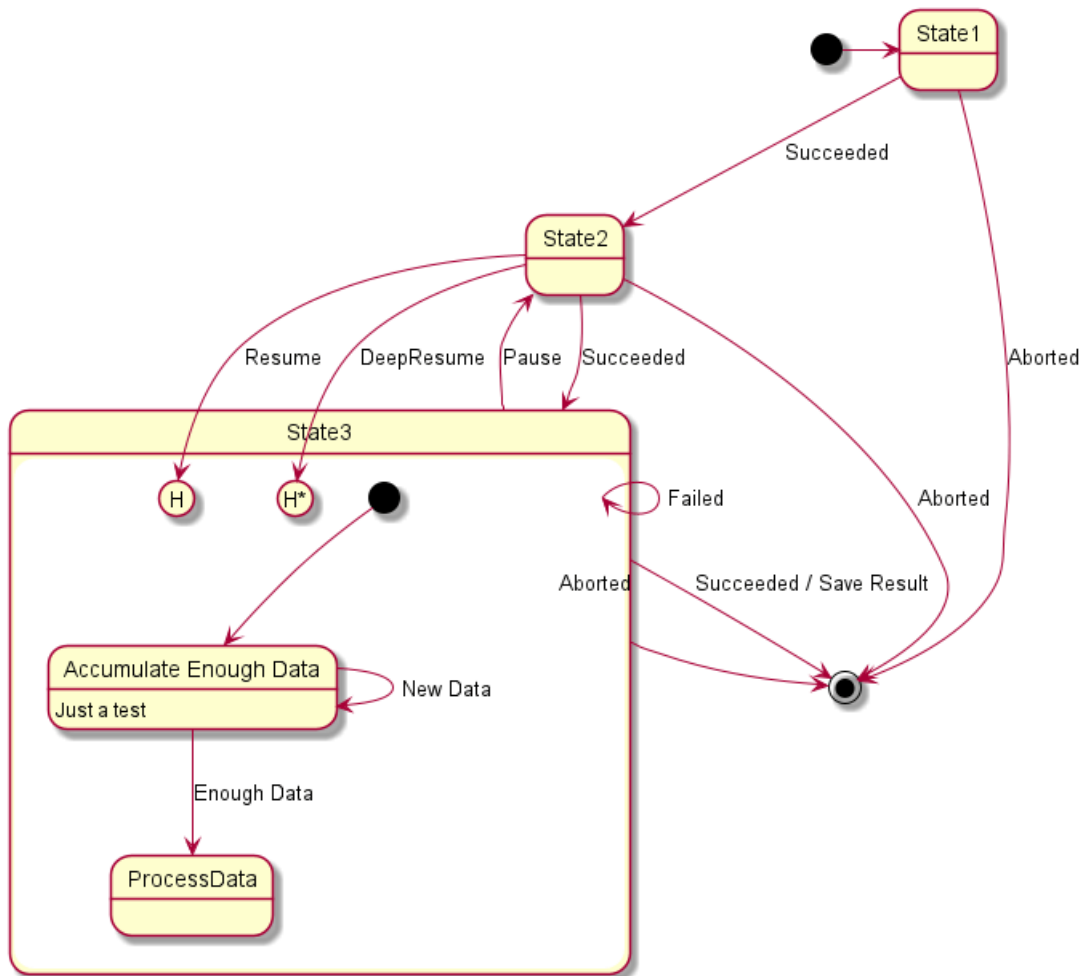
## 9.5 History [[H], [H\*]]

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]

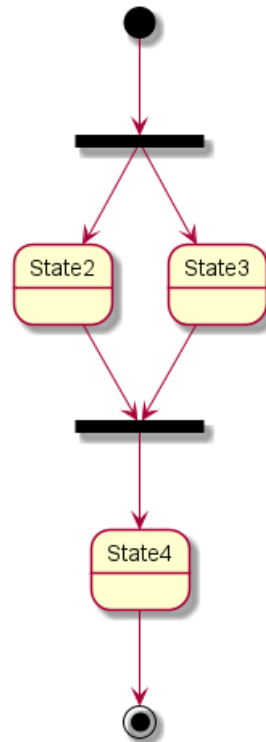
You can also fork and join using the <<fork>> and <<join>> stereotypes.

```
@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 --

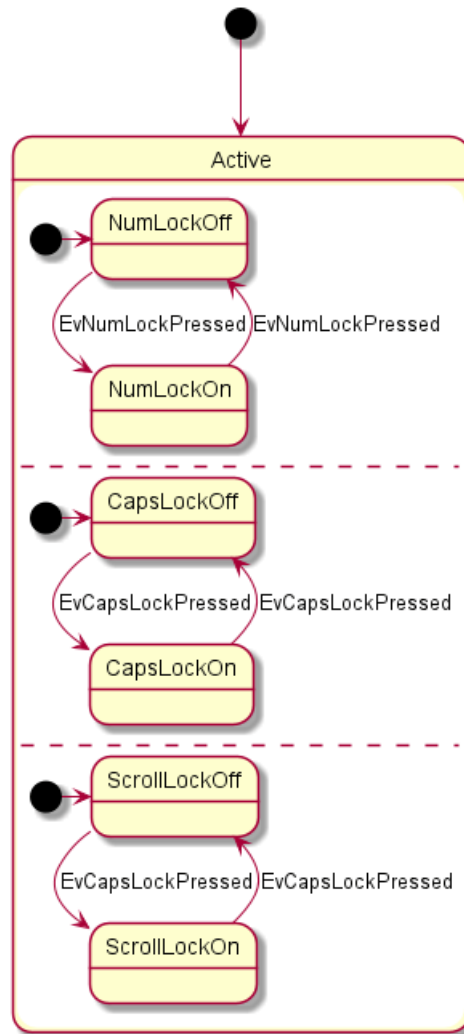
```

@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

```



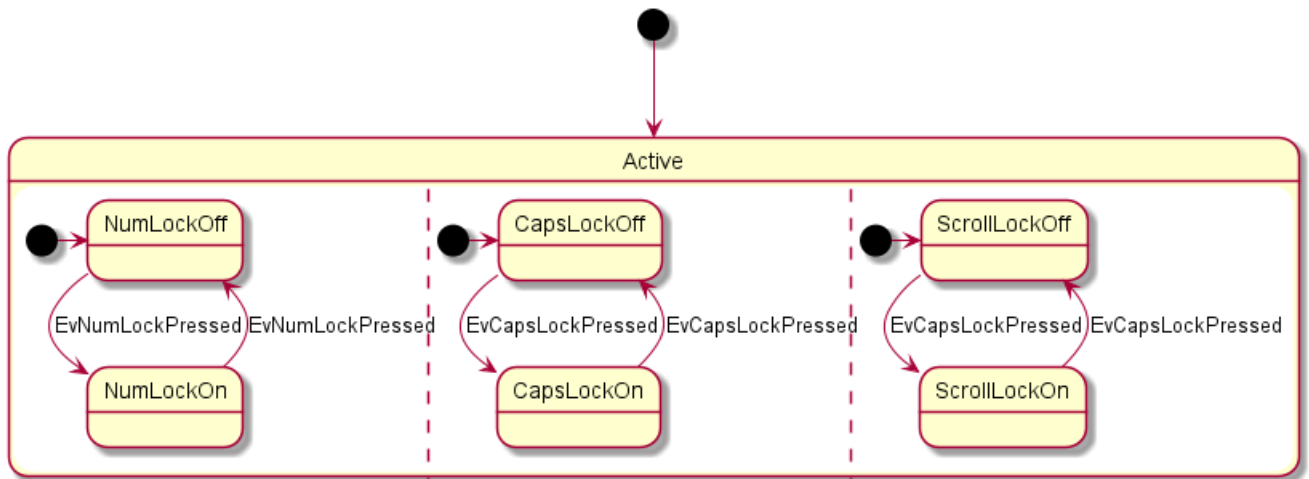
### 9.7.2 Vertical separator ||

```

@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]

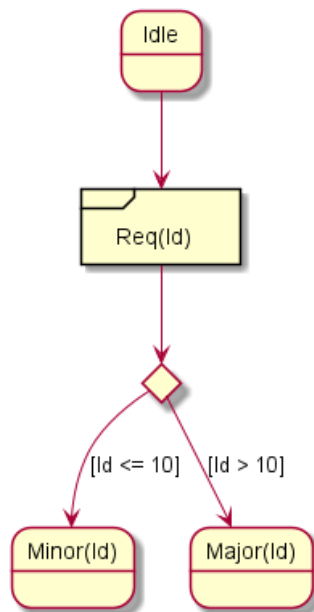
The stereotype <<choice>> can be used to use conditional state.

```

@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>>

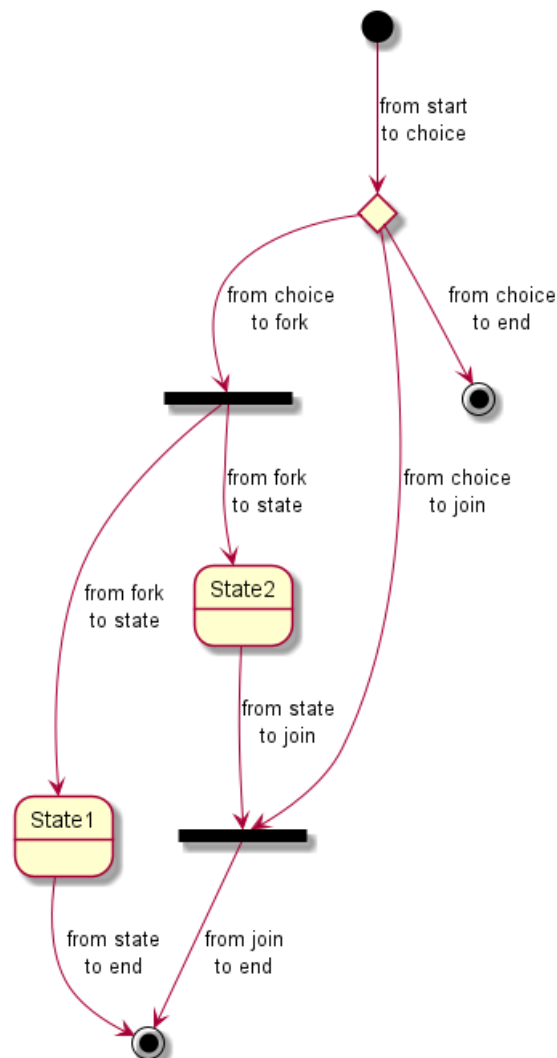
[*] --> choice1 : from start\nto choice
choice1 --> fork1 : from choice\nto fork
choice1 --> join2 : from choice\nto join
choice1 --> end3 : from choice\nto end

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:

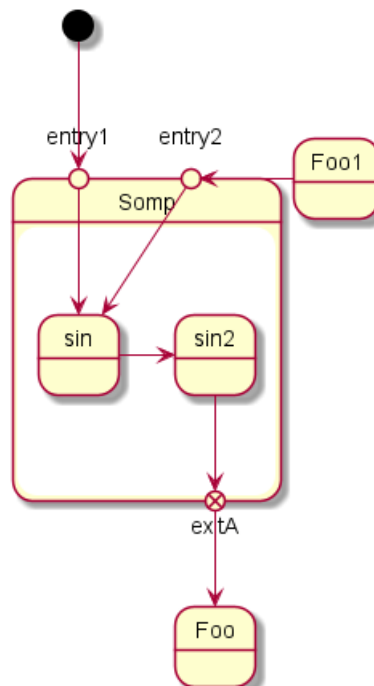


```

@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 added **pin** with `<<inputPin>>` and `<<outputPin>>` stereotypes:

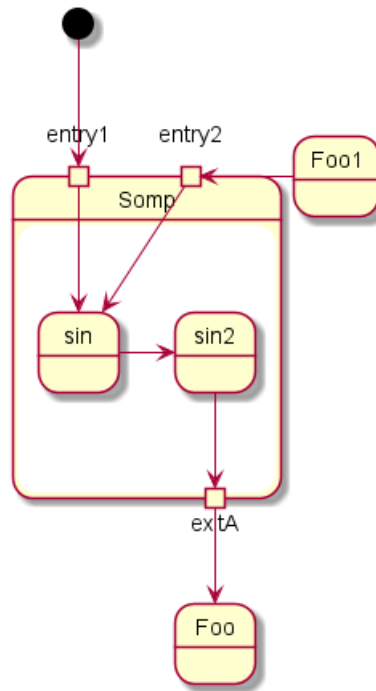
```

@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]

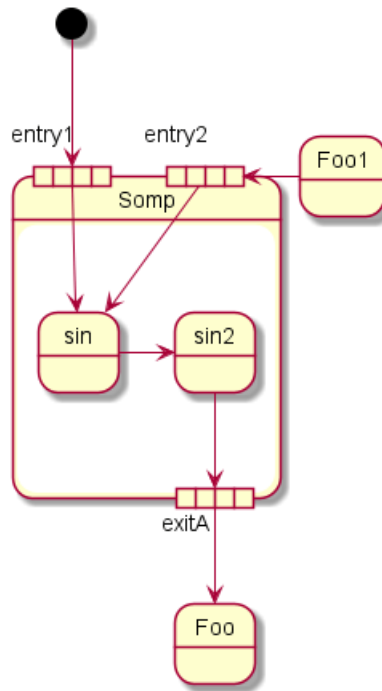
You can added **expansion** with `<<expansionInput>>` and `<<expansionOutput>>` stereotypes:

```

@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:

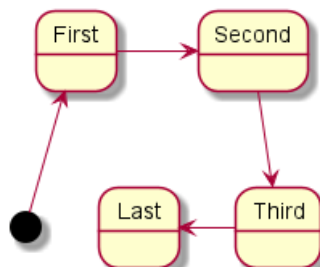
- `-down->` or `-->`
- `-right->` or `->` (default arrow)
- `-left->`
- `-up->`

@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.

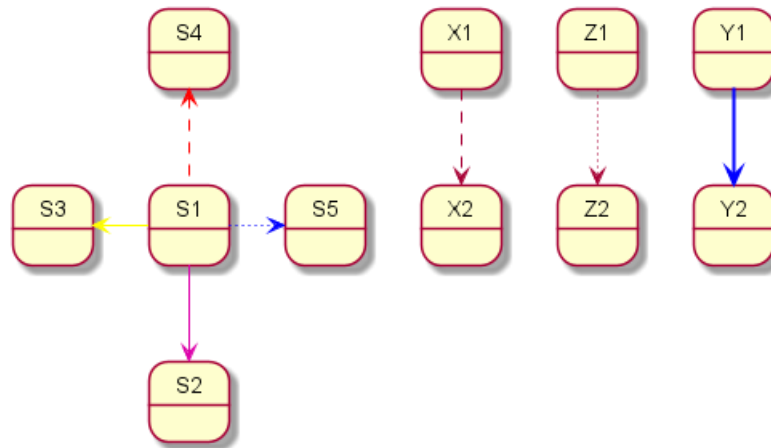


## 9.14 Change line color and style

You can change line color and/or line style.

```
@startuml
State S1
State S2
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

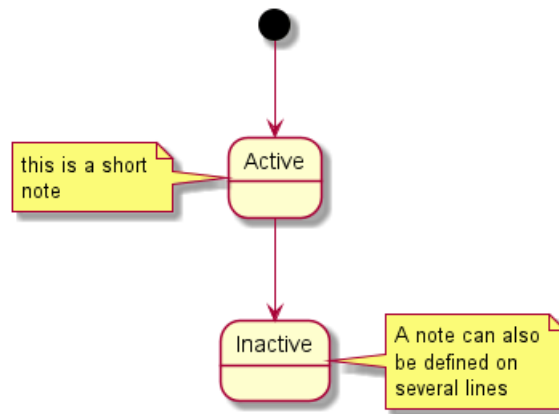
note left of Active : this is a short\nnote

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
  
```

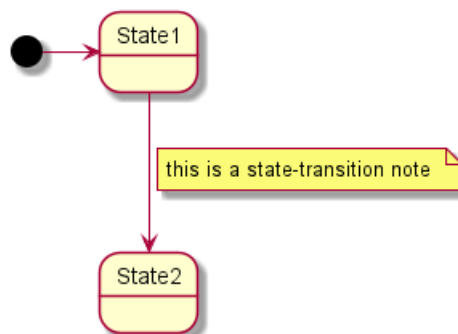


## 9.16 Note on link

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
  
```



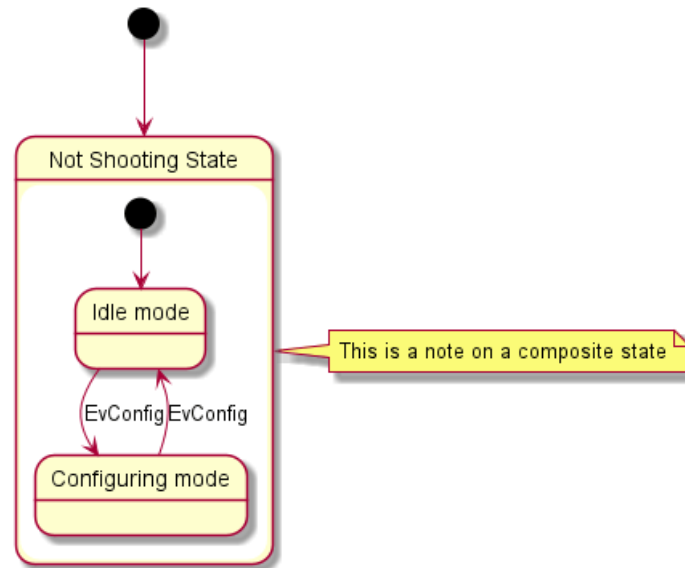
```

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

```

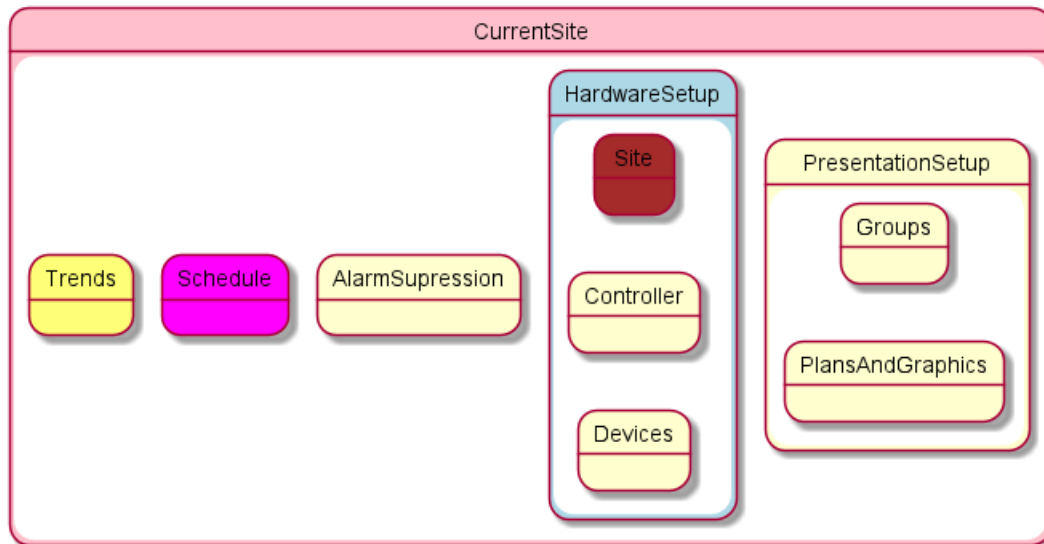


## 9.18 Inline color

```

@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
}
```

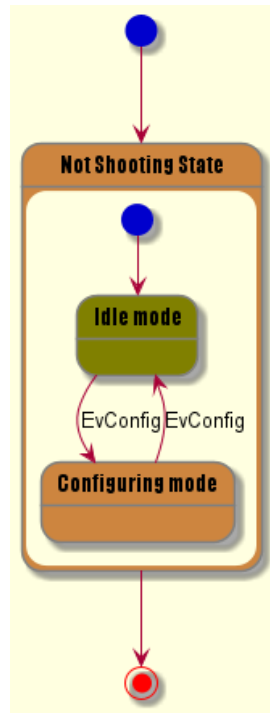
```
[*] --> NotShooting
```

```
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
```

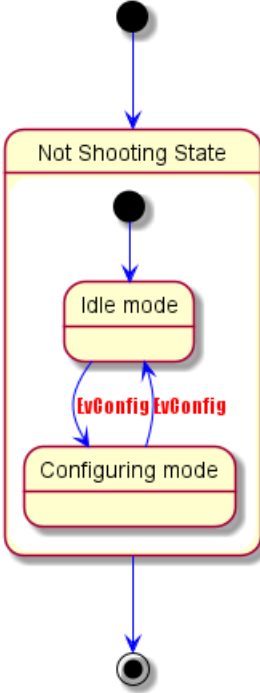
```
<style>
stateDiagram {
  BackgroundColor Peru
  'LineColor Gray
  FontName Impact
  FontColor Red
  arrow {
    FontSize 13
    LineColor Blue
  }
}
</style>
```

```
[*] --> 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
```





## 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).

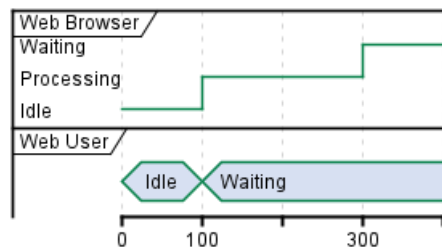
You define state change using the @ notation, and the is verb.

```
@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**

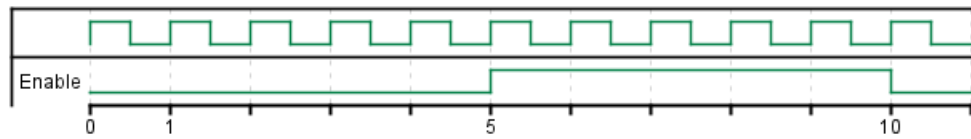
```
@startuml
clock clk with period 1
binary "Enable" as EN

@0
EN is low

@5
EN is high
```



```
@10
EN is low
@enduml
```



### 10.3 Adding message

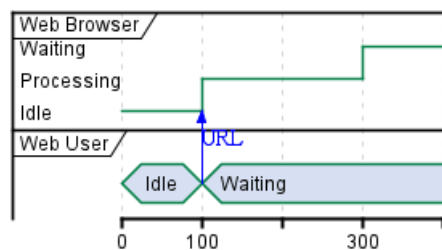
You can add message using the following syntax.

```
@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
```



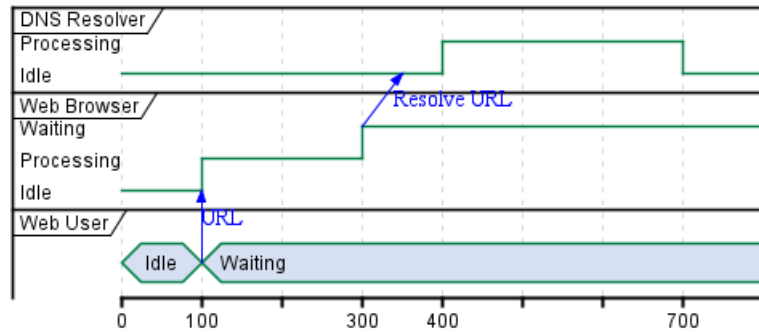
```

@+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"

```

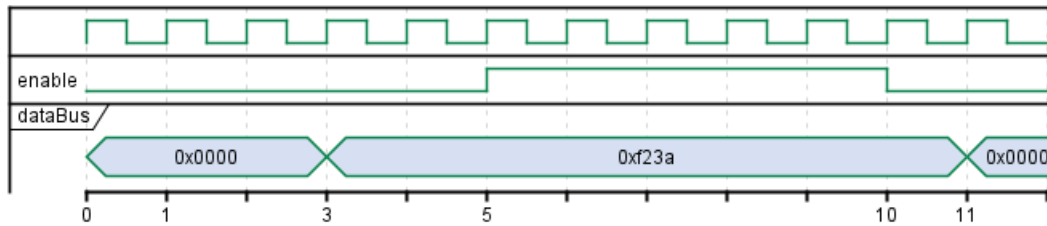
```

@:en_high+6
db is "0x0000"
@enduml

```







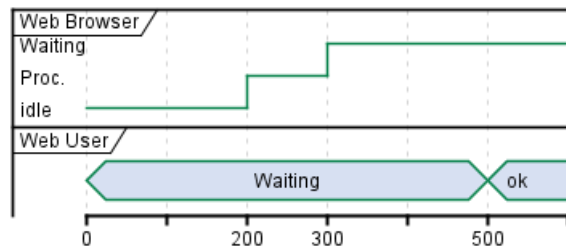
## 10.6 Participant oriented

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
```

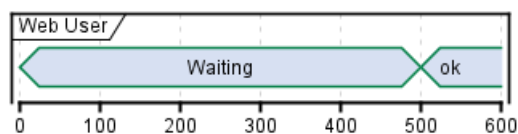


## 10.7 Setting scale

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.



```

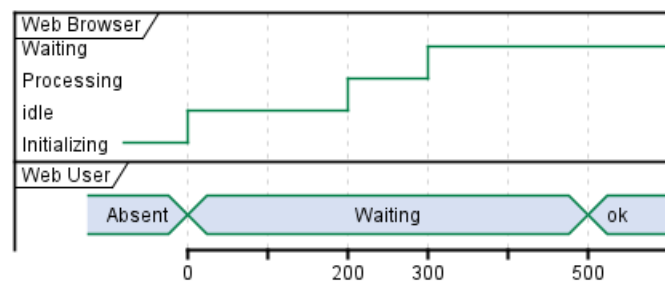
@startuml
robust "Web Browser" as WB
concise "Web User" as WU

WB is Initializing
WU is Absent

@WB
0 is idle
+200 is Processing
+100 is Waiting

@WU
0 is Waiting
+500 is ok
@enduml

```



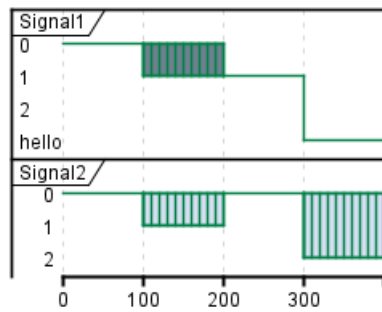
## 10.9 Intricated state

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 {-}
```

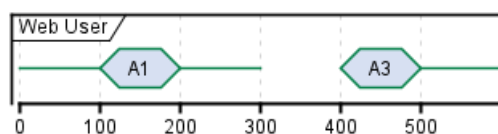
```
@100
WU is A1
```

```
@200
WU is {-}
```

```
@300
WU is {hidden}
```

```
@400
WU is A3
```

```
@500
WU is {-}
@enduml
```



## 10.11 Hide time axis

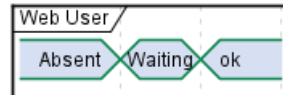
It is possible to hide time axis.

```
@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.

```

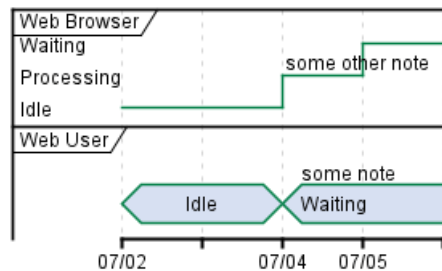
@startuml
robust "Web Browser" as WB
concise "Web User" as WU

@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

```



```

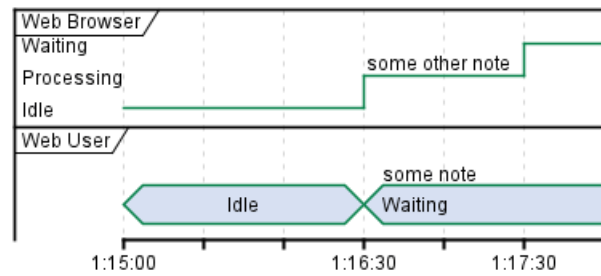
@startuml
robust "Web Browser" as WB
concise "Web User" as WU

@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

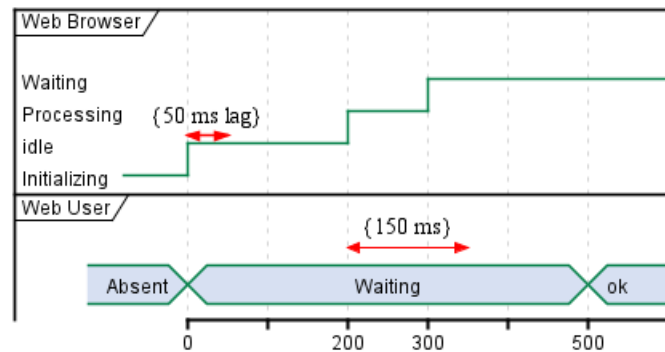
It is possible to display time constraints on the diagrams.

```
@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.

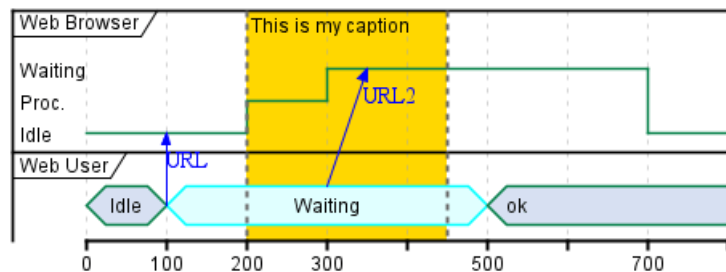
@300
WU -> WB@350 : URL2
WB is Waiting

@+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

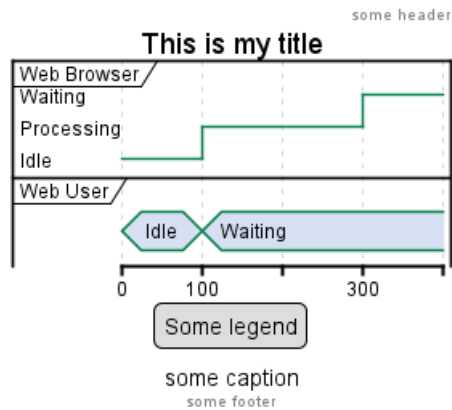
@0
WU is Idle
WB is Idle

@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

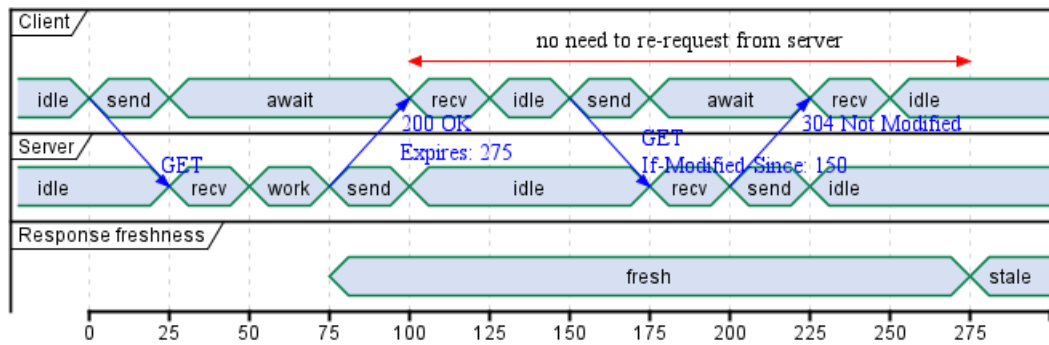
@Client
0 is send
Client -> Server@+25 : GET
+25 is await
+75 is recv
+25 is idle
+25 is send
Client -> Server@+25 : GET\nIf-Modified-Since: 150
+25 is await
+50 is recv
+25 is idle
@100 <-> @275 : no need to re-request from server

@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"

```

```

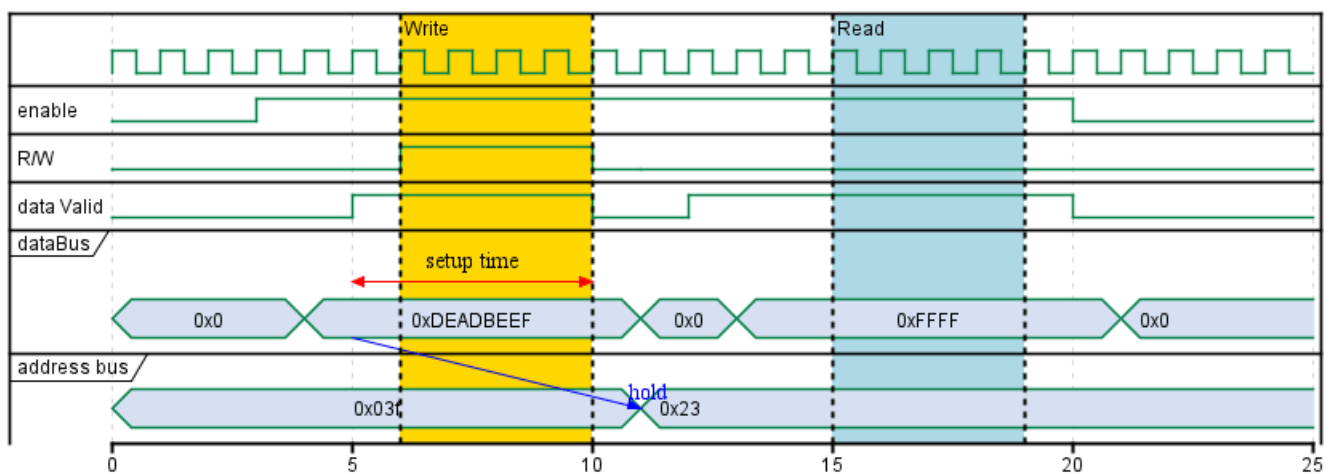
highlight :write_beg to :write_end #Gold:Write
highlight :read_beg to :read_end #lightBlue:Read

```

```

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

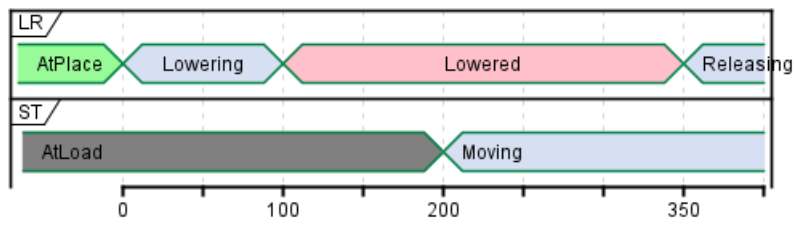
```

```

@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
```

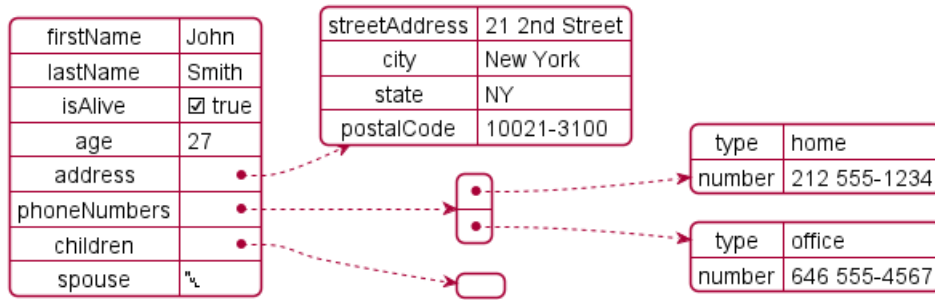
fruit	Apple
size	Large
color	Red

### 11.1 Complex example

You can use complex JSON structure.

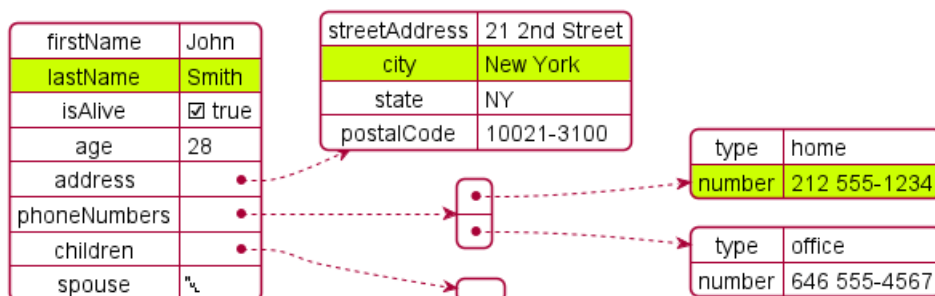
```
@startjson
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
@endjson
```





## 11.2 Highlight parts

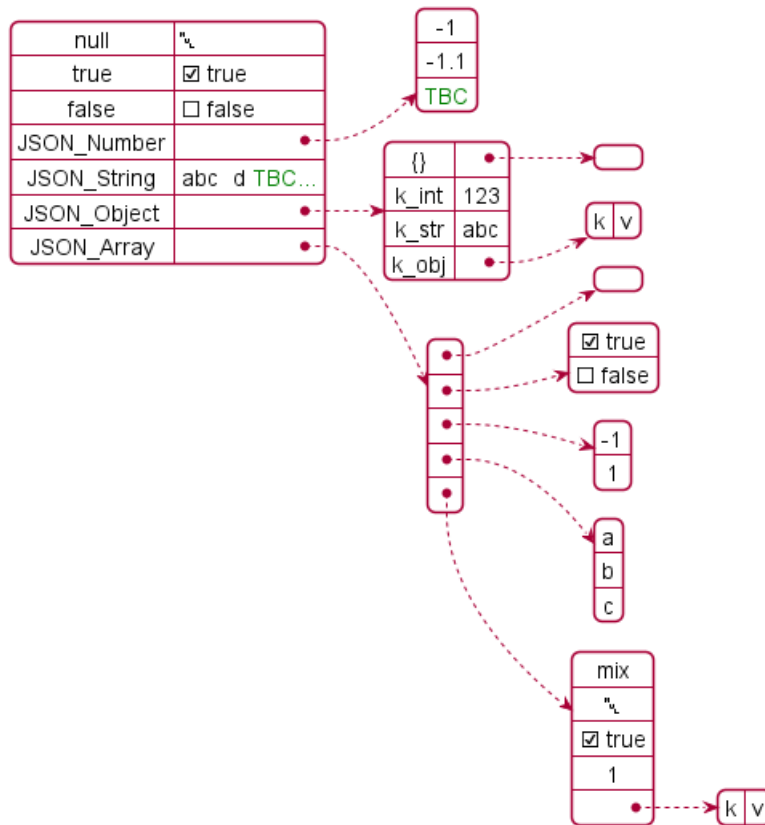
```
@startjson
#highlight "lastName"
#highlight "address" / "city"
#highlight "phoneNumbers" / "0" / "number"
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 28,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "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

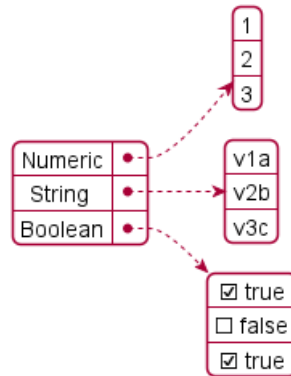
```
@startjson
{
  "null": null,
  "true": true,
  "false": false,
  "JSON_Number": [-1, -1.1, "<color:green>TBC"],
  "JSON_String": "a\nb\rc\td <color:green>TBC...",
  "JSON_Object": {
    "{}": {},
    "k_int": 123,
    "k_str": "abc",
    "k_obj": {"k": "v"}
  },
  "JSON_Array" : [
    [],
    [true, 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

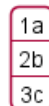
### 11.4.3 Number array

```
@startjson
[1, 2, 3]
@endjson
```



### 11.4.4 String array

```
@startjson
["1a", "2b", "3c"]
@endjson
```



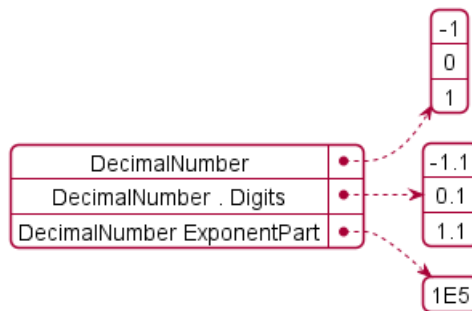
### 11.4.5 Boolean array

```
@startjson
[true, false, true]
@endjson
```

<input checked="" type="checkbox"/> true
<input type="checkbox"/> false
<input checked="" type="checkbox"/> true

### 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 .

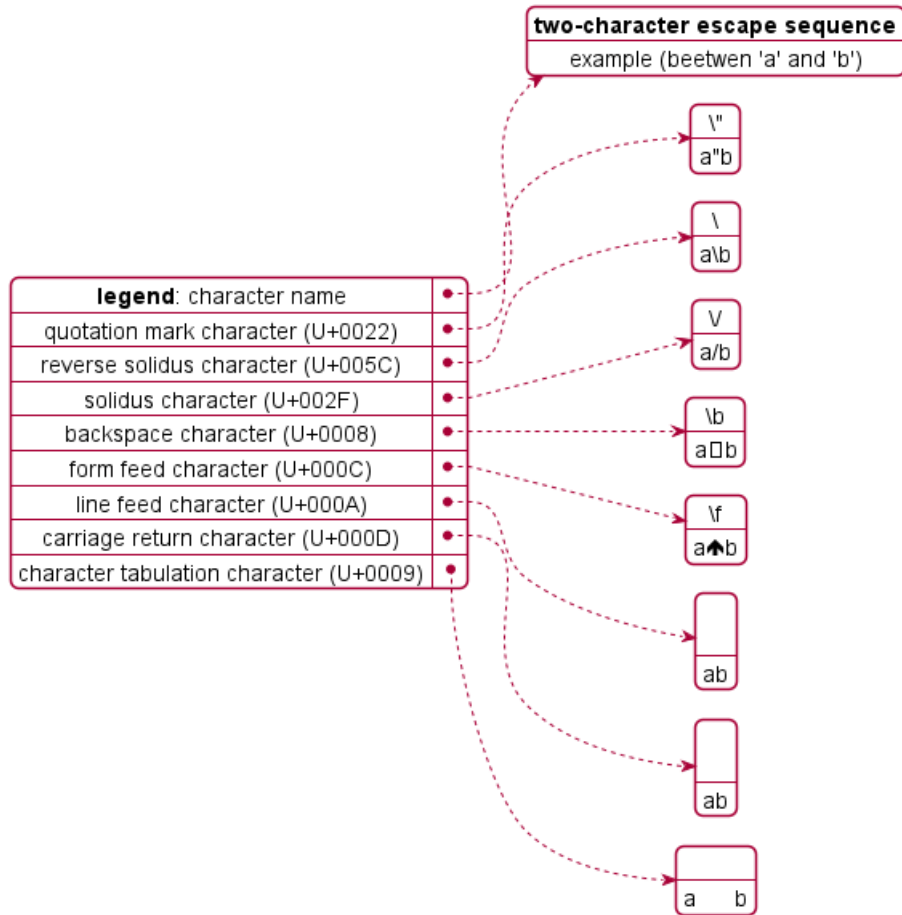
```
@startjson
{
  "<color:blue><b>code": "<color:blue><b>value",
  "a\u005Cb": "a\u005Cb",
  "\uD83D\uDE10": "\uD83D\uDE10",
  " ": " "
}
@endjson
```

code	value
a\u005Cb	a\b
\uD83D\uDE10	😄

#### 11.6.2 JSON two-character escape sequence

```
@startjson
{
  "**legend**: character name": ["**two-character escape sequence**", "example (beetwen 'a' and
  "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)*

## 12 Network diagram (nwdiag)

nwdiag has been created by Takeshi Komiya and allows to quickly draw network diagrams. So we thank him for his creation!

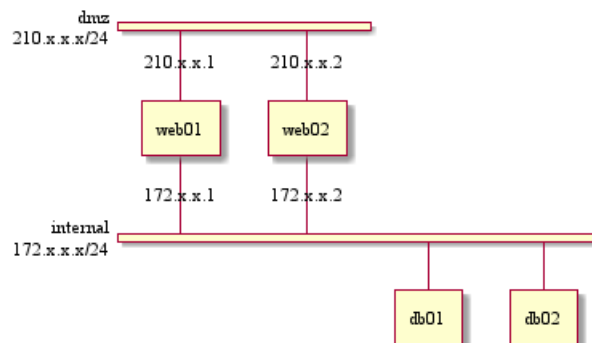
Since the syntax is clear and simple, this has been integrated within PlantUML. We reuse here the examples that Takeshi has documented.

### 12.1 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
```



### 12.2 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";

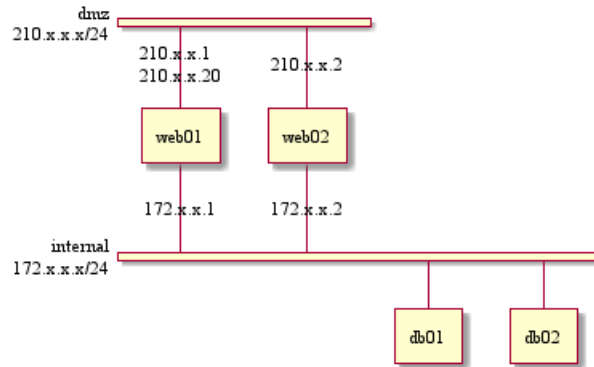
    web01 [address = "172.x.x.1"];
  }
}
@enduml
```



```

    web02 [address = "172.x.x.2"];
    db01;
    db02;
}
}
@enduml

```



### 12.3 Grouping nodes

```

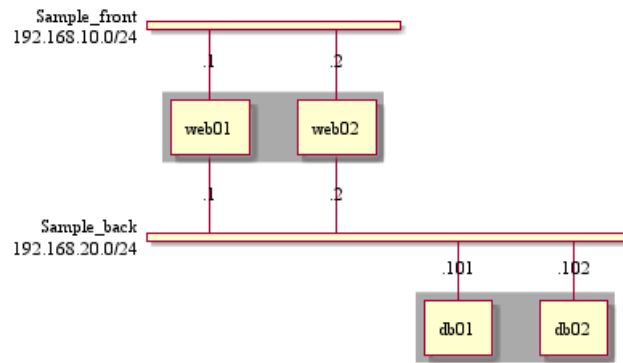
@startuml
nwdiag {
  network Sample_front {
    address = "192.168.10.0/24";

    // 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

```





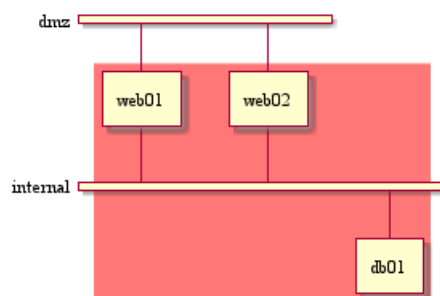
```

@startuml
nwdiag {
  // define group at outside network definitions
  group {
    color = "#FF7777";

    web01;
    web02;
    db01;
  }

  network dmz {
    web01;
    web02;
  }
  network internal {
    web01;
    web02;
    db01;
  }
}
@enduml

```



## 12.4 Extended Syntax

You can add or change:

- address;
- color;
- description;
- shape.



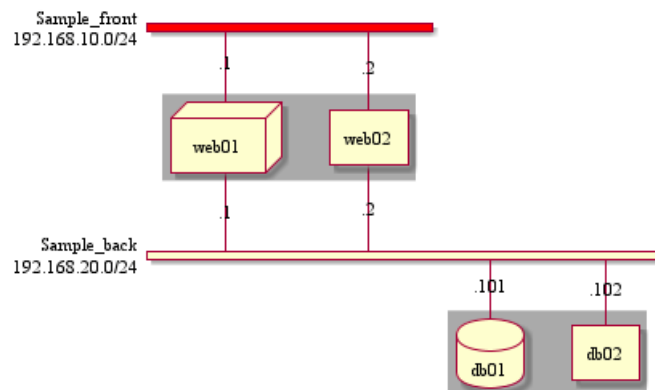
```

@startuml
nwdiag {
  network Sample_front {
    address = "192.168.10.0/24"
    color = "red"

    // define group
    group web {
      web01 [address = ".1", shape = "node"]
      web02 [address = ".2"]
    }
  }
  network Sample_back {
    address = "192.168.20.0/24"
    web01 [address = ".1"]
    web02 [address = ".2"]
    db01 [address = ".101", shape = database ]
    db02 [address = ".102"]

    // define network using defined nodes
    group db {
      db01;
      db02;
    }
  }
}
@enduml

```



## 12.5 Using Sprite on nwdiag

You can use all the Sprite of all Standard Library or other.

```

@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"];
  }
}

```

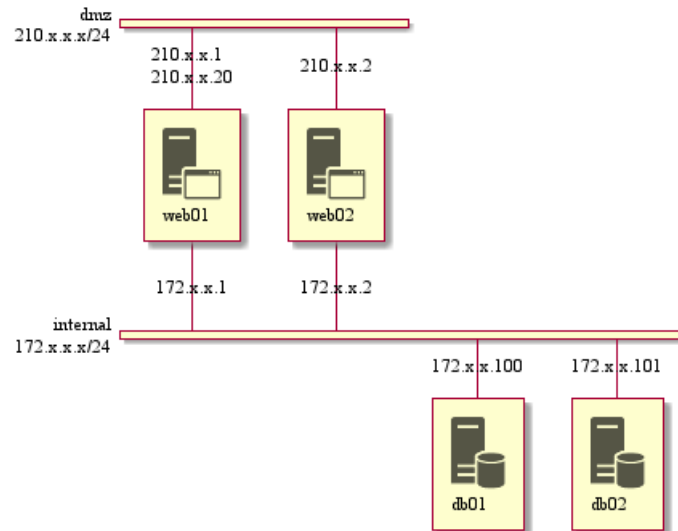


```

}
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]

## 12.6 Using OpenIconic on nwdiag

You can also use the icons from OpenIconic on the description.

```

@startuml
nwdiag {
  network dmz {
    address = "210.x.x.x/24"

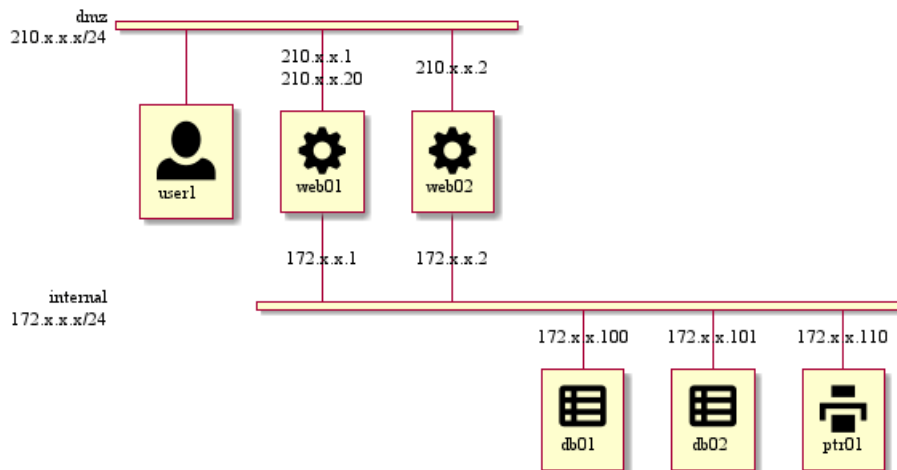
    user [description = "<&person*5>\n user1"];
    // set multiple addresses (using comma)
    web01 [address = "210.x.x.1, 210.x.x.20", description = "<&cog*4>\nweb01"];
    web02 [address = "210.x.x.2", description = "<&cog*4>\nweb02"];
  }
  network internal {
    address = "172.x.x.x/24";

    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
```



## 12.7 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";

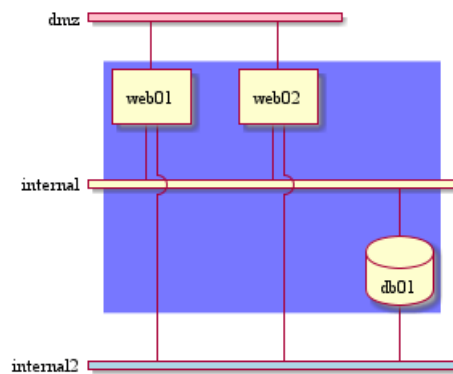
    web01;
    web02;
    db01;
  }
}
```



```

}
@enduml

```



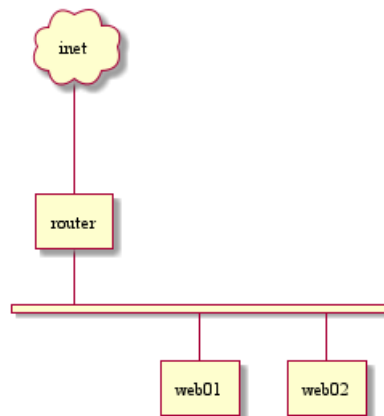
## 12.8 Peer networks

```

@startuml
nwdiag {
  inet [shape = cloud];
  inet -- router;

  network {
    router;
    web01;
    web02;
  }
}
@enduml

```



## 12.9 Peer networks and group

```

@startuml
nwdiag {
  internet [ shape = cloud];
  internet -- router;

  group {
    color = "pink";

```





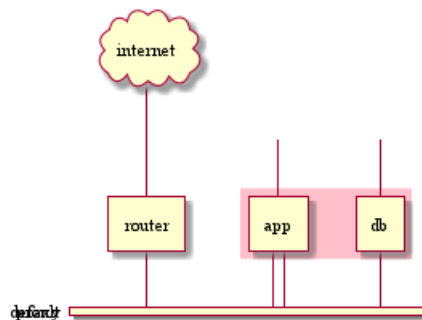
```

    app;
    db;
  }

  network proxy {
    width = full
    router;
    app;
  }

  network default {
    width = full
    app;
    db;
  }
}
@enduml

```



## 12.10 Add title, header, footer or legend on network diagram

```

@startuml

header some header

footer some footer

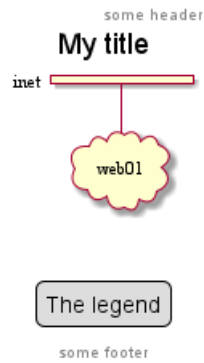
title My title

nwdiag {
  network inet {
    web01 [shape = cloud]
  }
}

legend
The legend
end legend

@enduml

```



[Ref: QA-11303]

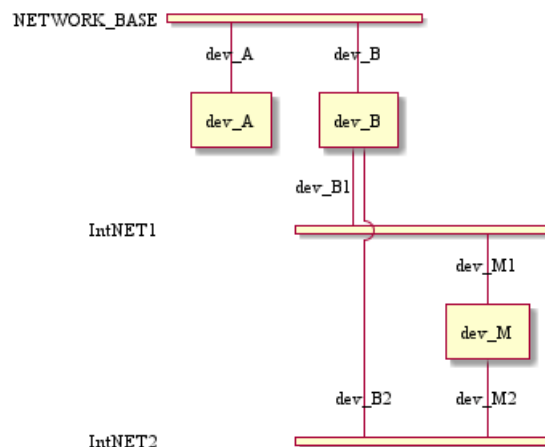
## 12.11 Change width of the networks

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

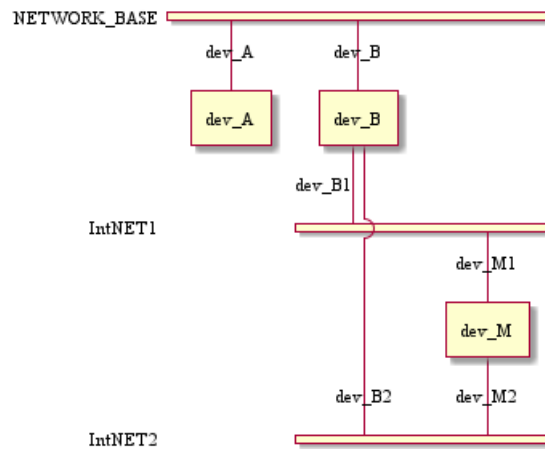
```
@startuml
nwdiag {
  network NETWORK_BASE {
    width = full
  }
}
```



```

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

```

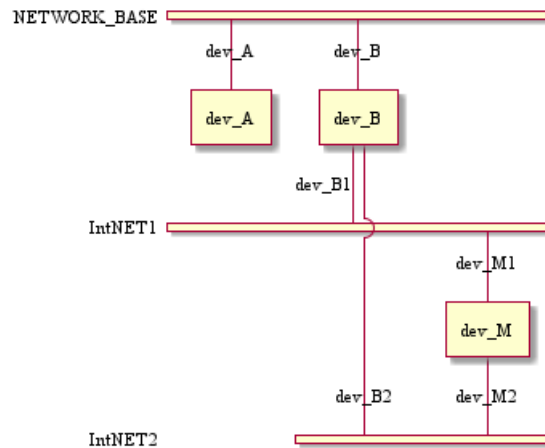


- the first and the second

```

@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

```

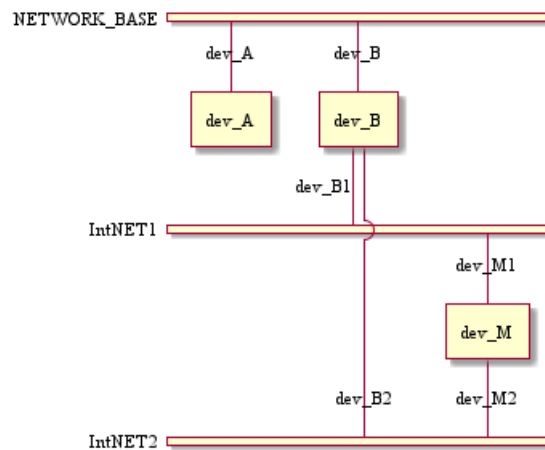


- all the network (with same full width)

```

@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 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.

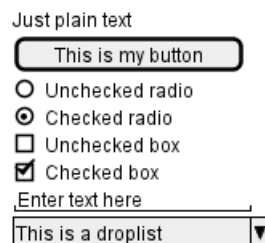
You can use either `@startsalt` keyword, or `@startuml` followed by a line with `salt` keyword.

### 13.1 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 `"`.
- Droplist 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 droplist^
}
@endsalt
```

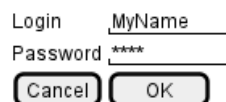


### 13.2 Using grid [[]]

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
```

### 13.3 Group box [^]

```
@startsalt
{^"My group box"
  Login   | "MyName  "
  Password| "****    "
  [Cancel]| [ OK   ]
}
@endsalt
```

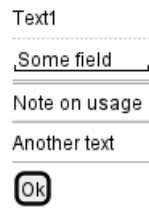
[Ref. QA-5840]

### 13.4 Using separator [., ==, ~~, --]

You can use several horizontal lines as separator.

```
@startsalt
{
  Text1
  ..
  "Some field"
  ==
  Note on usage
  ~~
  Another text
  --
  [Ok]
}
@endsalt
```





## 13.5 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
```



## 13.6 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]

### 13.7 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: \_\_\_\_\_

Modifiers:  public  default  private  protected  
 abstract  final  static

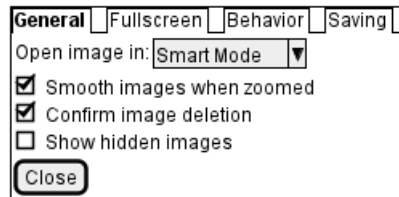
Superclass:

### 13.8 Adding tabs [/]

You can add tabs using {/ notation. Note that you can use HTML code to have bold text.

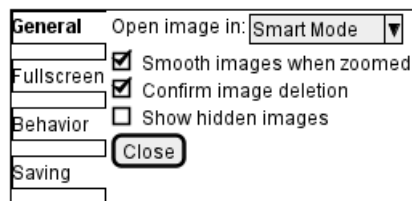
```
@startsalt
{+
{/ <b>General | Fullscreen | Behavior | Saving }
{
{ Open image in: | ^Smart Mode^ }
[X] Smooth images when zoomed
[X] Confirm image deletion
[ ] Show hidden images
}
[Close]
}
@endsalt
```





Tab could also be vertically oriented:

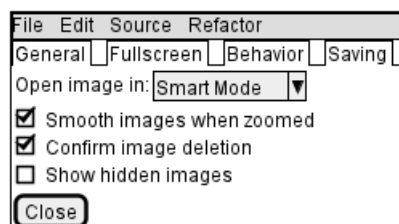
```
@startsalt
{+
{/ <b>General
Fullscreen
Behavior
Saving } |
{
{ Open image in: | ^Smart Mode^ }
[X] Smooth images when zoomed
[X] Confirm image deletion
[ ] Show hidden images
[Close]
}
}
@endsalt
```



## 13.9 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
```



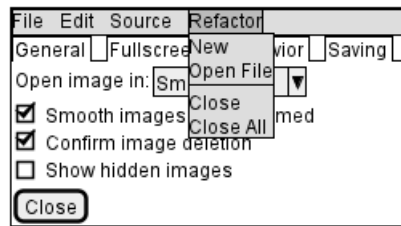
It is also possible to open a menu:



```

@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

```



### 13.10 Advanced table

You can use two special notations for table :

- \* to indicate that a cell with span with left
- . to denotate 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	

### 13.11 Scroll Bars [S, SI, S-]

You can use {S notation for scroll bar like in following examples:

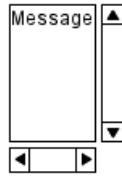
- {S: for horizontal and vertical scrollbars

```

@startsalt
{S
Message
.
.
.
.
}
@endsalt

```





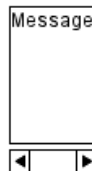
- {SI : for vertical scrollbar only

```
@startsalt
{SI
Message
.
.
.
.
}
@endsalt
```



- {S- : for horizontal scrollbar only

```
@startsalt
{S-
Message
.
.
.
.
}
@endsalt
```



## 13.12 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
```



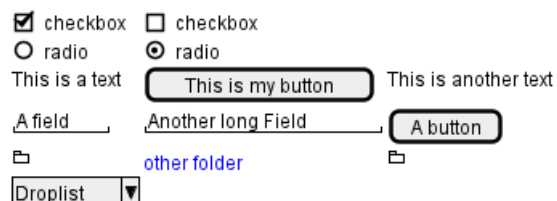


[Ref. QA-12177]

### 13.13 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
  .....
  .XXXXX.....
  .X...X.....
  .XXXXXXXXXX.
  .X.....X.
  .X.....X.
  .X.....X.
  .X.....X.
  .XXXXXXXXXX.
  .....
  >> | <color:blue>other folder | <<folder>>
  ^Droplist^
}
@endsalt
```



[Ref. QA-5849]

### 13.14 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
```



The complete list is available on OpenIconic Website, or you can use the following special diagram:

```
@startuml
listopeniconic
@enduml
```

List Open Iconic						
<i>Credit to</i> <a href="https://useiconic.com/open">https://useiconic.com/open</a>	bell	cloud	excerpt	justify-right	musical-note	star
	bluetooth	cloudy	expand-down	key	paperclip	sun
	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
action-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	microphone	shield	zoom-in
bar-chart	circle-x	elevator	infinity	minus	signal	zoom-out
basket	clipboard	ellipses	info	monitor	signpost	
battery-empty	clock	envelope-closed	italic	moon	sort-ascending	
battery-full	cloud-download	envelope-open	justify-center	move	sort-descending	
beaker	cloud-upload	euro	justify-left		spreadsheet	

## 13.15 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
```

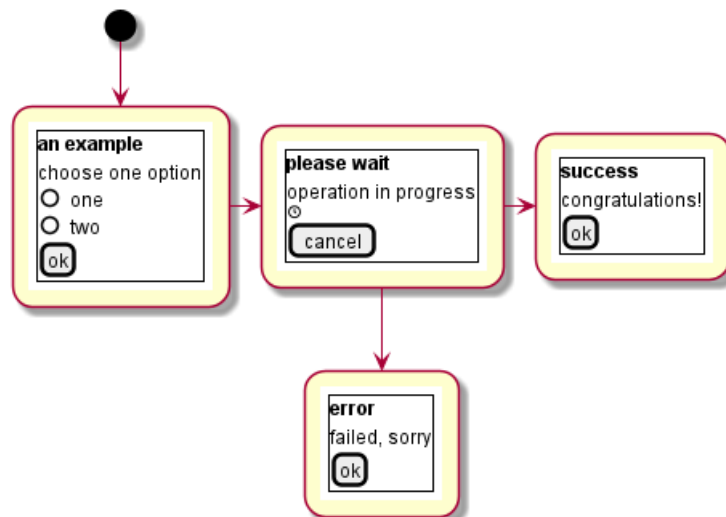


```

{+
<b>please wait
operation in progress
<&clock>
[cancel]
}
}}
" as wait
wait -right-> "
{{
salt
{+
<b>success
congratulations!
[ok]
}
}}
" as success

wait -down-> "
{{
salt
{+
<b>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

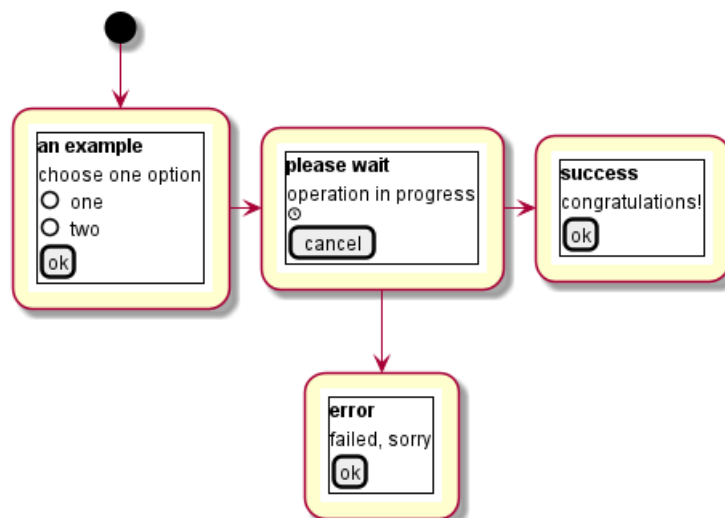
!procedure _wait()
{+
<b>please wait
operation in progress
<&clock>
[cancel]
}
!endprocedure

!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

```





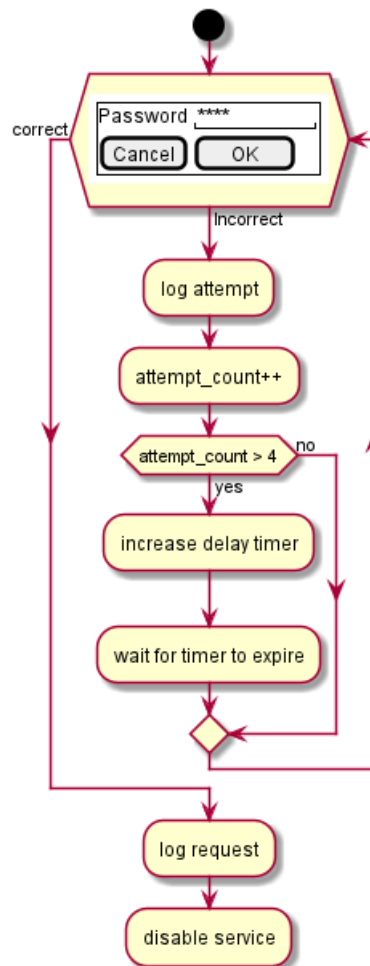
## 13.16 Include salt "on while condition of activity diagram"

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
endif
endwhile (correct)
:log request;
:disable service;
@enduml

```



[Ref. QA-8547]

## 14 Archimate Diagram

This is only a proposal and subject to change.

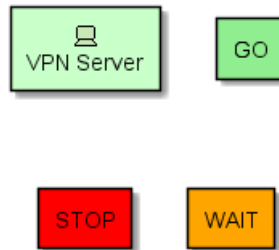
You are very welcome to create a new discussion on this future syntax. Your feedbacks, ideas and suggestions help us to find the right solution.

### 14.1 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
```



### 14.2 Defining Junctions

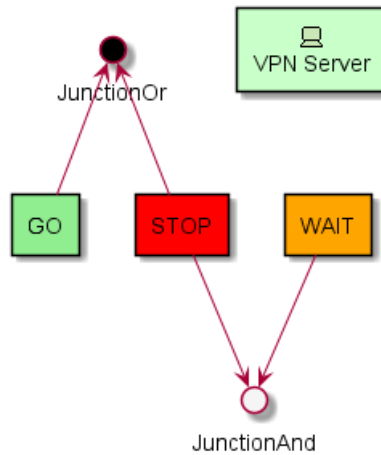
Using the circle keyword and the preprocessor, you can also create junctions.

```
@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
```



### 14.3 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 <<$aService>><<behavior>> #Application
rectangle "Customer administration" as customerAdministration <<$aService>><<behavior>> #Application
rectangle "Claims administration" 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

```

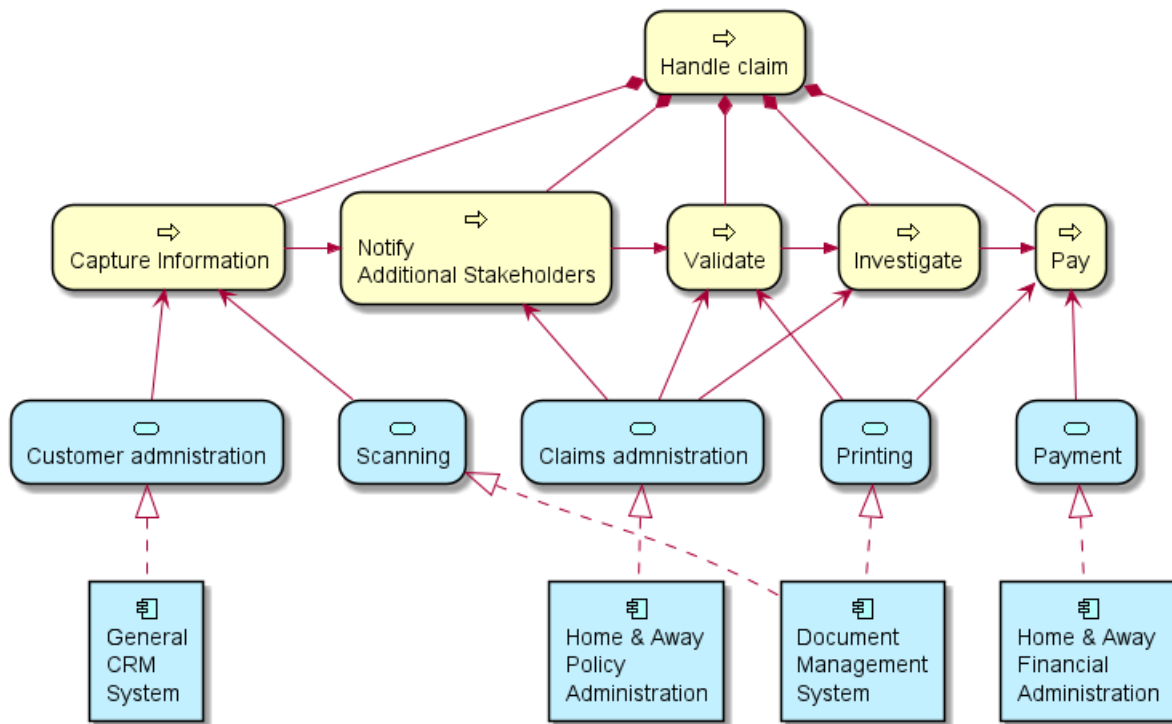
rectangle "Document\nManagement\nSystem" as DMS <<$aComponent>> #Application
rectangle "General\nCRM\nSystem" as CRM <<$aComponent>> #Application
rectangle "Home & Away\nPolicy\nAdministration" as HAPA <<$aComponent>> #Application
rectangle "Home & Away\nFinancial\nAdministration" as HFPA <<$aComponent>> #Application
    
```

```

DMS .up.|> scanning
DMS .up.|> Printing
CRM .up.|> customerAdministration
HAPA .up.|> claimsAdministration
HFPA .up.|> Payment
    
```

```

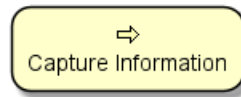
legend left
Example from the "Archinsurance case study" (OpenGroup).
See
====
<$bProcess> :business process
====
<$aService> : application service
====
<$aComponent> : application component
endlegend
@enduml
    
```



Example from the "Archinsurance case study" (OpenGroup). See
⇒ :business process
□ : application service
⊞ : application component

## 14.4 Example 2

```
@startuml
skinparam roundcorner 25
rectangle "Capture Information" as CI <<$archimate/business-process>> #Business
@enduml
```



## 14.5 List possible sprites

You can list all possible sprites for Archimate using the following diagram:

```
@startuml
listsprite
@enduml
```

### List Current Sprites

Credit to  
<http://www.archimatetool.com>

archimate :

access  
activity  
actor  
aggregation  
application-collaboration  
application-component  
application-data-object  
application-event  
application-function  
application-interface  
application-process  
application-service  
assessment-filled  
assessment  
assignment  
association-unidirect  
association  
business-activity  
business-actor  
business-collaboration  
business-contract  
business-event  
business-function  
business-interaction  
business-interface  
business-location  
business-meaning

business-object  
business-process  
business-product  
business-representation  
business-role  
business-service  
business-value  
collaboration  
communication-path  
component  
composition  
constraint-filled  
constraint  
contract  
deliverable-filled  
deliverable  
device  
driver-filled  
driver  
event  
flow  
function  
gap-filled  
gap  
goal-filled  
goal  
implementation-deliverable  
implementation-event  
implementation-gap  
implementation-plateau  
implementation-workpackage  
influence  
interaction  
interface-required

interface-symmetric  
interface  
junction-and  
junction-or  
junction  
location  
meaning  
motivation-assessment  
motivation-constraint  
motivation-driver  
motivation-goal  
motivation-meaning  
motivation-outcome  
motivation-principle  
motivation-requirement  
motivation-stakeholder  
motivation-value  
network  
node  
object  
physical-distribution-network  
physical-equipment  
physical-facility  
physical-material  
plateau  
principle-filled  
principle  
process  
product  
realisation  
representation  
requirement-filled  
requirement  
role

service  
serving  
specialisation  
specialization  
stakeholder-filled  
strategy-capability  
strategy-course-of-action  
strategy-resource  
strategy-value-stream  
system-software  
technology-artifact  
technology-collaboration  
technology-communication-network  
technology-communication-path  
technology-device  
technology-event  
technology-function  
technology-infra-interface  
technology-infra-service  
technology-interaction  
technology-interface  
technology-network  
technology-node  
technology-path  
technology-process  
technology-service  
technology-system-software  
triggering  
used-by  
value  
workpackage-filled

## 14.6 ArchiMate Macros

A list of Archimate macros are defined Archimate-PlantUML here which simplifies the creation of ArchiMate diagrams.

Using the macros, creation of ArchiMate elements are done using the following format: `Category_ElementName (nameOfTheElement "description")`

For Example:

- To define a Stakeholder element, which is part of Motivation category, the syntax will be `Motivation_Stakeholder (Stakeholder "Stakeholder Description")`



- To define a Business Service element,

```
Business_Service(BService, "Business Service")
```

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(fromElement, toElement, "description")`

The RelationTypes supported are:

- Access
- Aggregation
- Assignment
- Association
- Composition
- Flow
- Influence
- Realization
- Serving
- Specilization
- 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")
```

- 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")
```

## 15 Gantt Diagram

The Gantt is described in *natural* language, using very simple sentences (subject-verb-complement).

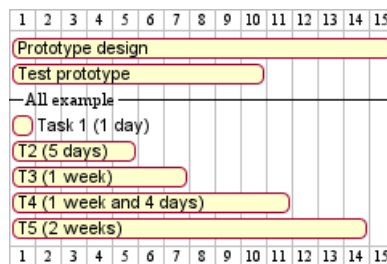
### 15.1 Declaring tasks

Tasks defined using square bracket.

#### 15.1.1 Duration

Their durations are defined using the last 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
```

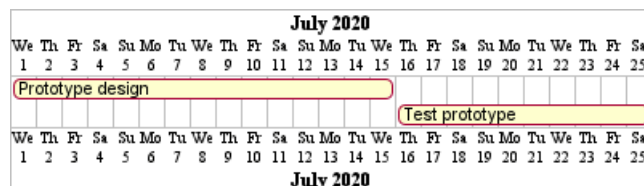


#### 15.1.2 Start

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
```



#### 15.1.3 End

Their ending are defined using the end verb:

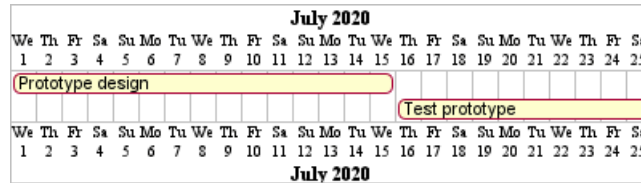
```

@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

```



### 15.1.4 Start/End

It is possible to define both absolutely, by specifying dates:

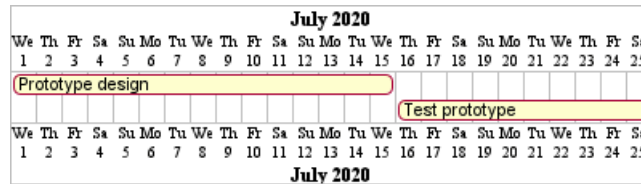
```

@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

```



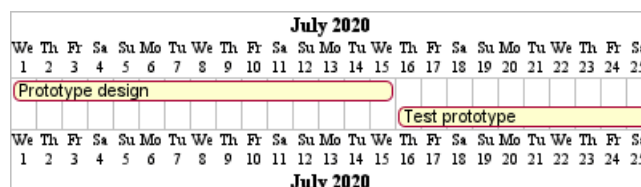
## 15.2 One-line declaration (with the and conjunction)

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

```



## 15.3 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

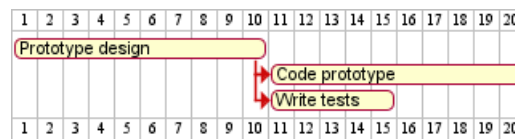
```



```

@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

```



## 15.4 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

```



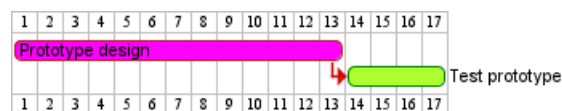
## 15.5 Customize colors

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

```



## 15.6 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

```



## 15.7 Milestone

You can define Milestones using the happen verb.

### 15.7.1 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

```

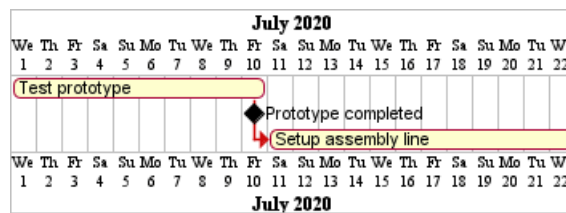


### 15.7.2 Absolute milestone (use of fixed date)

```

@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

```



### 15.7.3 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

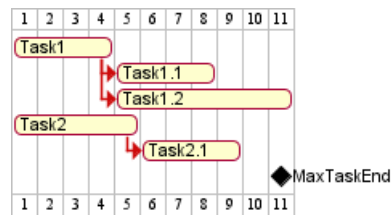
[MaxTaskEnd] happens at [Task1.1]'s end

```



```
[MaxTaskEnd] happens at [Task1.2]'s end
[MaxTaskEnd] happens at [Task2.1]'s end
```

```
@endgantt
```

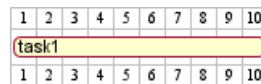


[Ref. QA-10764]

## 15.8 Hyperlinks

You can add hyperlinks to tasks.

```
@startgantt
[task1] lasts 10 days
[task1] links to [[http://plantuml.com]]
@endgantt
```



## 15.9 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
```



## 15.10 Coloring days

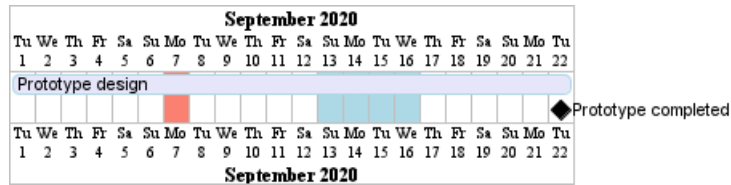
It is possible to add colors to some days.

```
@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
```





### 15.11 Changing scale

You can change scale for very long project, with one of those parameters:

- printscale
- gantt scale
- project scale

and one of the values:

- daily (by default)
- weekly
- monthly

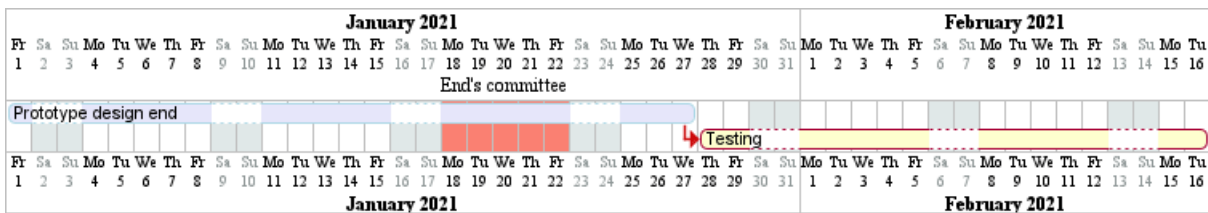
(See QA-11272, QA-9041 and QA-10948)

#### 15.11.1 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
```



#### 15.11.2 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
```

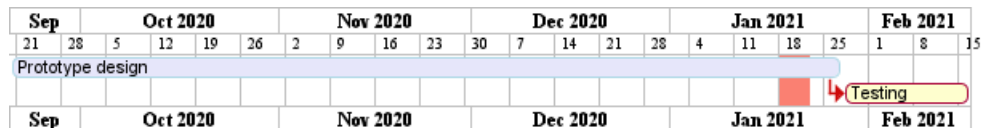
[TASK1]->[Testing]

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]

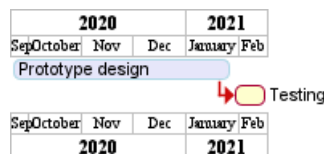
2021-01-18 to 2021-01-22 are named [End's committee]  
 2021-01-18 to 2021-01-22 are colored in salmon  
 @endgantt



### 15.11.3 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

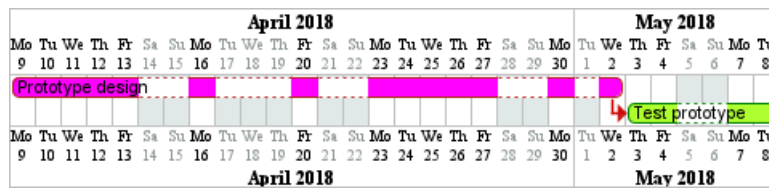


## 15.12 Close day

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
```



### 15.13 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 ->

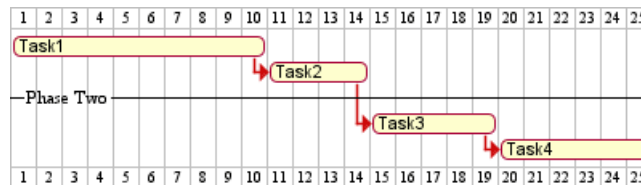
```
@startgantt
[Prototype design] lasts 14 days
[Build prototype] lasts 4 days
[Prepare test] lasts 6 days
[Prototype design] -> [Build prototype]
[Prototype design] -> [Prepare test]
@endgantt
```



## 15.14 Separator

You can use `--` to separate sets of tasks.

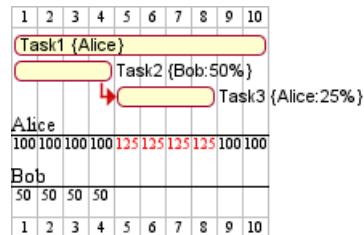
```
@startgantt
[Task1] lasts 10 days
then [Task2] lasts 4 days
-- Phase Two --
then [Task3] lasts 5 days
then [Task4] lasts 6 days
@endgantt
```



## 15.15 Working with resources

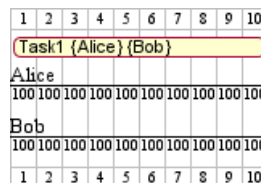
You can affect tasks on resources using the `on` keyword and brackets for resource name.

```
@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
```



June 2020														Jul						
Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	
19	20	21	22	23	24	25	26	27	28	29	30	1								
Task1 {Alice}																				
Alice																				
100	100	100	100	100						100	100	100	100	100						
Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	
19	20	21	22	23	24	25	26	27	28	29	30	1								
June 2020														Jul						

## 15.16 Complex example

It 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]'s end
```

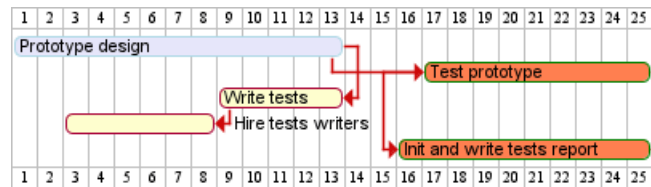
```
[Write tests] lasts 5 days and ends at [Prototype design]'s end
```

```
[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]'s end
```

```
@endgantt
```



## 15.17 Comments

As is mentioned on Common Commands page: `"` Everything that starts with simple quote ' is a comment.

You can also put comments on several lines using `/'` to start and `/'` to end. `"` (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
```



## 15.18 Using style

```
@startuml
```

```
<style>
```

```
ganttDiagram {
```

```
task {
```

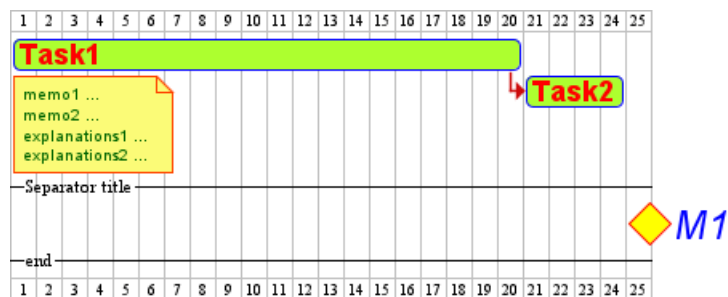




```

FontName Helvetica
FontColor red
FontSize 18
FontStyle bold
BackColor GreenYellow
LineColor blue
}
milestone {
FontColor blue
FontSize 25
FontStyle italic
BackColor yellow
LineColor red
}
note {
FontColor DarkGreen
FontSize 10
LineColor OrangeRed
}
}
</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

```



## 15.19 Add notes

```

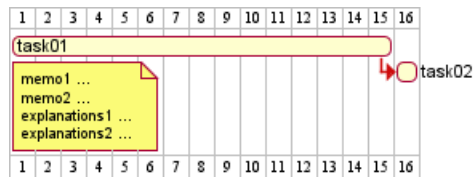
@startgantt
[task01] lasts 15 days
note bottom
  memo1 ...
  memo2 ...
  explanations1 ...
  explanations2 ...
end note

[task01] -> [task02]

```



```
@endgantt
```

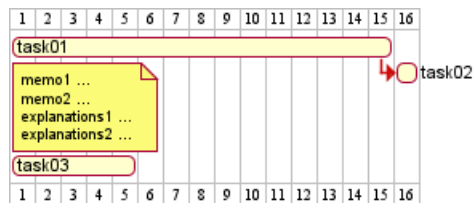


Example with overlap.

```
@startgantt
[task01] lasts 15 days
note bottom
  memo1 ...
  memo2 ...
  explanations1 ...
  explanations2 ...
end note
```

```
[task01] -> [task02]
[task03] lasts 5 days
```

```
@endgantt
```



```
@startgantt
```

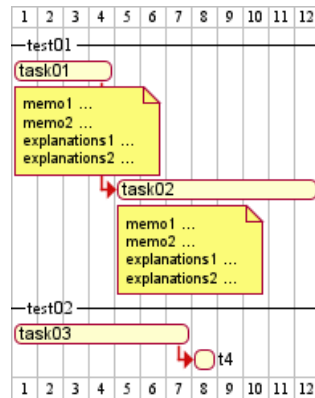
```
-- test01 --
```

```
[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

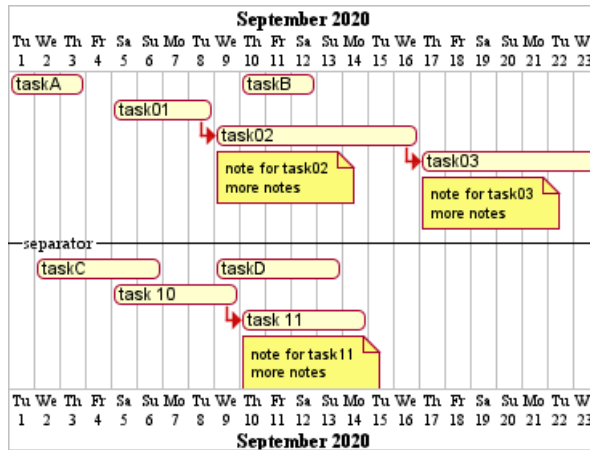
note for task11

more notes

end note

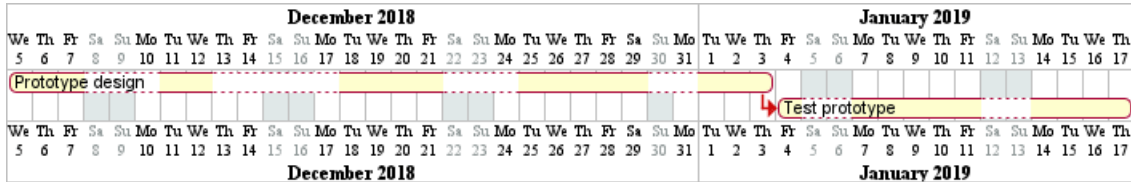
@endgantt





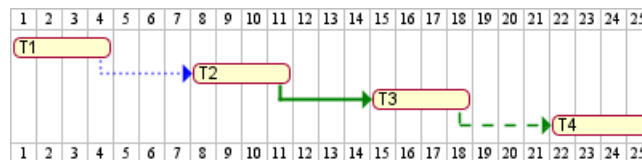
### 15.20 Pause tasks

```
@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
```



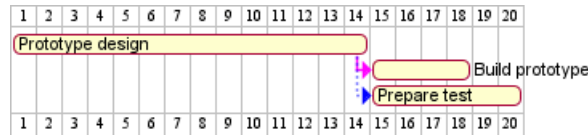
### 15.21 Change link colors

```
@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



### 15.22 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



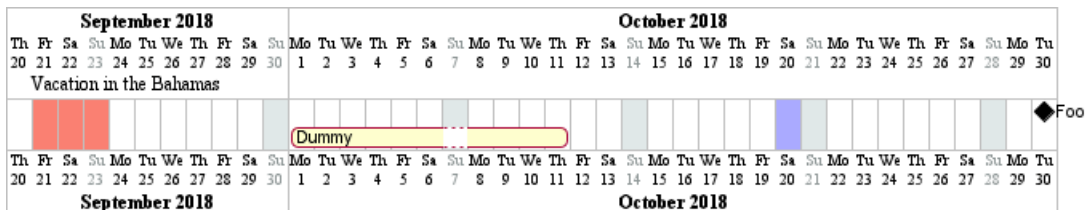
### 15.23 Highlight today

@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

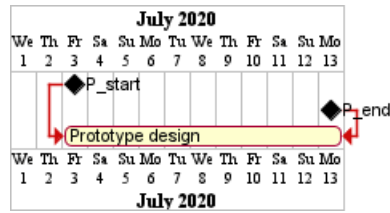


### 15.24 Task between two milestones

@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



## 15.25 Grammar and verbal form

Verbal form	Example
[T] starts	
[M] happens	

## 15.26 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
```

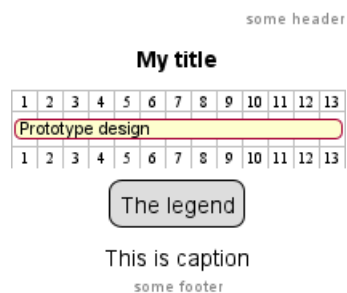
```
legend
```

```
The legend
```

```
end legend
```

```
caption This is caption
```

```
@enduml
```



(See also: *Common commands*)

## 15.27 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*)

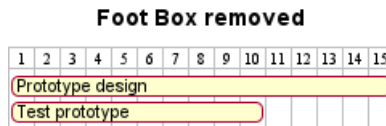
```
@startgantt
```

```
hide footbox
```

```
title Foot Box removed
```



```
[Prototype design] lasts 15 days
[Test prototype] lasts 10 days
@endgantt
```

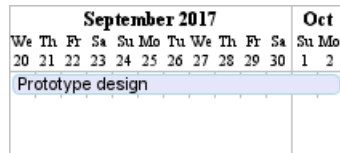


- daily scale

```
@startgantt
```

```
Project starts the 20th of september 2017
[Prototype design] as [TASK1] lasts 13 days
[TASK1] is colored in Lavender/LightBlue
```

```
hide footbox
@endgantt
```



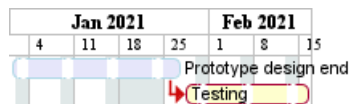
- weekly scale

```
@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  
@endgantt





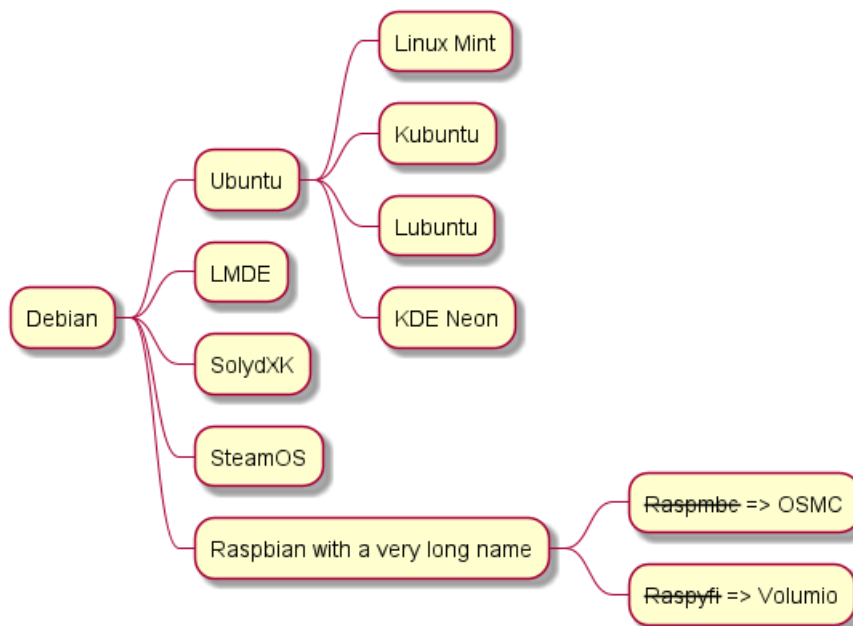
## 16 MindMap

MindMap diagram are still in beta: the syntax may change without notice.

### 16.1 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
```



### 16.2 Multilines

You can use : and ; to have multilines box.

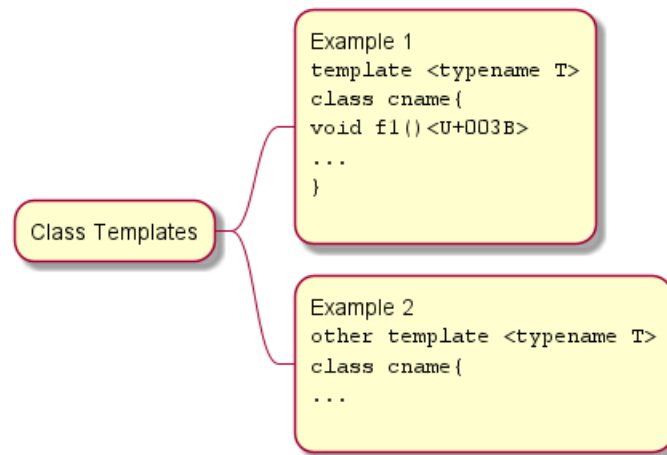
```
@startmindmap
* Class Templates
**Example 1
<code>
template <typename T>
class cname{
void f1()<U+003B>
...
}
```



```

</code>
;
**:Example 2
<code>
other template <typename T>
class cname{
...
</code>
;
@endmindmap

```



## 16.3 Colors

It is possible to change node color.

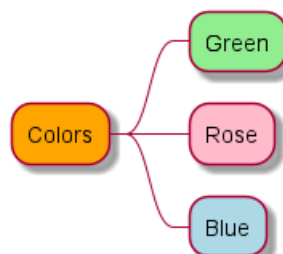
### 16.3.1 With inline color

- OrgMode syntax mindmap

```

@startmindmap
* [#Orange] Colors
** [#lightgreen] Green
** [#FFBCC] Rose
** [#lightblue] Blue
@endmindmap

```



- Markdown syntax mindmap

```

@startmindmap
* [#Orange] root node
  * [#lightgreen] some first level node
    * [#FFBCC] second level node

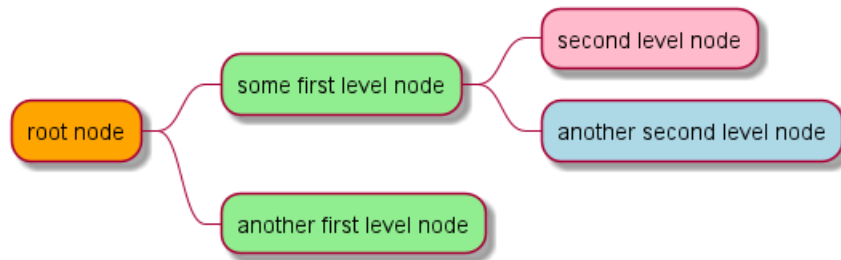
```



```

*[#lightblue] another second level node
*[#lightgreen] another first level node
@endmindmap

```



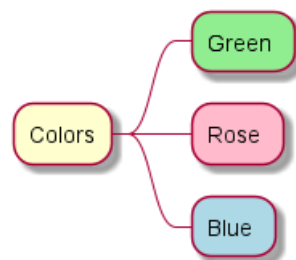
### 16.3.2 With style color

- OrgMode syntax mindmap

```

@startmindmap
<style>
mindmapDiagram {
  .green {
    BackgroundColor lightgreen
  }
  .rose {
    BackgroundColor #FFBCC
  }
  .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 #FFBCC
  }
}

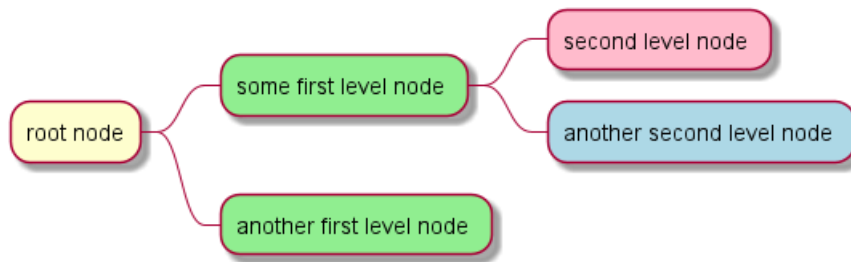
```



```

    .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

```



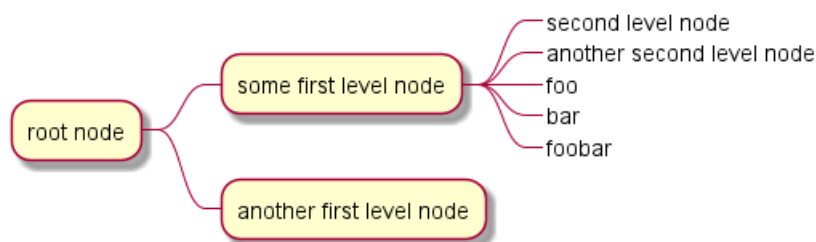
## 16.4 Removing box

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

```



## 16.5 Arithmetic notation

You can use the following notation to choose diagram side.

```

@startmindmap
+ OS
++ Ubuntu
+++ Linux Mint
+++ Kubuntu
+++ Lubuntu
+++ KDE Neon

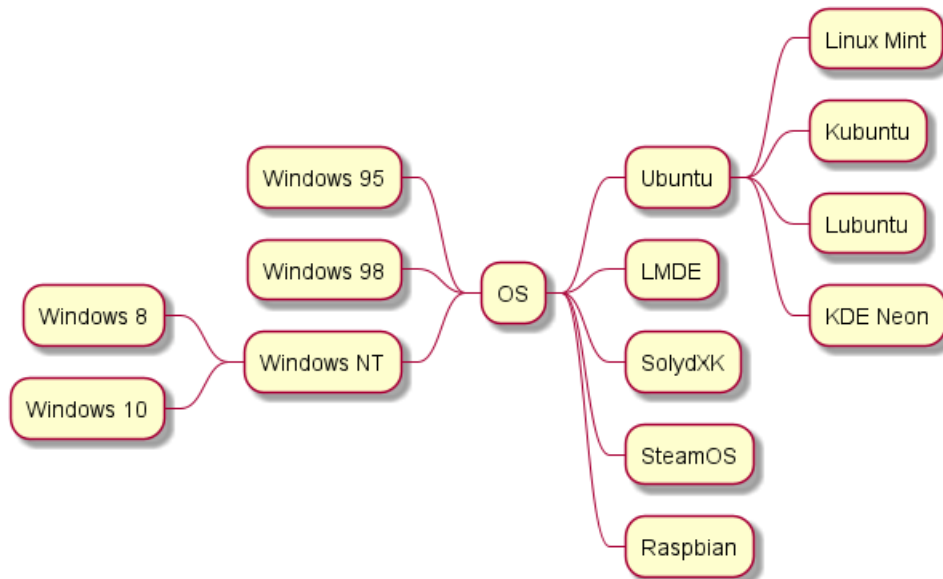
```



```

++ LMDE
++ SolydXK
++ SteamOS
++ Raspbian
-- Windows 95
-- Windows 98
-- Windows NT
--- Windows 8
--- Windows 10
@endmindmap

```



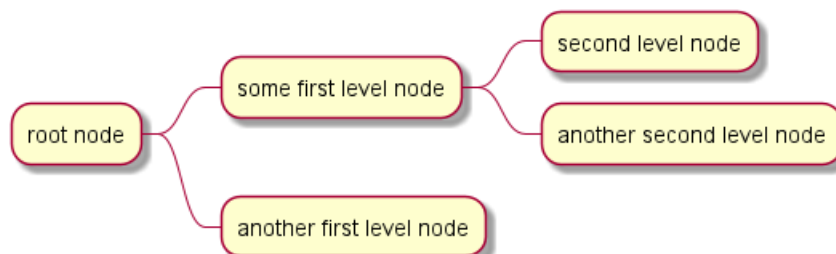
## 16.6 Markdown syntax

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

```



## 16.7 Changing style

```

@startmindmap
<style>

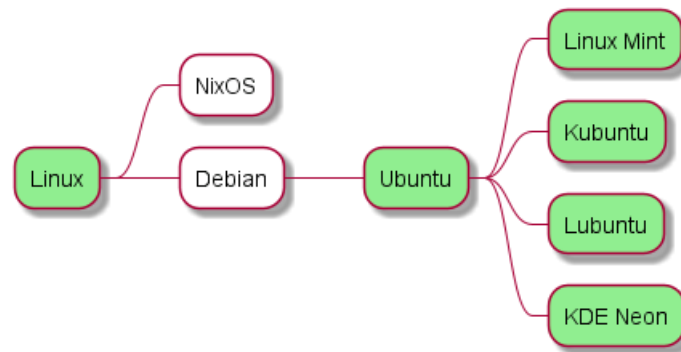
```



```

mindmapDiagram {
  node {
    BackgroundColor lightGreen
  }
  :depth(1) {
    BackGroundColor white
  }
}
</style>
* Linux
** NixOS
** Debian
*** Ubuntu
**** Linux Mint
**** Kubuntu
**** Lubuntu
**** KDE Neon
@endmindmap

```



## 16.8 Changing diagram direction

It is possible to use both sides of the diagram.

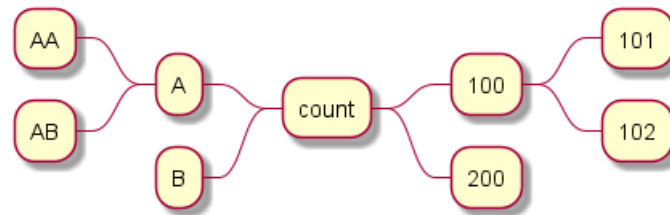
```

@startmindmap
* count
** 100
*** 101
*** 102
** 200

left side

** A
*** AA
*** AB
** B
@endmindmap

```



## 16.9 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
  
```

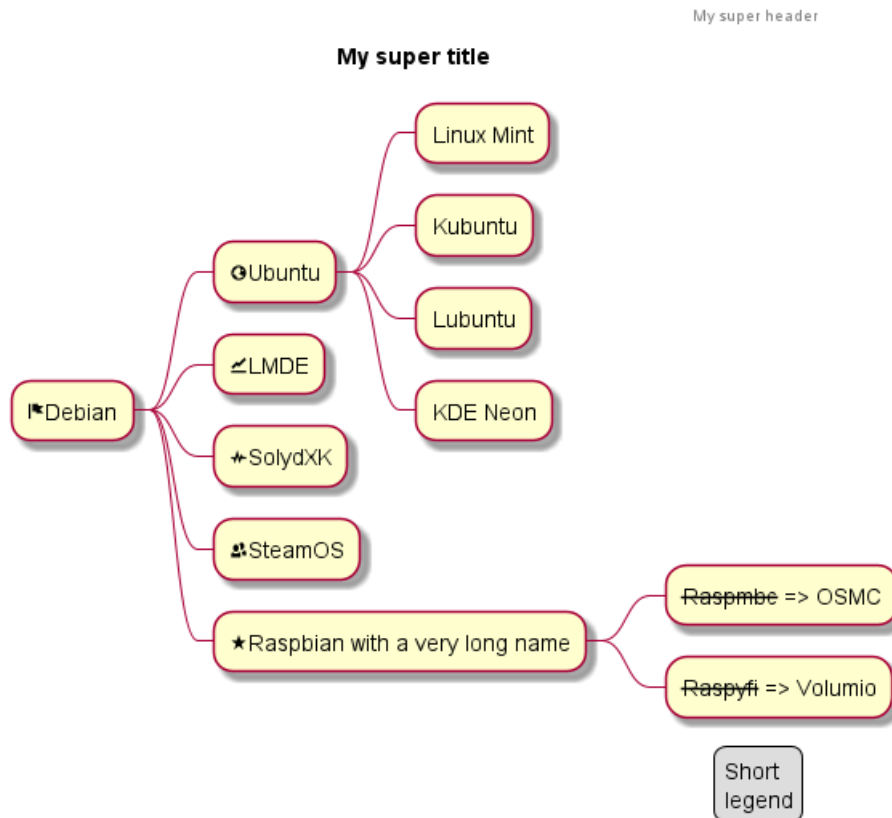


figure 1  
My super footer

## 16.10 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
}

```





```

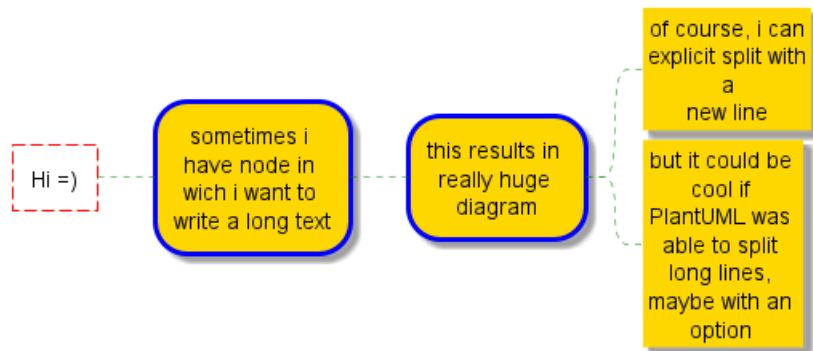
leafNode {
  LineColor gold
  RoundCorner 0
  Padding 3
}

arrow {
  LineStyle 4
  LineThickness 0.5
  LineColor green
}
</style>

* Hi =)
** sometimes i have node in wich 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

```



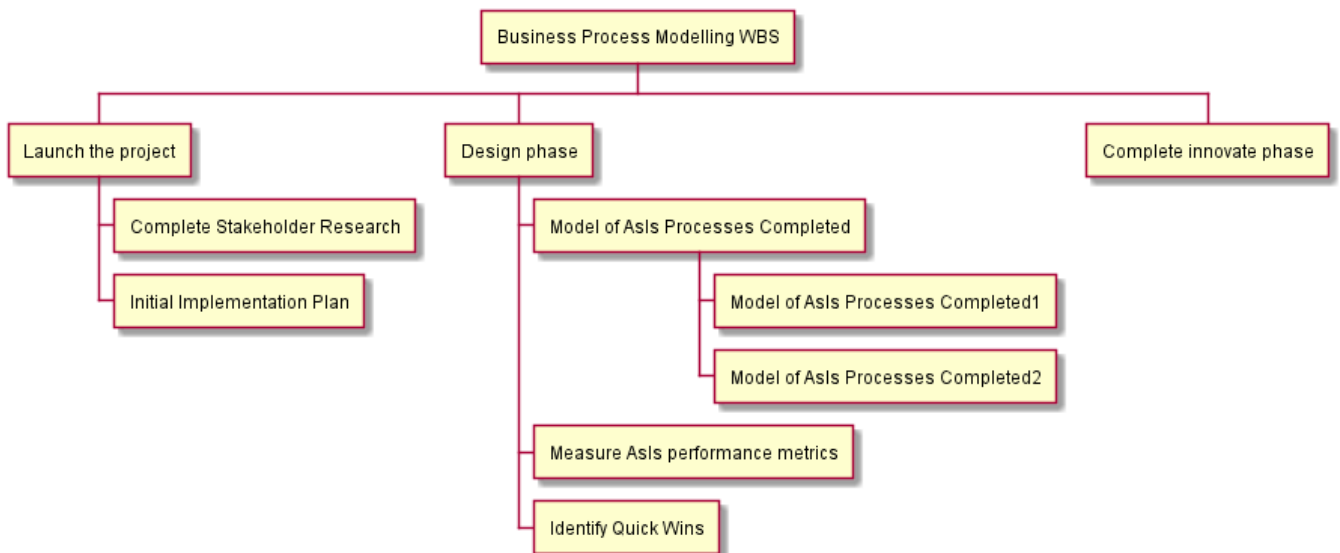
## 17 Work Breakdown Structure (WBS)

WBS diagram are still in beta: the syntax may change without notice.

### 17.1 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
```

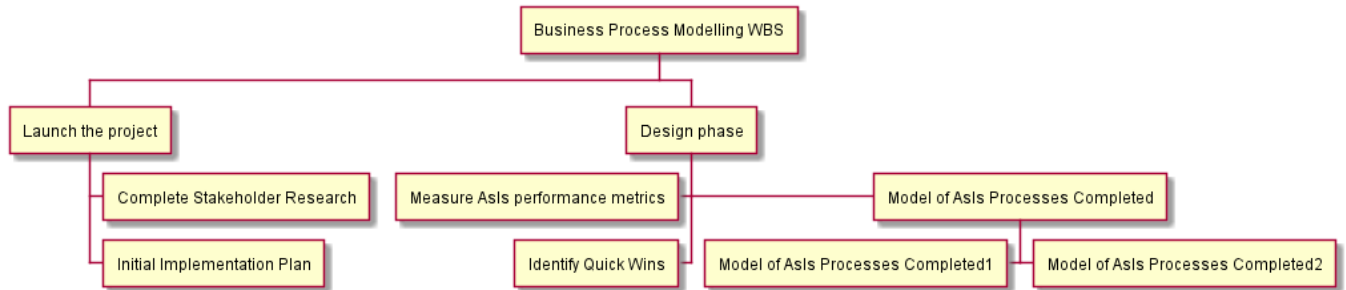


### 17.2 Change direction

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
```



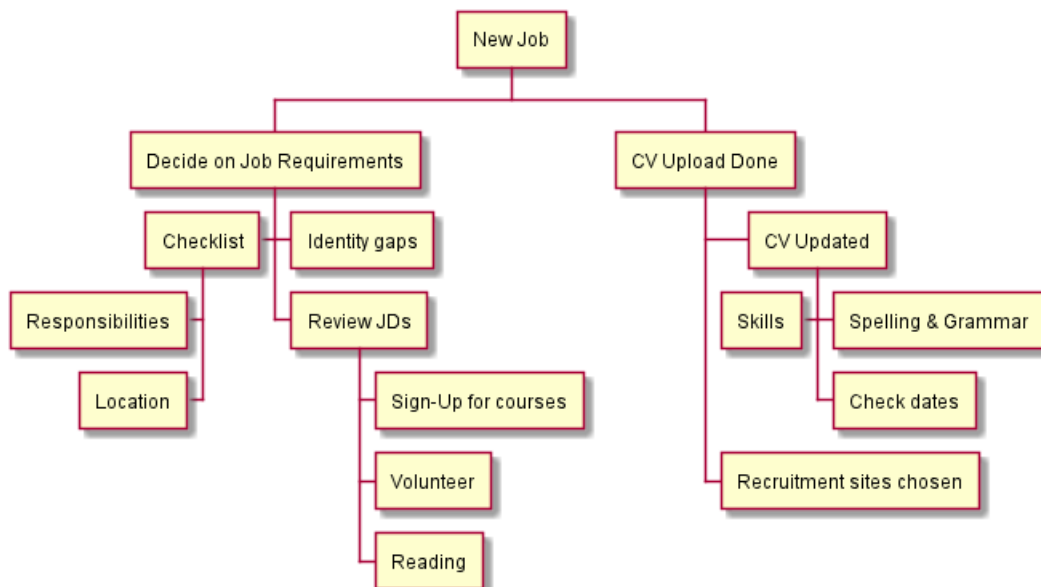


### 17.3 Arithmetic notation

You can use the following notation to choose diagram side.

```

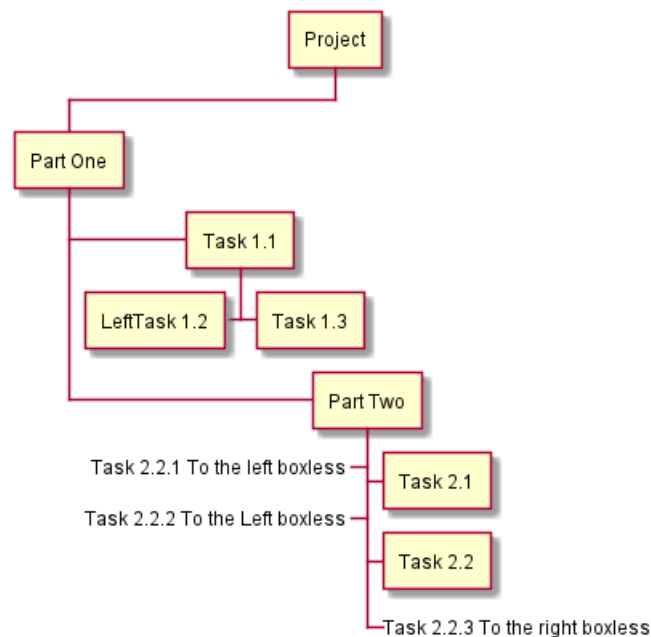
@startwbs
+ New Job
++ Decide on Job Requirements
+++ Identity gaps
+++ Review JDs
++++ Sign-Up for courses
++++ Volunteer
++++ Reading
++- Checklist
+++- Responsibilities
+++- Location
++ CV Upload Done
+++ CV Updated
++++ Spelling & Grammar
++++ Check dates
---- Skills
+++ Recruitment sites chosen
@endwbs
    
```



## 17.4 Removing box

You can use underscore \_ to remove box drawing.

```
@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
```

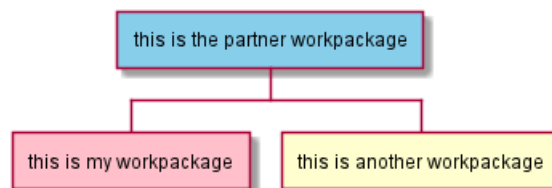


## 17.5 Colors (with inline or style color)

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
```



[Ref. QA-12374, only from v1.2020.20]

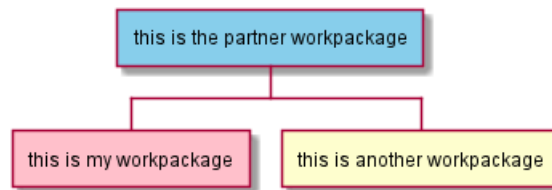


- with style color

```

@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

```



## 17.6 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 red
  }
  arrow {
    :depth(2) {
      // Targetting only connector between Mexico-Chihuahua and USA-Texas
      LineColor blue
      LineStyle 4
      LineThickness .5
    }
  }
}
node {
  :depth(2) {
    LineStyle 2
  }
}

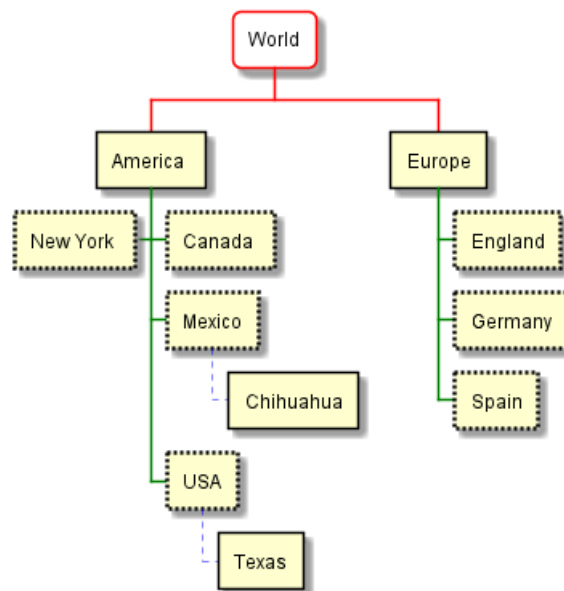
```



```

    LineThickness 2.5
  }
}
}
</style>
* World
** America
*** Canada
*** Mexico
**** Chihuahua
*** USA
**** Texas
***< New York
** Europe
*** England
*** Germany
*** Spain
@endwbs

```



## 17.7 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 =)

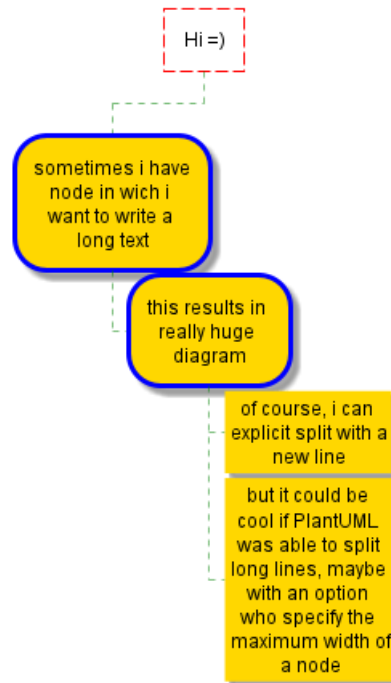
\*\* sometimes i have node in wich i want to write a long text

\*\*\* 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 who specify the ma

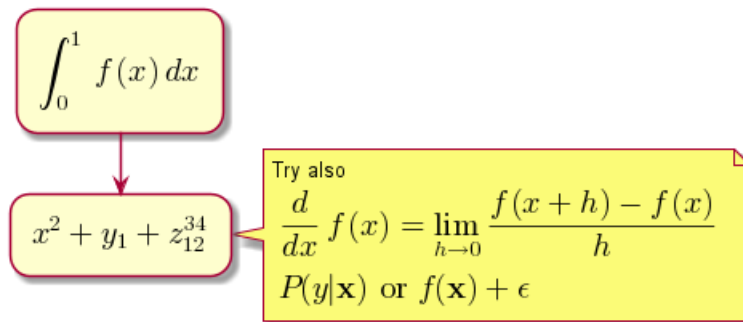
@endwbs



## 18 Maths

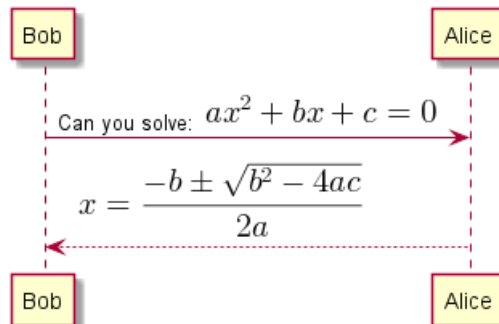
You can use AsciiMath or JLaTeXMath notation within PlantUML:

```
@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>
<latex>P(y|\mathbf{x}) \ \mbox{ or } \ f(\mathbf{x}) + \epsilon</latex>
end note
@enduml
```



or:

```
@startuml
Bob -> Alice : Can you solve: <math>ax^2 + bx + c = 0</math>
Alice --> Bob: <math>x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}</math>
@enduml
```



### 18.1 Standalone diagram

You can also use @startmath/@endmath to create standalone AsciiMath formula.

```
@startmath
f(t) = (a_0)/2 + \sum_{n=1}^{\infty} a_n \cos((n\pi t)/L) + \sum_{n=1}^{\infty} b_n \sin((n\pi t)/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
```





$$\sum_{i=0}^{n-1} (a_i + b_i^2)$$

## 18.2 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.



## 19 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.

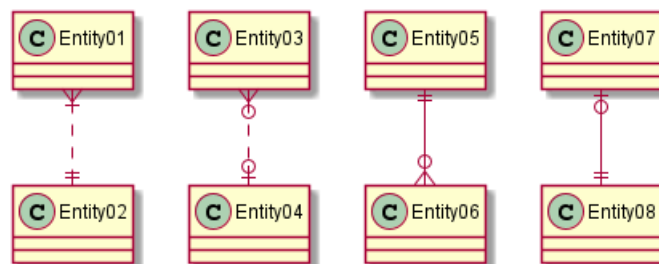
Otherwise, the syntax for drawing diagrams is the same as for class diagrams. All other features of class diagrams are also supported.

### 19.1 Information Engineering Relations

Type	Symbol
Zero or One	o--
Exactly One	--
Zero or Many	}o--
One or Many	} --

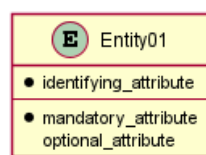
Examples:

```
@startuml
Entity01 }|..|| Entity02
Entity03 }o..o| Entity04
Entity05 ||--o{ Entity06
Entity07 |o--|| Entity08
@enduml
```



### 19.2 Entities

```
@startuml
entity Entity01 {
  * identifying_attribute
  --
  * mandatory_attribute
  optional_attribute
}
@enduml
```

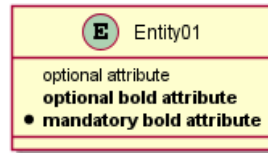


Again, this is the normal class diagram syntax (aside from use of `entity` instead of `class`). Anything that you can do in a class diagram can be done here.



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
```



### 19.3 Complete Example

```
@startuml

' hide the spot
hide circle

' avoid problems with angled crows feet
skinparam linetype ortho

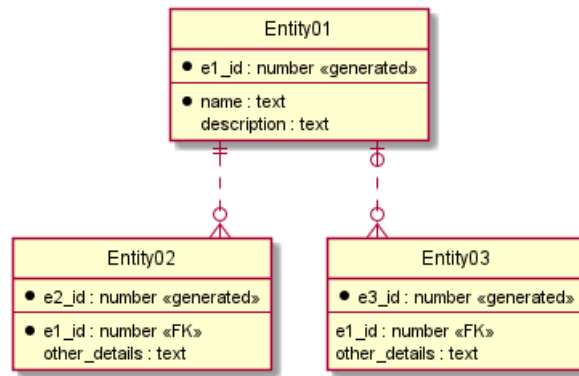
entity "Entity01" as e01 {
  *e1_id : number <<generated>>
  --
  *name : text
  description : text
}

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.

## 20 Common commands

### 20.1 Comments

Everything that starts with `simple quote ' is a comment.`

You can also put comments on several lines using `/' to start and '/ to end.`

### 20.2 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 for title, it is possible to define a header or a footer on several lines.

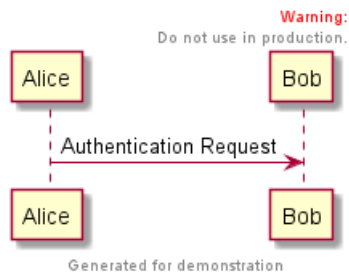
It is also possible to put some HTML into the header or footer.

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

```
header
<font color=red>Warning:</font>
Do not use in production.
endheader
```

```
center footer Generated for demonstration
```

```
@enduml
```



### 20.3 Zoom

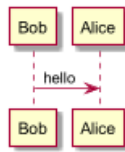
You can use the `scale` command to zoom the generated image.

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
```



## 20.4 Title

The title keywords is used to put a title. You can add newline using `\n` in the title description.

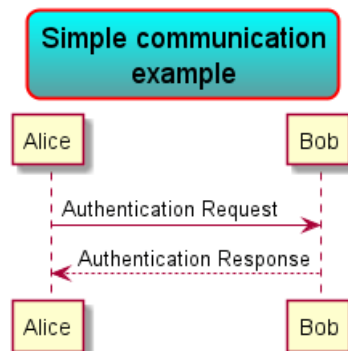
Some skinparam settings are available to put borders on the title.

```
@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.

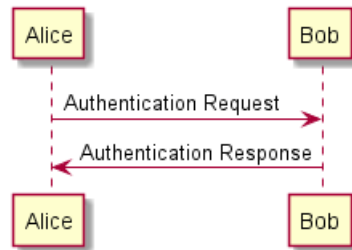
```
@startuml
title
<u>Simple</u> communication example
on <i>several</i> lines and using <back:cadetblue>creole tags</back>
end title
```

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

```
@enduml
```



### Simple communication example on several lines and using creole tags

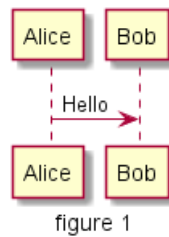


## 20.5 Caption

There is also a `caption` keyword to put a caption under the diagram.

```

@startuml
caption figure 1
Alice -> Bob: Hello
@enduml
  
```



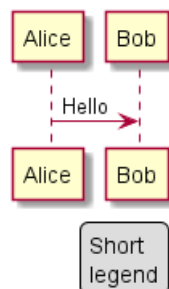
## 20.6 Legend the diagram

The `legend` and `end legend` keywords is 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
  
```



```

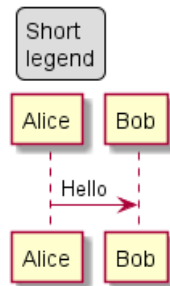
@startuml
Alice -> Bob : Hello
legend top left
  
```



```

Short
legend
endlegend
@enduml

```



## 20.7 Appendice: Examples on all diagram

### 20.7.1 Activity

```

@startuml
header some header

footer some footer

title My title

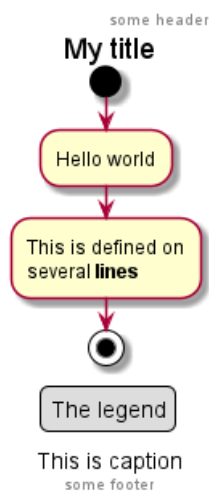
caption This is caption

legend
The legend
end legend

start
:Hello world;
:This is defined on
several lines;
stop

@enduml

```





### 20.7.2 Archimate

```

@startuml
header some header

footer some footer

title My title

caption This is caption

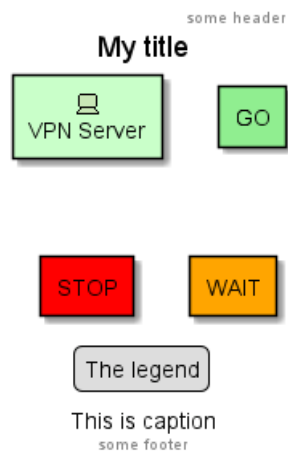
legend
The legend
end legend

archimate #Technology "VPN Server" as vpnServerA <<technology-device>>

rectangle GO #lightgreen
rectangle STOP #red
rectangle WAIT #orange

@enduml

```



### 20.7.3 Class

```

@startuml
header some header

footer some footer

title My title

caption This is caption

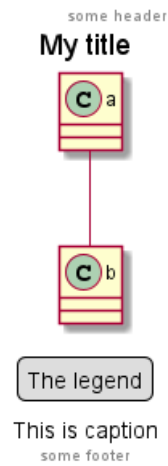
legend
The legend
end legend

a -- b

@enduml

```





#### 20.7.4 Component, Deployment, Use-Case

```

@startuml
header some header

footer some footer

title My title

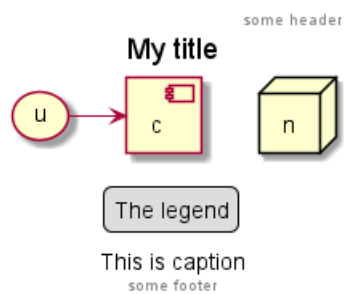
caption This is caption

legend
The legend
end legend

node n
(u) -> [c]

@enduml

```



#### 20.7.5 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
```



**TODO:** DONE [(Header, footer) corrected on V1.2020.18]

### 20.7.6 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
```



### 20.7.7 MindMap

```
@startmindmap
header some header

footer some footer
```



```

title My title

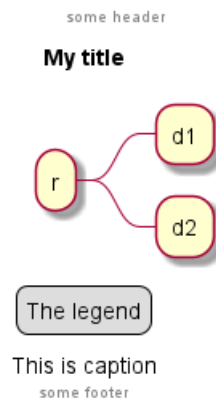
caption This is caption

legend
The legend
end legend

* r
** d1
** d2

@endmindmap

```



### 20.7.8 Network (nwdiag)

```

@startuml
header some header

footer some footer

title My title

caption This is caption

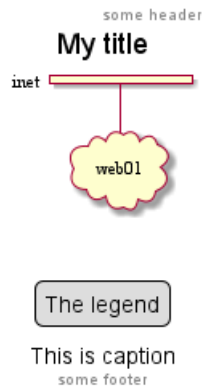
legend
The legend
end legend

nwdiag {
  network inet {
    web01 [shape = cloud]
  }
}

@enduml

```





### 20.7.9 Sequence

```

@startuml
header some header

footer some footer

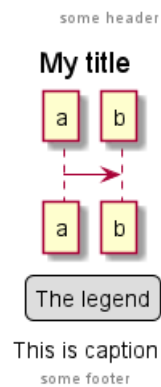
title My title

caption This is caption

legend
The legend
end legend

a->b
@enduml

```



### 20.7.10 State

```

@startuml
header some header

footer some footer

title My title

caption This is caption

legend
The legend

```



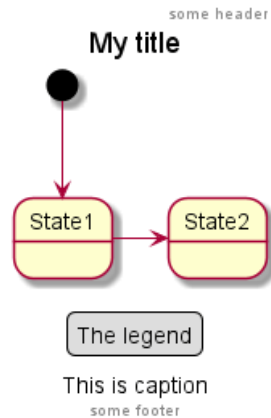
```

end legend

[*] --> State1
State1 -> State2

@enduml

```



### 20.7.11 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

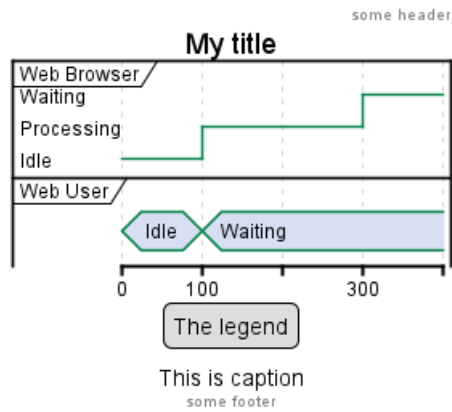
@100
WU is Waiting
WB is Processing

@300
WB is Waiting

@enduml

```





### 20.7.12 Work Breakdown Structure (WBS)

```

@startwbs
header some header

footer some footer

title My title

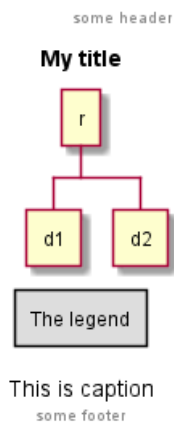
caption This is caption

legend
The legend
end legend

* r
** d1
** d2

@endwbs

```



**TODO:** DONE [Corrected on V1.2020.17]

### 20.7.13 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]

## 20.8 Appendice: 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>

```

### 20.8.1 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

```



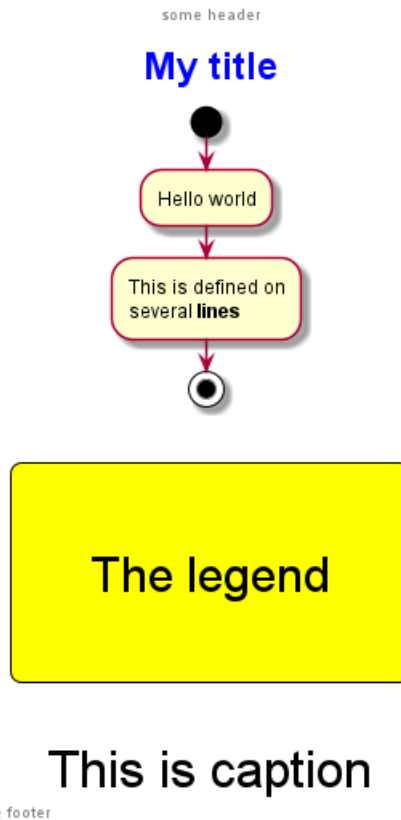
```

legend
The legend
end legend

start
:Hello world;
:This is defined on
several **lines**;
stop

@enduml

```



### 20.8.2 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

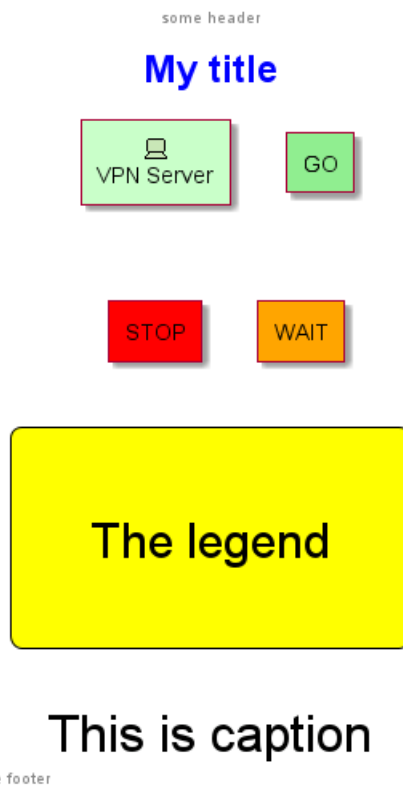
legend
The legend
end legend

archimate #Technology "VPN Server" as vpnServerA <<technology-device>>

rectangle GO #lightgreen
rectangle STOP #red
rectangle WAIT #orange

@enduml

```



**20.8.3 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

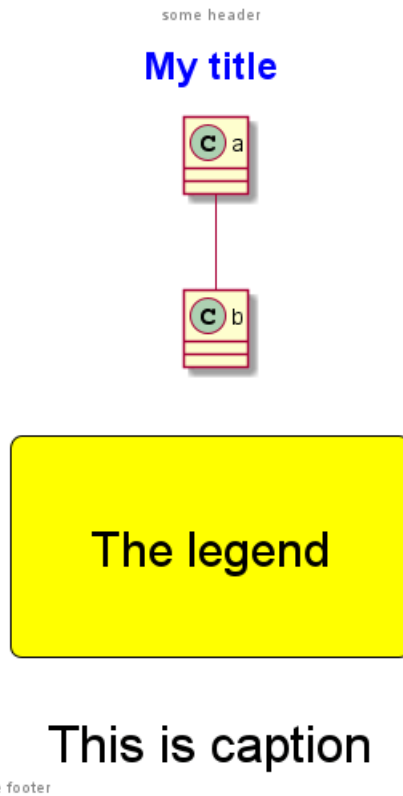
title My title

caption This is caption

legend
The legend
end legend

a -- b

@enduml
```



#### 20.8.4 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

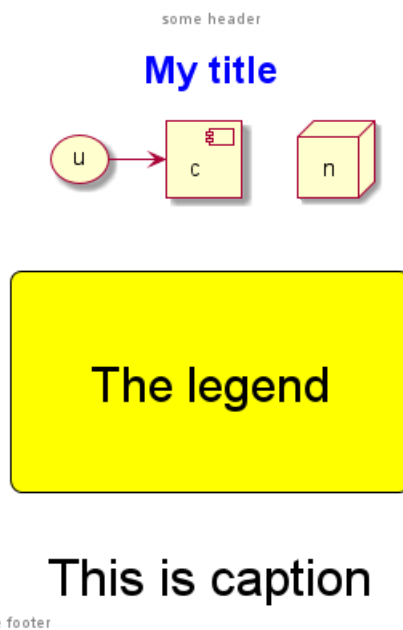
caption This is caption

legend
The legend
end legend

node n
(u) -> [c]

@enduml

```



### 20.8.5 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

```



### 20.8.6 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

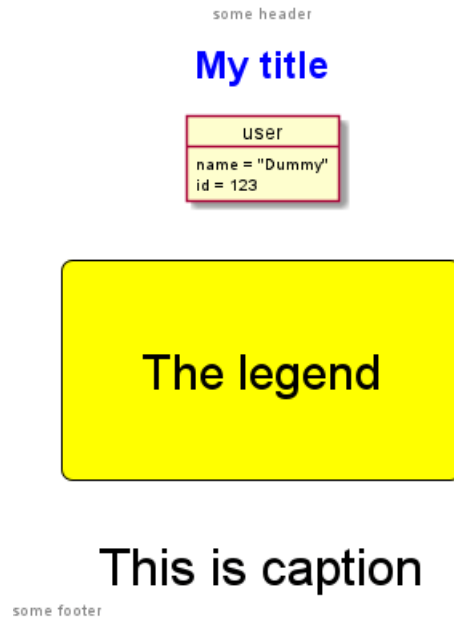
caption This is caption

```
legend  
The legend  
end legend
```

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

@enduml





### 20.8.7 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

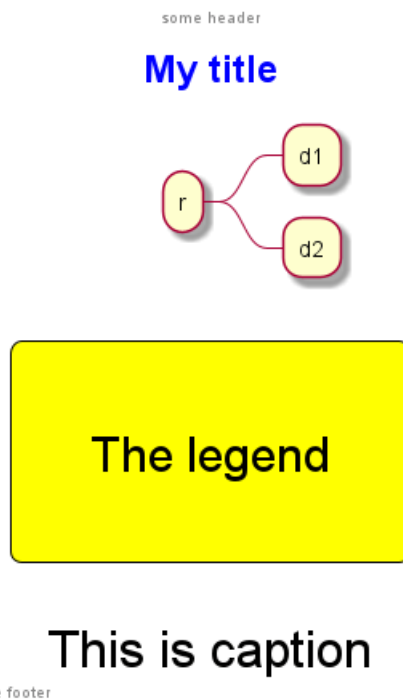
caption This is caption

legend
The legend
end legend

* r
** d1
** d2

@endmindmap

```



### 20.8.8 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

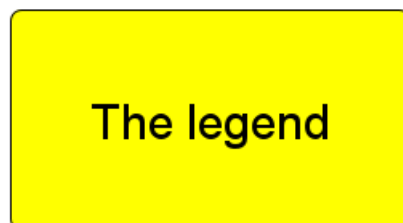
caption This is caption

legend
The legend
end legend

nwdiag {
  network inet {
    web01 [shape = cloud]
  }
}

@enduml

```



**This is caption**

some footer

**20.8.9 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

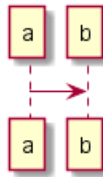
caption This is caption

legend
The legend
end legend

a->b
@enduml
```

some header

**My title**



The legend

This is caption

some footer

#### 20.8.10 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

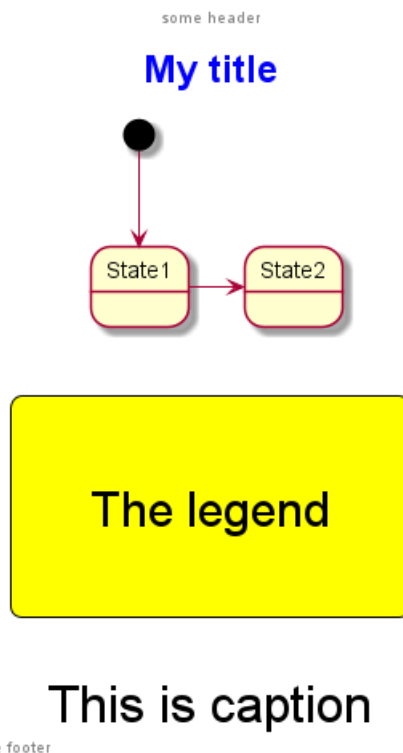
caption This is caption

legend
The legend
end legend

[*] --> State1
State1 -> State2

@enduml

```



### 20.8.11 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
```

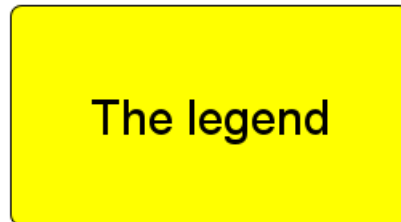
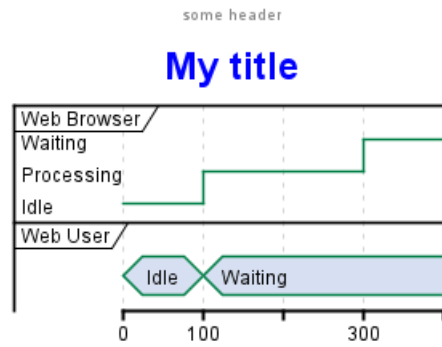
```
robust "Web Browser" as WB  
concise "Web User" as WU
```

```
@0  
WU is Idle  
WB is Idle
```

```
@100  
WU is Waiting  
WB is Processing
```

```
@300  
WB is Waiting
```

```
@enduml
```



This is caption

some footer

### 20.8.12 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

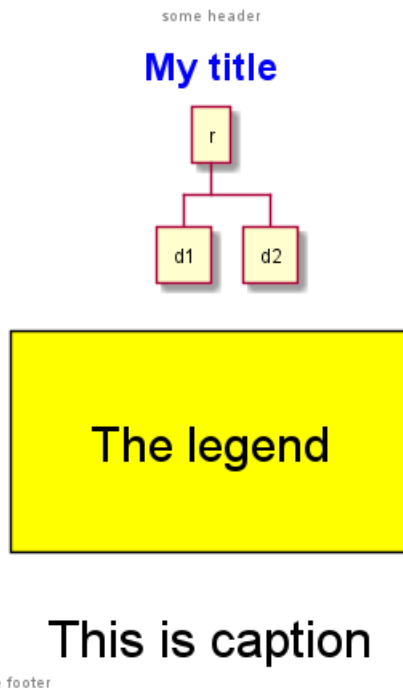
caption This is caption

legend
The legend
end legend

* r
** d1
** d2

@endwbs

```



### 20.8.13 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

```



## 21 Creole

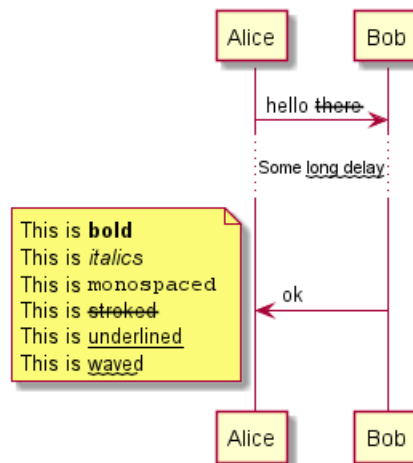
A light Creole engine has been integrated into PlantUML to have a standardized way of defining text style.

All diagrams are now supporting this syntax.

Note that ascending compatibility with HTML syntax is preserved.

### 21.1 Emphasized text

```
@startuml
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
@enduml
```



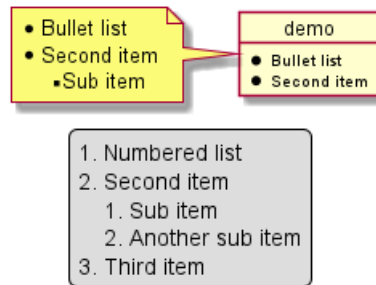
### 21.2 List

```
@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
end
```



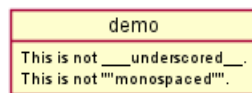
```
# Third item
end legend
@enduml
```



## 21.3 Escape character

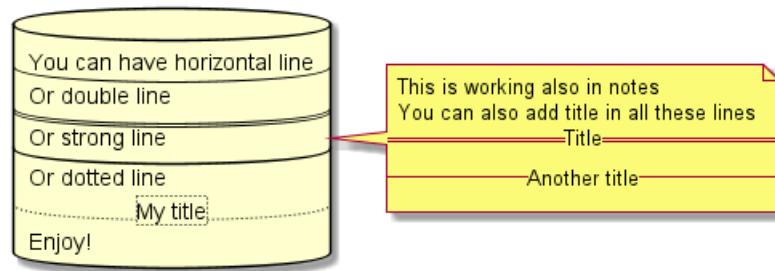
You can use the tilde ~ to escape special creole characters.

```
@startuml
object demo {
  This is not ~__underscored__.
  This is not ~'"monospaced"'".
}
@enduml
```



## 21.4 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
```



## 21.5 Headings

```
@startuml
usecase UC1 as "
= Extra-large heading
Some text
== Large heading
Other text
=== Medium heading
Information
...
==== Small heading"
@enduml
```



## 21.6 Legacy HTML

Some HTML tags are also working:

- `<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
- `<color:#AAAAAA>` or `<color:[[color|colorName]]>`
- `<back:#AAAAAA>` or `<back:[[color|colorName]]>` for background color
- `<size:nn>` to change font size
- `<img:file>`: the file must be accessible by the filesystem
- `<img:http://plantuml.com/logo3.png>`: the URL must be available from the Internet

```
@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>
```



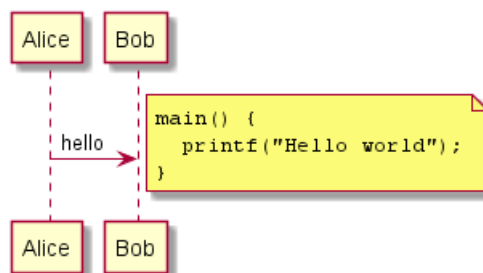
```
* You use <u:red>color</u> <s:green>in HTML</s> <w:#0000FF>tag</w>
----
* Use image : <img:http://plantuml.com/logo3.png>
;
@enduml
```



## 21.7 Code

You can use `<code>` if you put some language code in your diagram.

```
@startuml
Alice -> Bob : hello
note right
<code>
main() {
    printf("Hello world");
}
</code>
end note
@enduml
```



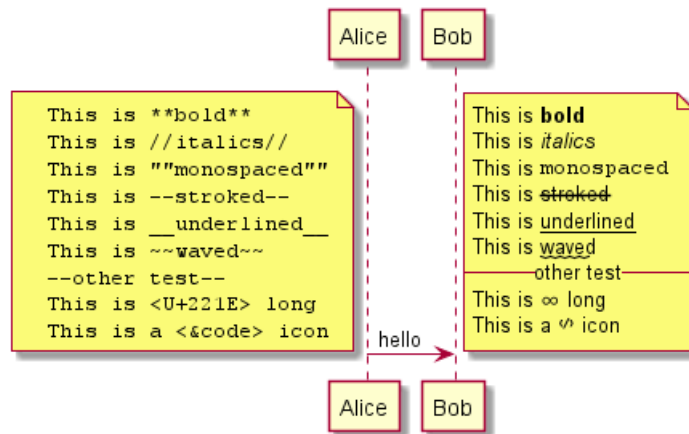
```
@startuml
Alice -> Bob : hello
note left
<code>
    This is bold
    This is italics
    This is "monospaced"
    This is --stroked--
    This is underlined
    This is ~waved~
    --other test--
    This is <U+221E> long
</code>
end note
@enduml
```



```

This is a <&code> icon
</code>
end note
note right
  This is **bold**
  This is //italics//
  This is "monospaced"
  This is --stroked--
  This is __underlined__
  This is ~waved~
  --other test--
  This is <U+221E> long
  This is a <&code> icon
end note
@enduml

```



## 21.8 Table

### 21.8.1 Build a table

It is possible to build table, with | separator.

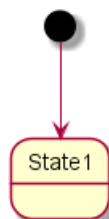
```

@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

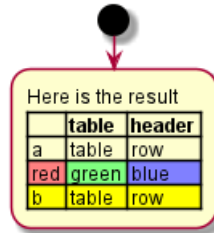
	table	header
a	table	row
b	table	row



### 21.8.2 Add color on cells or lines

You can specify background colors for cells and lines.

```
@startuml
start
:Here is the result
|= |= table |= header |
| a | table | row |
|<#FF8080> red |<#80FF80> green |<#8080FF> blue |
<#yellow>| b | table | row |;
@enduml
```



### 21.8.3 Add color on border

You can also specify background colors and colors for border.

```
@startuml
title
<#lightblue,#red>|= Step |= Date |= Name |= Status |= Link |
<#lightgreen>| 1.1 | TBD | plantuml news |<#Navy><color:OrangeRed><b> Unknown | [[https://plantuml.c
end title
@enduml
```

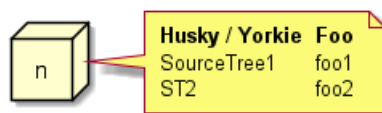
Step	Date	Name	Status	Link
1.1	TBD	plantuml news	Unknown	plantuml news

[Ref. QA-7184]

### 21.8.4 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]





### 21.8.5 Bold header or not

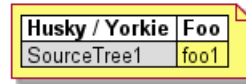
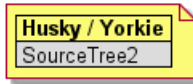
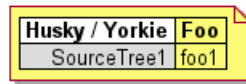
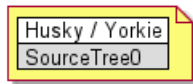
You can have a bold header or not.

```
@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]

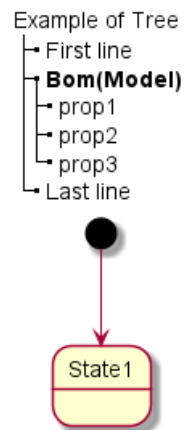
## 21.9 Tree

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
```

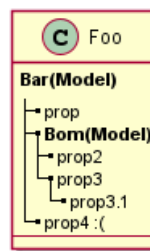




On Class diagram:

```

@startuml
class Foo{
**Bar(Model)**
|_ prop
|_ **Bom(Model)**
|_ prop2
|_ prop3
|_ prop3.1
|_ prop4 :(
--
}
@enduml
  
```



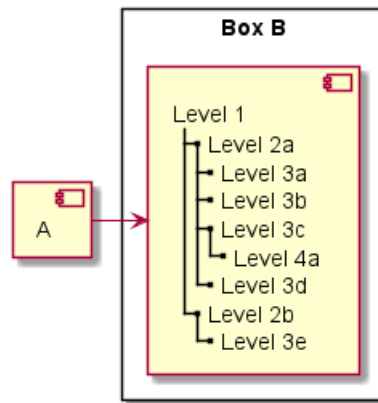
[Ref. QA-3448]

And on component or deployment diagram:

```

@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]

## 21.10 Special characters

It's possible to use any unicode characters with `&#` syntax or `<U+XXXX>`

```

@startuml
usecase foo as "this is &#8734; long"
usecase bar as "this is also <U+221E> long"
@enduml
  
```



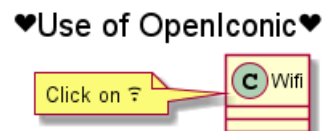
## 21.11 OpenIconic

OpenIconic is an 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>`.

```

@startuml
title: <size:20><&heart>Use of OpenIconic<&heart></size>
class Wifi
note left
  Click on <&wifi>
end note
@enduml
  
```



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
<i>Credit to</i>	🔗 bluetooth	☁️ cloudy	⏏ expand-down	🔑 key	📄 paperclip	☀ sun
<a href="https://useiconic.com/open">https://useiconic.com/open</a>	<b>B</b> bold	📄 code	⏪ expand-left	💻 laptop	✎ pencil	📱 tablet
⇒ account-login	⚡ bolt	🧠 cog	⏩ expand-right	📁 layers	👥 people	🏷 tag
⇒ account-logout	📖 book	⏴ collapse-down	⏴ expand-up	💡 lightbulb	👤 person	🏷 tags
↶ action-redo	🔖 bookmark	⏪ collapse-left	🔗 external-link	🔗 link-broken	📞 phone	🎯 target
↶ action-undo	📦 box	⏩ collapse-right	👁 eye	🔗 link-intact	📊 pie-chart	🗂 task
≡ align-center	👛 briefcase	⏴ collapse-up	👉 eyedropper	≡ list-rich	📌 pin	🖥 terminal
≡ align-left	🇬🇧 british-pound	⌘ command	📁 file	≡ list	🕒 play-circle	T text
≡ align-right	🌐 browser	📐 comment-square	🔥 fire	📍 location	+ plus	👎 thumb-down
🔍 aperture	🖌 brush	🧭 compass	🚩 flag	🔒 lock-locked	🔌 power-standby	👍 thumb-up
↓ arrow-bottom	🐛 bug	🕒 contrast	⚡ flash	🔓 lock-unlocked	🖨 print	⌚ timer
🕒 arrow-circle-bottom	📣 bullhorn	≡ copywriting	📁 folder	🔄 loop-circular	📁 project	⇄ transfer
🕒 arrow-circle-left	📅 calendar	💳 credit-card	🍴 fork	📐 loop-square	+ pulse	🗑 trash
🕒 arrow-circle-right	📷 camera-slr	📏 crop	🖱 fullscreen-enter	🔄 loop	🧩 puzzle-piece	📄 underline
🕒 arrow-circle-top	⏴ caret-bottom	📊 dashboard	🖱 fullscreen-exit	🔍 magnifying-glass	? question-mark	⏴ vertical-align-bottom
← arrow-left	⏪ caret-left	⬇ data-transfer-download	🌐 globe	📍 map-marker	☔ rain	≡ vertical-align-center
→ arrow-right	⏩ caret-right	⬆ data-transfer-upload	📊 graph	🗺 map	✳ random	≡ vertical-align-top
↓ arrow-thick-bottom	⤴ caret-top	🗑 delete	📊 grid-four-up	⏸ media-pause	🔄 reload	📹 video
← arrow-thick-left	⤵ caret-top	☎ dial	📊 grid-three-up	▶ media-play	↔ resize-both	🔊 volume-high
→ arrow-thick-right	💬 chat	📄 document	📊 grid-two-up	⏮ media-record	↕ resize-height	🔊 volume-low
↑ arrow-thick-top	✓ check	\$ dollar	📁 hard-drive	⏮ media-skip-backward	↔ resize-width	⏻ volume-off
↑ arrow-top	⏴ chevron-bottom	” double-quote-sans-left	📁 header	▶ media-skip-forward	📡 rss	⚠ warning
🔊 audio-spectrum	⏪ chevron-left	“ double-quote-sans-right	🎧 headphones	▶ media-step-backward	📡 rss-alt	📶 wifi
🔊 audio	⏩ chevron-right	” double-quote-serif-left	♥ heart	▶ media-step-forward	📄 script	🔧 wrench
† badge	⏴ chevron-top	” double-quote-serif-right	🏠 home	■ media-stop	📦 share-boxed	✖ x
📊 ban	🕒 circle-check	💧 droplet	🖼 image	🏥 medical-cross	➦ share	¥ yen
📊 bar-chart	🕒 circle-x	📶 eject	📧 inbox	≡ menu	🛡 shield	🔍 zoom-in
🛒 basket	📋 clipboard	🛗 elevator	∞ infinity	🎧 microphone	↑ signpost	🔍 zoom-out
🔋 battery-empty	⋯ ellipses	📄 ellipses	📄 info	➖ minus	📊 sort-ascending	
🔋 battery-full	🕒 clock	📁 envelope-closed	📄 italic	📺 monitor	📊 sort-descending	
🍷 beaker	☁ cloud-download	📁 envelope-open	≡ justify-center	🌙 moon	➦ move	
	☁ cloud-upload	€ euro	≡ justify-left	+ move		

## 21.12 Appendice: Examples of "Creole List" on all diagrams

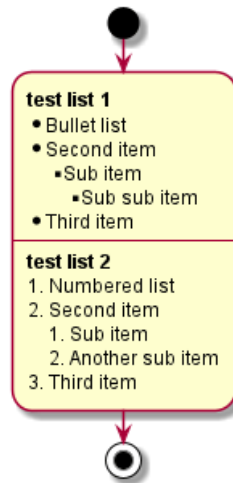
### 21.12.1 Activity

```

@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

```





### 21.12.2 Class

**TODO:** FIXME ☐

- *Sub item*
- *Sub sub item*

**TODO:** FIXME

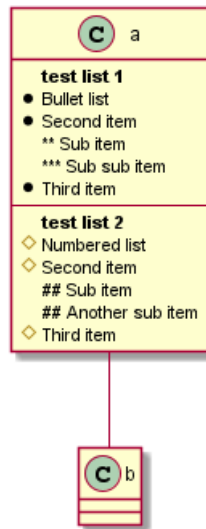
@startuml

```
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





### 21.12.3 Component, Deployment, Use-Case

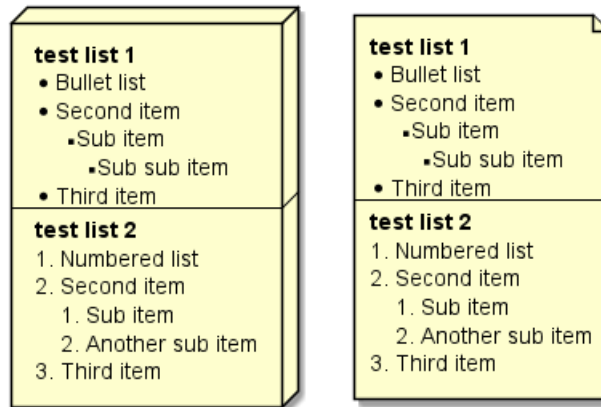
```

@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 on V1.2020.18]

#### 21.12.4 Gantt project planning

N/A

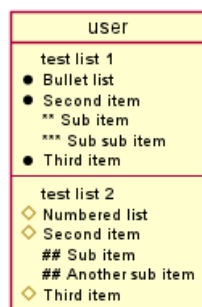
#### 21.12.5 Object

**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
```



**21.12.6 MindMap**

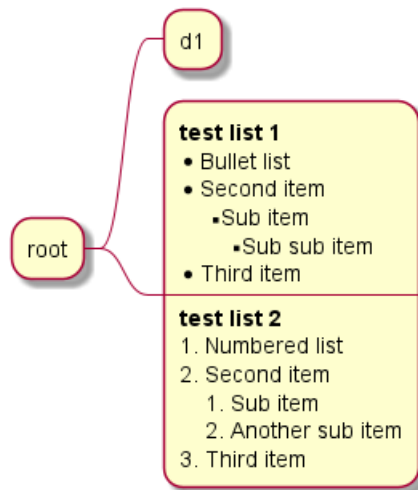
```

@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

```

**21.12.7 Network (nwdiag)**

N/A

**21.12.8 Note**

```

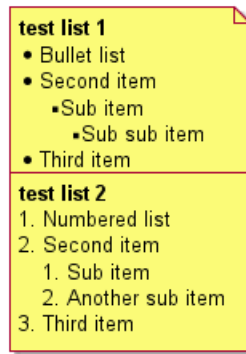
@startuml
note as n
**test list 1**
* Bullet list
* Second item
** Sub item
*** Sub sub item
* Third item
----
**test list 2**

```





```
# Numbered list
# Second item
## Sub item
## Another sub item
# Third item
end note
@enduml
```



### 21.12.9 Sequence

N/A (or on note or common commands)

### 21.12.10 State

N/A (or on note or common commands)

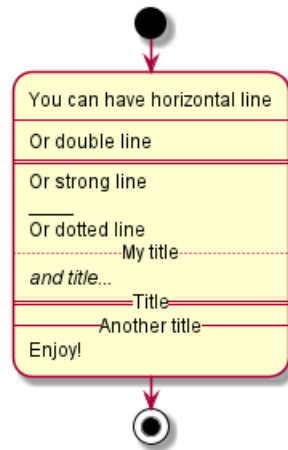
## 21.13 Appendice: Examples of "Creole horizontal lines" on all diagrams

### 21.13.1 Activity

**TODO:** FIXME  strong line \_\_\_\_ **TODO:** FIXME

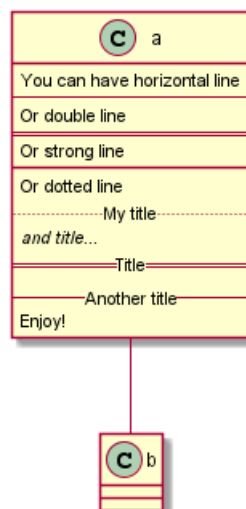
```
@startuml
start
:You can have horizontal line
----
Or double line
====
Or strong line
-----
Or dotted line
..My title..
//and title... //
==Title==
--Another title--
Enjoy!;
stop
@enduml
```





### 21.13.2 Class

```
@startuml
class a {
You can have horizontal line
----
Or double line
====
Or strong line
----
Or dotted line
..My title..
//and title... //
==Title==
--Another title--
Enjoy!
}
a -- b
@enduml
```



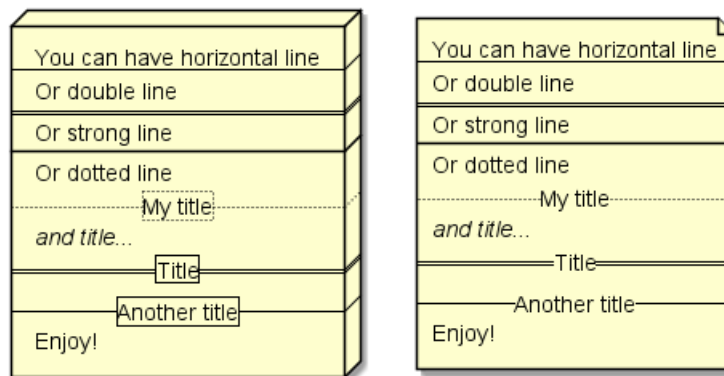
### 21.13.3 Component, Deployment, Use-Case

```

@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

```



### 21.13.4 Gantt project planning

N/A

### 21.13.5 Object

```

@startuml
object user {
You can have horizontal line
----
Or double line

```

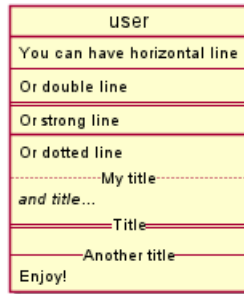


```

====
Or strong line
----
Or dotted line
..My title..
//and title... //
==Title==
--Another title--
Enjoy!
}

@enduml

```



**TODO:** DONE [Corrected on V1.2020.18]

### 21.13.6 MindMap

**TODO:** FIXME □ strong line \_\_\_\_ **TODO:** FIXME

```

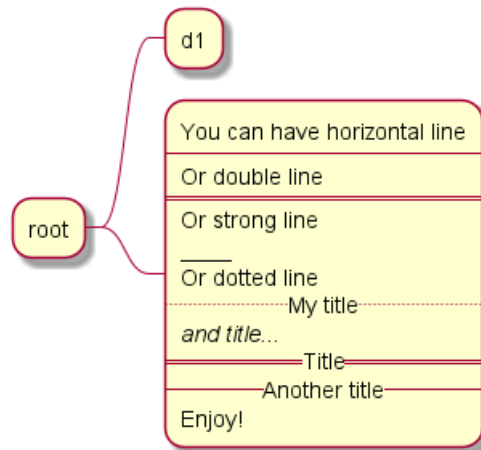
@startmindmap

* 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

```



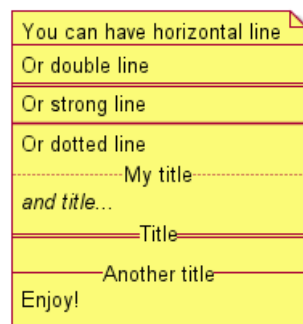


### 21.13.7 Network (nwdiag)

N/A

### 21.13.8 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
```



### 21.13.9 Sequence

N/A (or on note or common commands)



## 21.13.10 State

N/A (or on note or common commands)

## 21.14 Style equivalent (between Creole and HTML)

Style	Creole	Legacy HTML like
<b>bold</b>	This is <b>bold</b>	This is <b>bold</b>
<i>italics</i>	This is //italics//	This is <i>italics</i>
monospaced	This is "monospaced"	This is <font:monospaced>monospaced</font>
stroked	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:"<color:#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 <b>**bold**</b>	This is <b>bold</b>
This is <i>//italics//</i>	This is <i>italics</i>
This is <code>""monospaced""</code>	This is monospaced
This is <del>--stroked--</del>	This is <del>stroked</del>
This is <u>__underlined__</u>	This is <u>underlined</u>
This is <u>~~waved~~</u>	This is <u>waved</u>

Legacy HTML like	
code	output
This is <code>&lt;b&gt;bold&lt;/b&gt;</code>	This is <b>bold</b>
This is <code>&lt;i&gt;italics&lt;/i&gt;</code>	This is <i>italics</i>
This is <code>&lt;font:monospaced&gt;monospaced&lt;/font&gt;</code>	This is monospaced
This is <code>&lt;s&gt;stroked&lt;/s&gt;</code>	This is <del>stroked</del>
This is <code>&lt;u&gt;underlined&lt;/u&gt;</code>	This is <u>underlined</u>
This is <code>&lt;w&gt;waved&lt;/w&gt;</code>	This is <u>waved</u>

And color as a bonus...

code	output
This is <code>&lt;s:green&gt;stroked&lt;/s&gt;</code>	This is <del>stroked</del>
This is <code>&lt;u:red&gt;underlined&lt;/u&gt;</code>	This is <u>underlined</u>
This is <code>&lt;w:#0000FF&gt;waved&lt;/w&gt;</code>	This is <u>waved</u>

## 22 Defining and using sprites

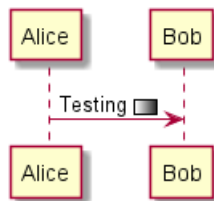
A *Sprite* is a small graphic element that can be used in diagrams.

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.

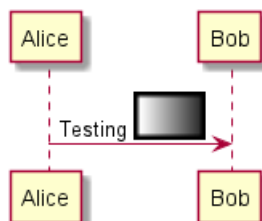
Then you can use the sprite using <\$XXX> where XXX is the name of the sprite.

```
@startuml
sprite $foo1 {
  FFFFFFFFFFFFFFFF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  FFFFFFFFFFFFFFFF
}
Alice -> Bob : Testing <$foo1>
@enduml
```



You can scale the sprite.

```
@startuml
sprite $foo1 {
  FFFFFFFFFFFFFFFF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  FFFFFFFFFFFFFFFF
}
Alice -> Bob : Testing <$foo1{scale=3}>
@enduml
```

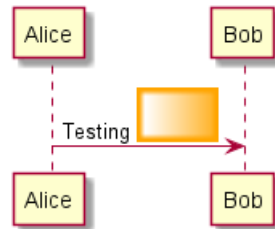




## 22.1 Changing colors

Although sprites are monochrome, it's possible to change their color.

```
@startuml
sprite $foo1 {
  FFFFFFFFFFFFFFFF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  FFFFFFFFFFFFFFFF
}
Alice -> Bob : Testing <$foo1,scale=3.4,color=orange>
@enduml
```



## 22.2 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.

## 22.3 Importing Sprite

You can also launch the GUI to generate a sprite from an existing image.

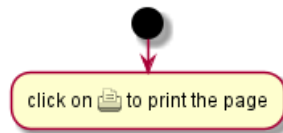
Click in the menubar then on `File/Open Sprite Window`.

After copying an image into you clipboard, several possible definitions of the corresponding sprite will be displayed : you will just have to pickup the one you want.

## 22.4 Examples

```
@startuml
sprite $printer [15x15/8z] N0tH3W0W208HxFz_kMAhj71HWpa1XC716szOPq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvF
start
:click on <$printer> to print the page;
@enduml
```





```

@startuml
sprite $bug [15x15/16z] PKzR2i0m2BFMi15p__FEjQEqB1z27aeqCqixa8S40T7C53cKpsHpaYPDJY_12MHM-BLRyywPhrr1w
sprite $printer [15x15/8z] N0tH3WOW208HxFz_kMAhj71HWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvF
sprite $disk {
  444445566677881
  43600000009991
  4360000000ACA1
  5370000001A7A1
  53700000012B8A1
  53800000123B8A1
  63800001233C9A1
  634999AABBC99B1
  744566778899AB1
  7456AAAAA99AAB1
  8566AFC228AABB1
  8567AC8118BBBB1
  867BD4433BBBBB1
  39AAAAABBBBBBC1
}

title Use of sprites (<$printer>, <$bug>...)

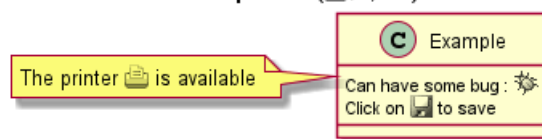
class Example {
  Can have some bug : <$bug>
  Click on <$disk> to save
}

note left : The printer <$printer> is available

@enduml

```

### Use of sprites (🖨️, ⚙️...)



## 22.5 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



## 22.6 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-ico
!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.

## 23 Skinparam command

You can change colors and font of the drawing using the `skinparam` command.

Example:

```
skinparam backgroundColor transparent
```

### 23.1 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.

### 23.2 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
}
```

### 23.3 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
```



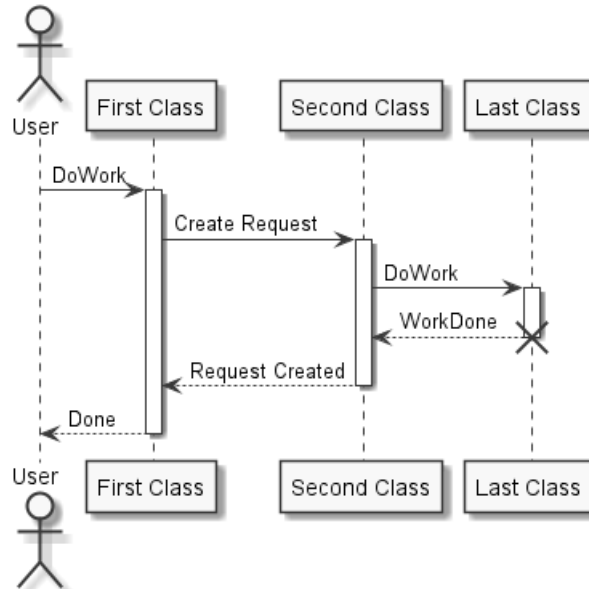
```

deactivate B

A --> User: Done
deactivate A

@enduml

```



## 23.4 Shadowing

You can disable the shadowing using the skinparam shadowing false command.

```

@startuml

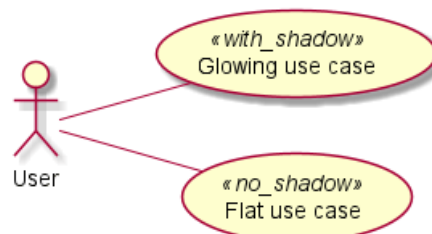
left to right direction

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

```



## 23.5 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

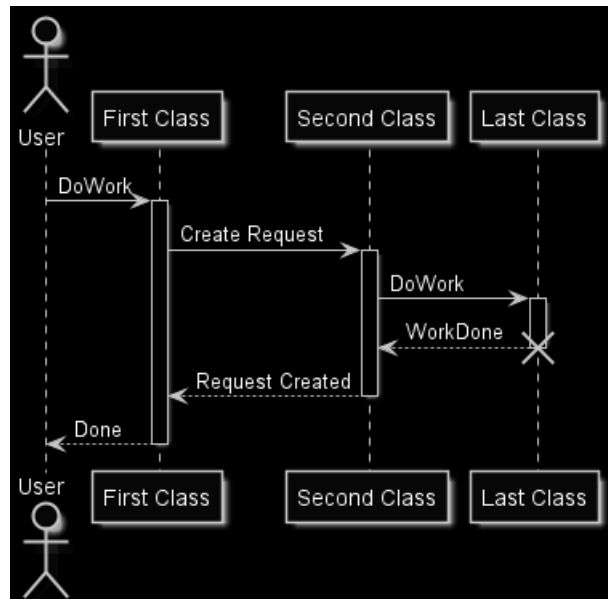
B -> C: DoWork
activate C
C --> B: WorkDone
destroy C

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml

```



## 23.6 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.

## 23.7 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
```

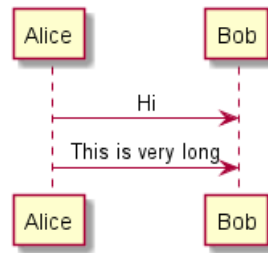
## 23.8 Text Alignment

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
<code>sequenceMessageAlign</code>	left	Used for messages in sequence diagrams
<code>sequenceReferenceAlign</code>	center	Used for ref over in sequence diagrams

```
@startuml
skinparam sequenceMessageAlign center
Alice -> Bob : Hi
Alice -> Bob : This is very long
@enduml
```





## 23.9 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 Apex
}

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

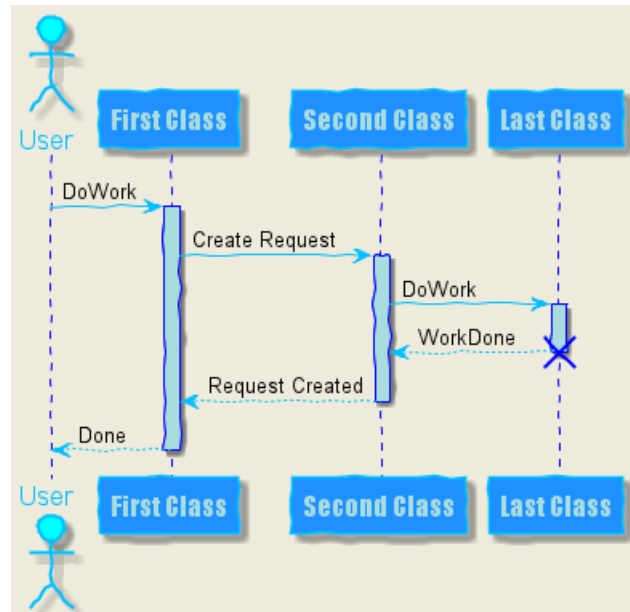
B --> A: Request Created
deactivate B

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

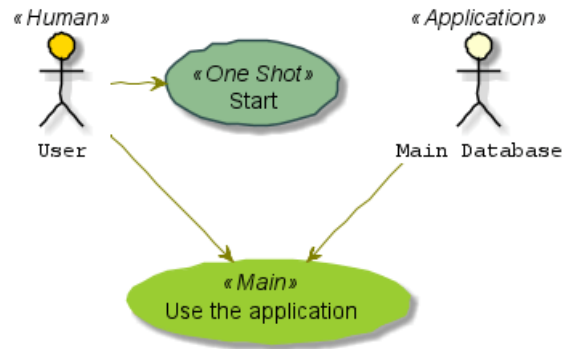
  ArrowColor Olive
}

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

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

MySql --> (Use)
@enduml

```



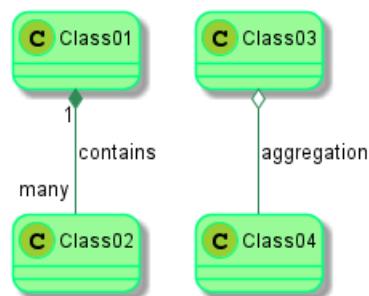
```

@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
}
  
```

```

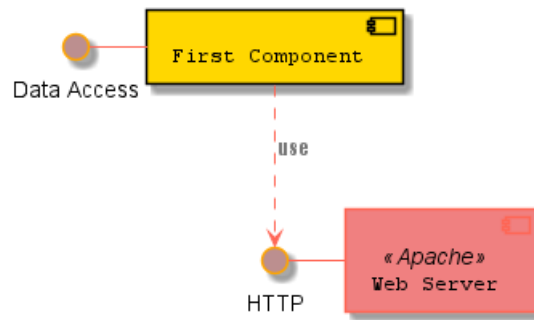
() "Data Access" as DA
[Web Server] << Apache >>
  
```

```

DA - [First Component]
[First Component] ..> () HTTP : use
  
```



```
HTTP - [Web Server]
@enduml
```



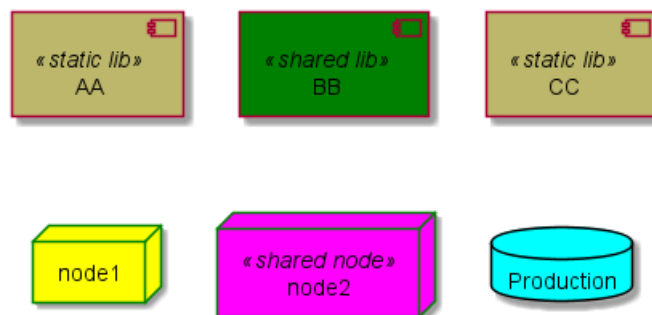
```

@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
  
```



## 23.10 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:

That will give you the following result:

```

@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 <https://plantuml-documentation.readthedocs.io/en/latest/for-skin-params.html>.

## 24 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 !.

### 24.1 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.

### 24.2 Variable definition

Although this is not mandatory, we highly suggest that variable names start with a \$.

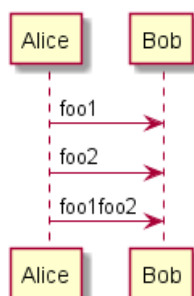
There are two types of data:

- **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
```



## 24.3 Boolean expression

### 24.3.1 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]

### 24.3.2 Boolean operation and operator [&&, ||, ()]

You can use boolean expression, in the test, with :

- *parenthesis* ();
- *and operator* &&;
- *or operator* ||.

(See next example, within *if* test.)

### 24.3.3 Boolean builtin functions [%false(), %true(), %not(<exp>)]

For convenience, you can use those boolean builtin functions:

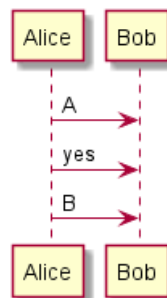
- %false()
- %true()
- %not(<exp>)

[See also *Builtin functions*]

## 24.4 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
```

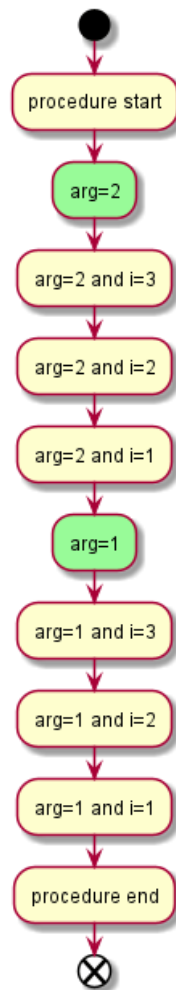


## 24.5 While loop [!while, !endwhile]

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)
```





```

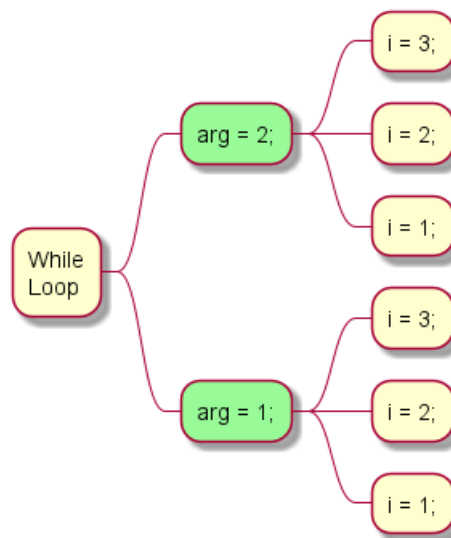
!while $arg!=0
  !$i=3
  **[#palegreen] arg = $arg;
  !while $i!=0
    *** i = $i;
    !$i = $i - 1
  !endwhile
  !$arg = $arg - 1
!endwhile
!endprocedure

```

```

*:While
Loop;
$foo(2)
@endmindmap

```



## 24.6 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.

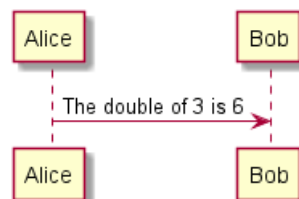
## 24.7 Return function [!function, !endfunction]

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
```

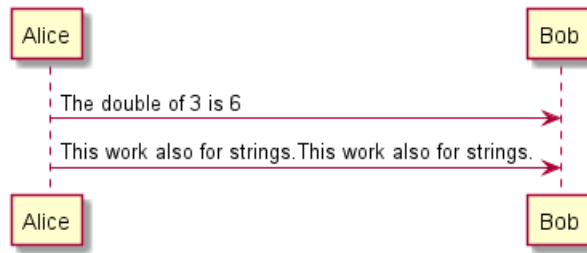


It is possible to shorten simple function definition in one line:

```
@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
```





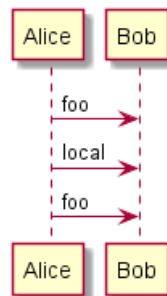
As in procedure (void function), variables are local by default (they are destroyed when the function is exited). However, you can access global variables from a function. However, you can use the `local` keyword to create a local variable if ever a global variable exists with the same name.

```

@startuml
!function $dummy()
!local $ijk = "local"
!return "Alice -> Bob : " + $ijk
!endfunction

!global $ijk = "foo"

Alice -> Bob : $ijk
$dummy()
Alice -> Bob : $ijk
@enduml
  
```



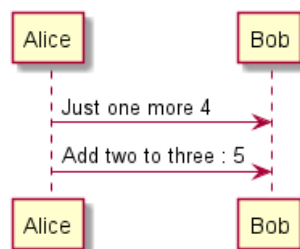
## 24.8 Default argument value

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.

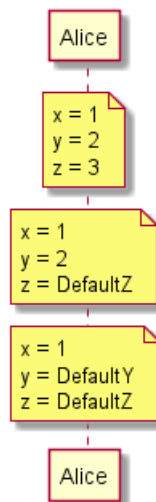


```

@startuml
!procedure defaultttest($x, $y="DefaultY", $z="DefaultZ")
note over Alice
  x = $x
  y = $y
  z = $z
end note
!endprocedure

defaultttest(1, 2, 3)
defaultttest(1, 2)
defaultttest(1)
@enduml

```



## 24.9 Unquoted procedure or function [!unquoted]

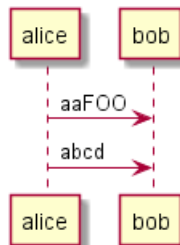
By default, you have to put quotes when you call a function or a procedure. It is possible to use the unquoted keyword to indicate that a function or a procedure does not require quotes for its arguments.

```

@startuml
!unquoted function id($text1, $text2="FOO") !return $text1 + $text2

alice -> bob : id(aa)
alice -> bob : id(ab,cd)
@enduml

```



## 24.10 Keywords arguments

Like in Python, you can use keywords arguments :

```

@startuml

```



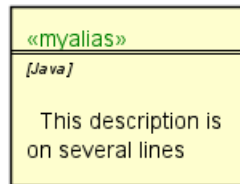
```

!unquoted procedure $element($alias, $description="", $label="", $technology="", $size=12, $colour="green",
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

```



## 24.11 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.

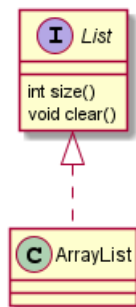
Imagine you have the very same class that appears in many diagrams. Instead of duplicating the description of this class, you can define a file that contains the description.

```

@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.

## 24.12 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
```

## 24.13 Builtin functions [%]

Some functions are defined by default. Their name starts by %



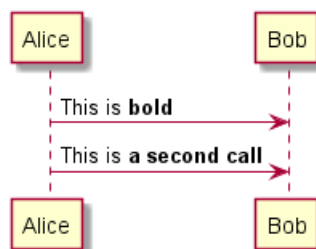
Name	Description	Example
%date	Retrieve current date. You can provide an optional format for the date	%date("yyyy.MM.dd' at
%dirpath	Retrieve current dirpath	%dirpath()
%false	Return always false	%false()
%file_exists	Check if a file exists on the local filesystem	%file_exists("c:/foo/d
%filename	Retrieve current filename	%filename()
%function_exists	Check if a function exists	%function_exists("\$som
%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", "ef"
%substr	Extract a substring. Takes 2 or 3 arguments	%substr("abcdef", 3, 2
%true	Return always true	%true()
%upper	Return an uppercase string	%upper("Hello")
%variable_exists	Check if a variable exists	%variable_exists("\$my_
%version	Return PlantUML current version	%version()

## 24.14 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
```



## 24.15 Memory dump [!memory\_dump]

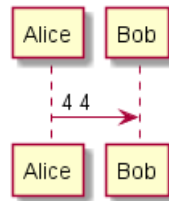
You can use `!memory_dump` to dump the full content of the memory when generating the diagram. An optional string can be put after `!memory_dump`. This has no impact at all on the diagram itself. This could be useful for debug purpose.

```
@startuml
!function $inc($string)
!$val = %intval($string)
!log value is $val
```



```
!dump_memory
!return $val+1
!endfunction

Alice -> Bob : 4 $inc("3")
!unused = "foo"
!dump_memory EOF
@enduml
```



## 24.16 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!

If you use this software, you accept its license.  
(details by typing license keyword)

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>



PlantUML 1.2020.24beta3

[From string (line 3) ]

```
@startuml
```

```
Alice -> Bob : Hello
```

```
!assert %strpos("abcdef", "cd")==3 : "This always fails"
```

```
Assertion error : This always fails
```

## 24.17 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
...

```

## 24.18 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
```

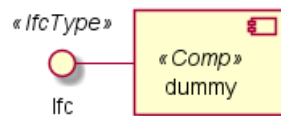
Note the this `-D` option has to put before the `-jar` option. `-D` options after the `-jar` option will be used to define constants within `plantuml` preprocessor.

## 24.19 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

```



## 24.20 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.

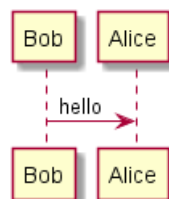
For example, you can have:

```
@startuml
!procedure $go()
  Bob -> Alice : hello
!endprocedure

!$wrapper = "$go"

%invoke_procedure($wrapper)
@enduml

```

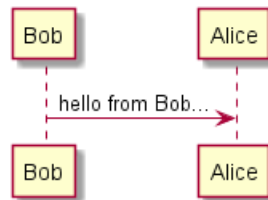


```

@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()` :

```

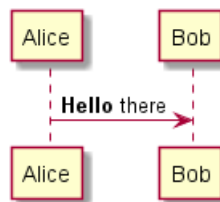
@startuml
!function bold($text)
!return "<b>"+ $text + "</b>"
!endfunction

```

```

Alice -> Bob : %call_user_func("bold", "Hello") there
@enduml

```



## 24.21 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>
type	str	str	str (concatenation)
example	"a"	"b"	ab
type	str	int	str (concatenation)
ex.	"a"	2	a2
type	str	int	str (concatenation)
ex.	1	"b"	1b
type	bool	str	str (concatenation)
ex.	%true()	"b"	1b
type	str	bool	str (concatenation)
ex.	"a"	%false()	a0
type	int	int	int (addition of int)
ex.	1	2	3
type	bool	int	int (addition)
ex.	%true()	2	3
type	int	bool	int (addition)
ex.	1	%false()	1
type	int	int	int (addition)
ex.	1	%intval("2")	3

## 24.22 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)

## 25 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*.

### 25.1 Examples

```
@startuml
skinparam handwritten true
skinparam backgroundColor #EEEEBD

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
```

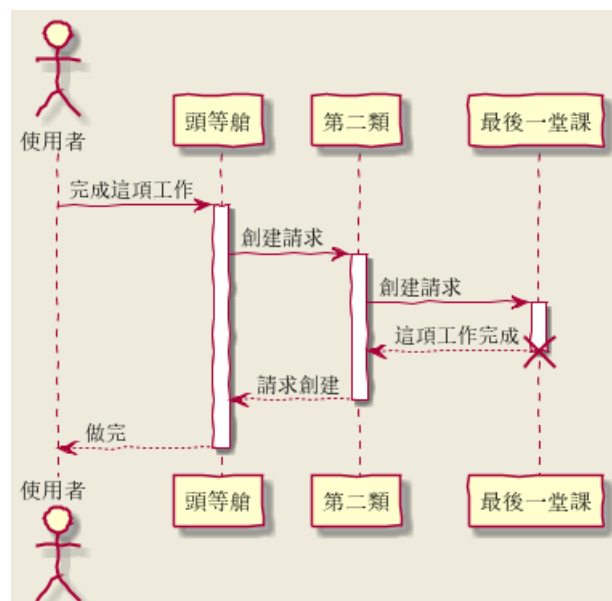
```
使用者 -> A: 完成這項工作
activate A
```

```
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
```



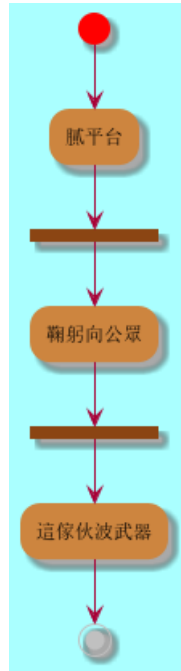
```
@startuml

(*) --> "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
```

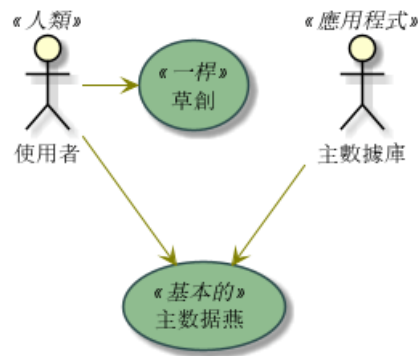
```
skinparam usecaseBackgroundColor DarkSeaGreen
skinparam usecaseArrowColor Olive
skinparam actorBorderColor black
skinparam usecaseBorderColor DarkSlateGray
```

```
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
```





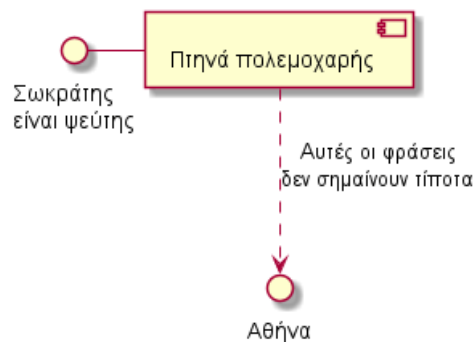
@startuml

() "Σωκράτης ψεύτης" as Σωκράτης

Σωκράτης - [Πτηνά πολεμοχαρής]

[Πτηνά πολεμοχαρής] ..> () Αθήνα : Αυτές οι φράσεις σημαίνουν τίποτα

@enduml



## 25.2 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 the 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.



## 26 Standard Library

This page explains the official Standard Library for PlantUML. This Standard Library is now included in official releases of PlantUML. Including files follows the C convention for "C standard library" (see [https://en.wikipedia.org/wiki/C\\_standard\\_library](https://en.wikipedia.org/wiki/C_standard_library))

Contents of the library come from third party contributors. We thank them for their useful contribution!

### 26.1 Amazon Labs Library

<https://github.com/aws-labs/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/aws-labs/aws-icons-for-plantuml/blob

!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
```

### 26.2 AWS library

<https://github.com/milo-minderbinder/AWS-PlantUML>

The AWS library consists of Amazon AWS icons, it provides icons of two different sizes.

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>`.

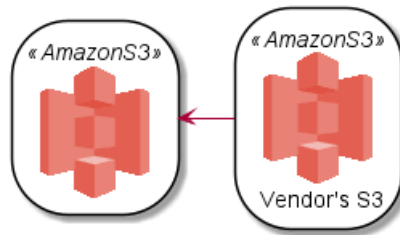


You may also include the `common.puml` file, eg: `!include <aws/common>`, which contains helper macros defined. With the `common.puml` imported, you can use the `NAME_OF_SPRITE(parameters...)` macro.

Example of usage:

```
@startuml
!include <aws/common>
!include <aws/Storage/AmazonS3/AmazonS3>
!include <aws/Storage/AmazonS3/bucket/bucket>

AMAZONS3(s3_internal)
AMAZONS3(s3_partner,"Vendor's S3")
s3_internal <- s3_partner
@enduml
```



## 26.3 Azure library

<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/Databases/AzureCosmosDb.puml>
```

left to right direction

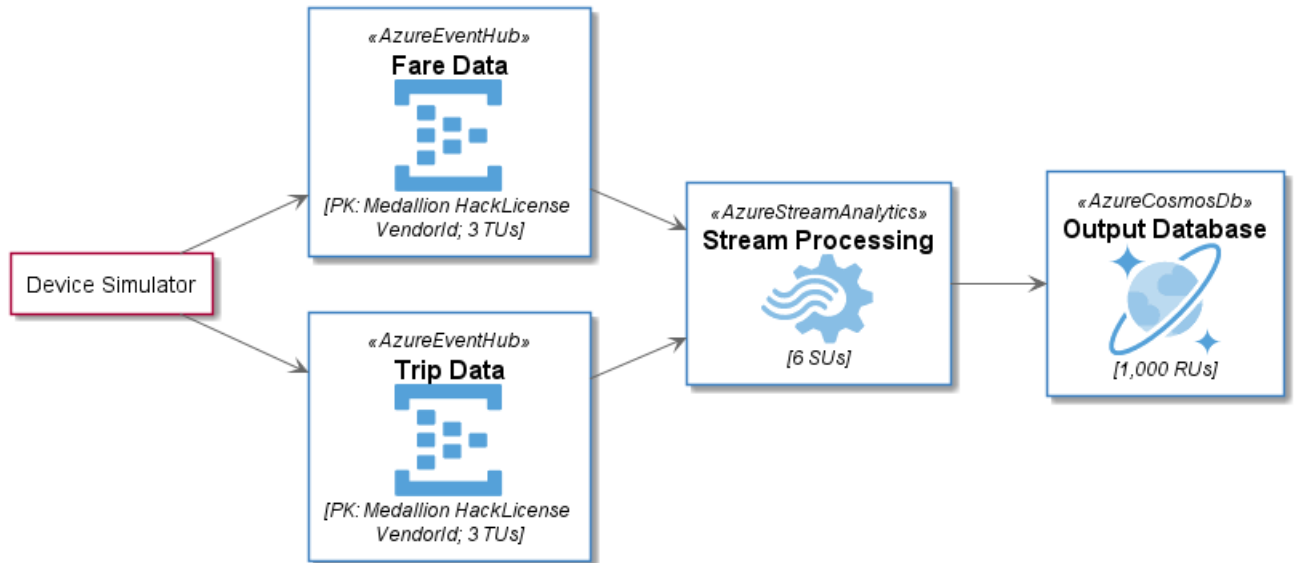
```
agent "Device Simulator" as devices #fff
```

```
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
```







## 26.4 Cloud Insight

<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>

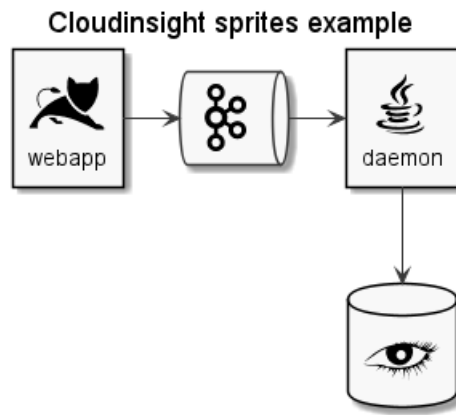
title Cloudinsight sprites example

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
  
```





## 26.5 Elastic library

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/elastic_search/elastic_search.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.

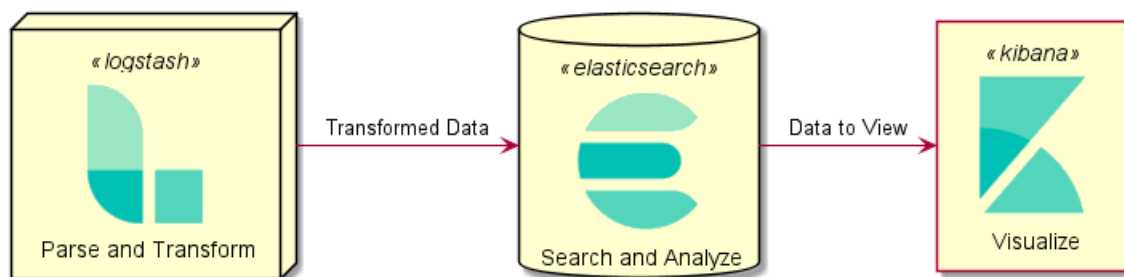
Example of usage:

```

@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
  
```



## 26.6 Tupadr3 library

<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/align_center>`. 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>

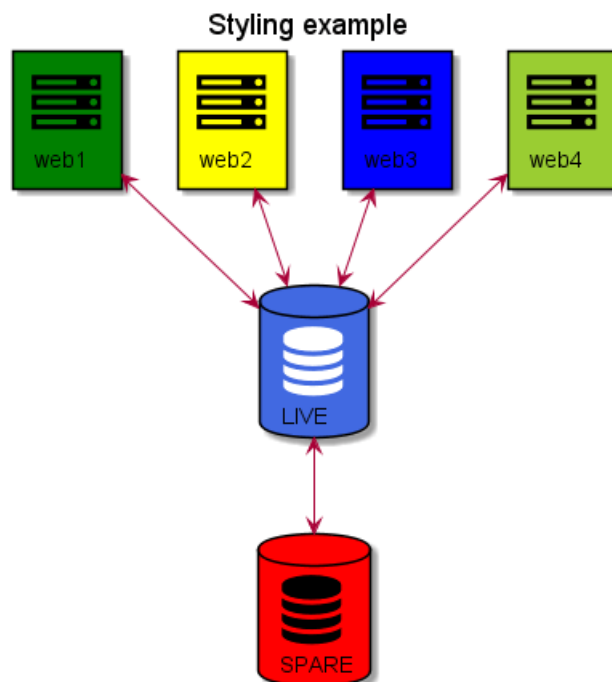
title Styling example

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

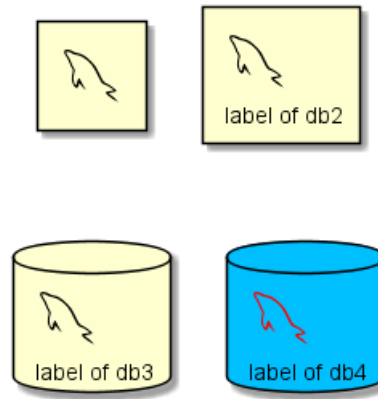
db1 <--> db2

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
```



## 26.7 Google Material Icons

<https://github.com/Templarian/MaterialDesign>

This library consists of a free Material style icons from Google and other artists.

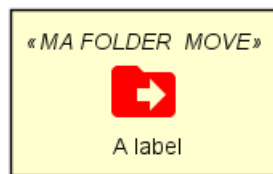
Use it by including the file that contains the sprite, eg: `!include <material/ma_folder_move>`. When imported, you can use the sprite as normally you would, using `<$ma_sprite_name>`. Notice that this library requires an `ma_` prefix on sprites names, this is to avoid clash of names if multiple sprites 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:

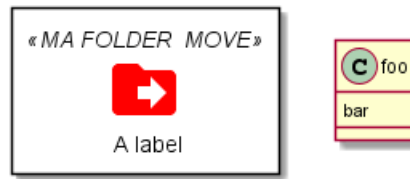
```
@startuml
!include <material/common>
' To import the sprite file you DON'T need to place a prefix!
!include <material/folder_move>
```

```
MA_FOLDER_MOVE(Red, 1, dir, rectangle, "A label") {
}
```

```
class foo {
    bar
```



```
}
@enduml
```



## 26.8 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.

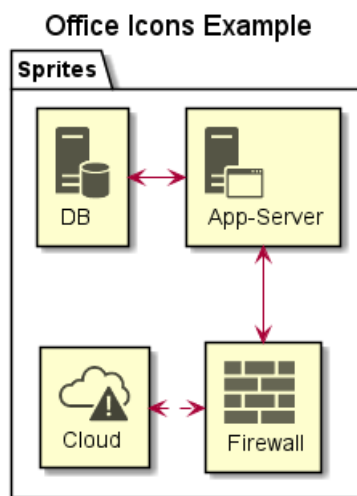
Example of usage:

```
@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>

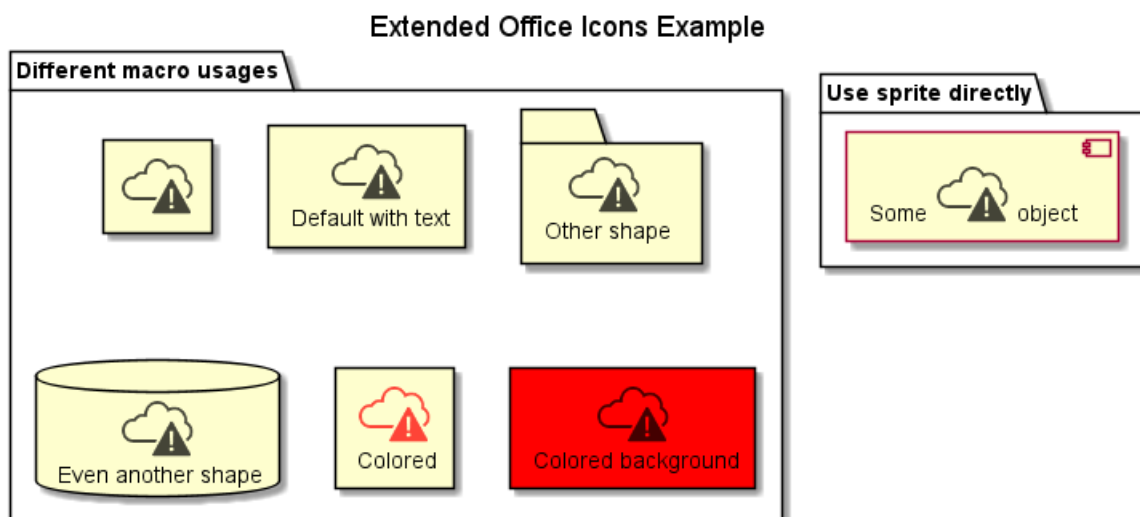
' Used to center the label under the images
skinparam defaultTextAlignment center

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

```



## 26.9 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)")

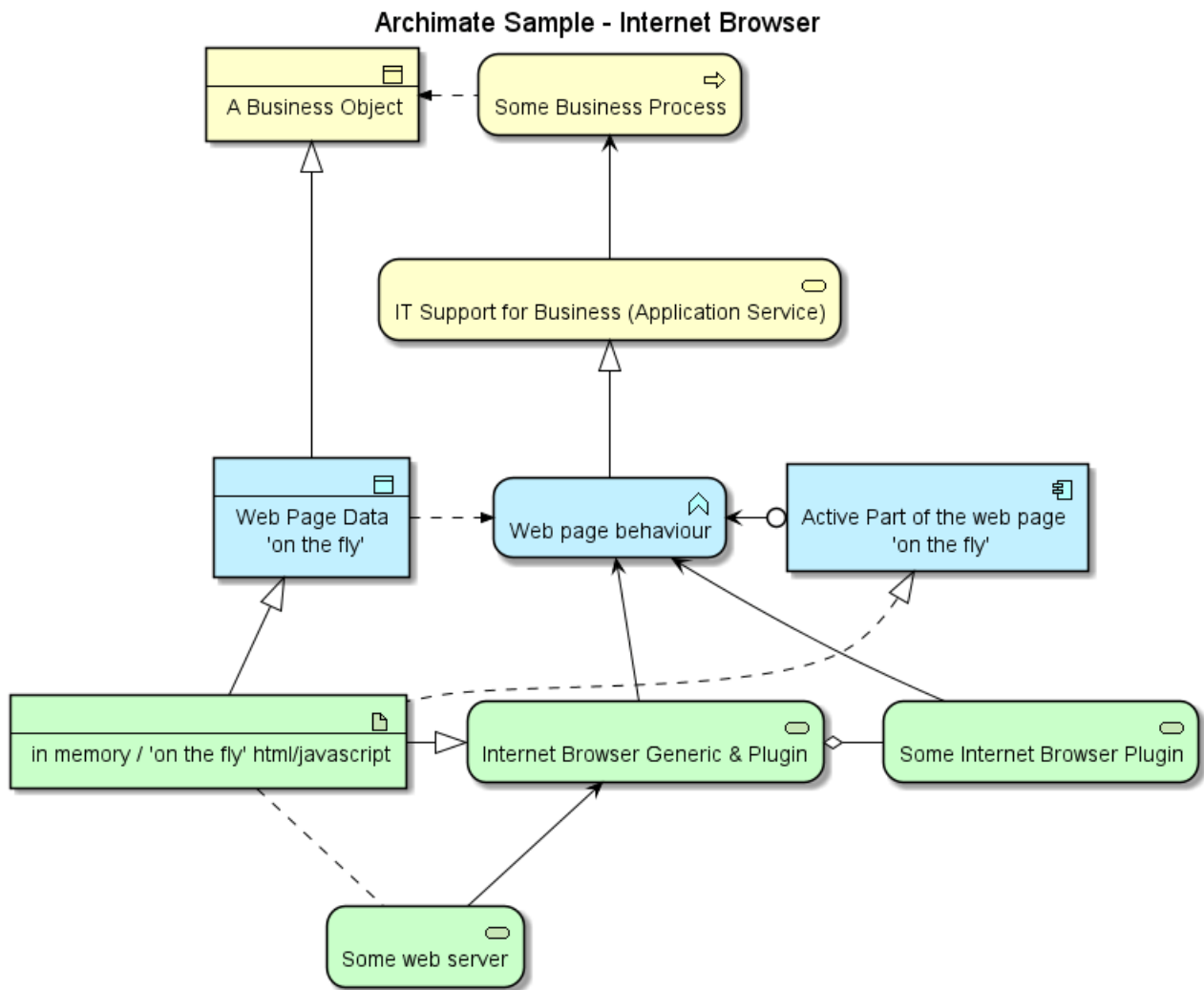
```



```
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
```



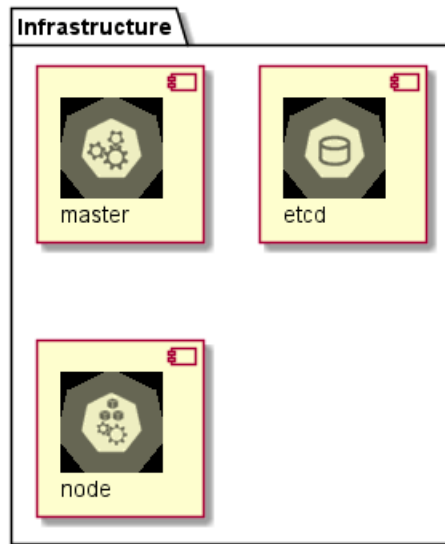
## 26.10 Kubernetes

<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
  
```





## 26.11 Miscellaneous

You can list standard library folders using the special diagram:

```
@startuml
stdlib
@enduml
```

**archimate**

Version 0.0.1

Delivered by <https://github.com/ebbpeter/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 1.0.0

Delivered by <https://github.com/RicardoNiepel/C4-PlantUML>**cloudinsight**

Version 1.0.0

Delivered by <https://github.com/rabelenda/cicon-plantuml-sprites/>**cloudogu**

Version 0.0.1

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 0.0.1

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.



## Contents

<b>1</b>	<b>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 . . . . .	3
1.5.1	Text of response message below the arrow . . . . .	3
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 . . . . .	8
1.12	Secondary group label . . . . .	10
1.13	Notes on messages . . . . .	10
1.14	Some other notes . . . . .	11
1.15	Changing notes shape . . . . .	12
1.16	Creole and HTML . . . . .	12
1.17	Divider . . . . .	13
1.18	Reference . . . . .	14
1.19	Delay . . . . .	15
1.20	Text wrapping . . . . .	15
1.21	Space . . . . .	16
1.22	Lifeline Activation and Destruction . . . . .	16
1.23	Return . . . . .	18
1.24	Participant creation . . . . .	19
1.25	Shortcut syntax for activation, deactivation, creation . . . . .	19
1.26	Incoming and outgoing messages . . . . .	20
1.27	Short arrows for incoming and outgoing messages . . . . .	22
1.28	Anchors and Duration . . . . .	23
1.29	Stereotypes and Spots . . . . .	23
1.30	More information on titles . . . . .	24
1.31	Participants encompass . . . . .	25
1.32	Removing Foot Boxes . . . . .	26
1.33	Skinparam . . . . .	26
1.34	Changing padding . . . . .	28
1.35	Appendice: Examples of all arrow type . . . . .	29
1.35.1	Normal arrow . . . . .	29
1.35.2	Incoming and outgoing messages (with '[', ']') . . . . .	30
1.35.3	Short incoming and outgoing messages (with '?') . . . . .	33
1.36	Specific SkinParameter . . . . .	35
1.36.1	By default . . . . .	35
1.36.2	lifelineStrategy solid . . . . .	35
1.36.3	style strictuml . . . . .	35
1.37	Hide unlinked participant . . . . .	36
<b>2</b>	<b>Use Case Diagram</b>	<b>37</b>
2.1	Usecases . . . . .	37
2.2	Actors . . . . .	37
2.3	Change Actor style . . . . .	38
2.3.1	Stick man ( <i>by default</i> ) . . . . .	38
2.3.2	Awesome man . . . . .	38
2.3.3	Hollow man . . . . .	38
2.4	Usecases description . . . . .	39
2.5	Use package . . . . .	39
2.6	Basic example . . . . .	41
2.7	Extension . . . . .	41



2.8	Using notes . . . . .	42
2.9	Stereotypes . . . . .	43
2.10	Changing arrows direction . . . . .	43
2.11	Splitting diagrams . . . . .	45
2.12	Left to right direction . . . . .	45
2.13	Skinparam . . . . .	46
2.14	Complete example . . . . .	47
<b>3</b>	<b>Class Diagram</b>	<b>48</b>
3.1	Declaring element . . . . .	48
3.2	Relations between classes . . . . .	48
3.3	Label on relations . . . . .	49
3.4	Adding methods . . . . .	50
3.5	Defining visibility . . . . .	51
3.6	Abstract and Static . . . . .	52
3.7	Advanced class body . . . . .	52
3.8	Notes and stereotypes . . . . .	53
3.9	More on notes . . . . .	54
3.10	Note on field (field, attribut, member) or method . . . . .	55
3.10.1	Note on field or method . . . . .	55
3.10.2	Note on method with the same name . . . . .	55
3.11	Note on links . . . . .	56
3.12	Abstract class and interface . . . . .	56
3.13	Using non-letters . . . . .	57
3.14	Hide attributes, methods... . . . .	58
3.15	Hide classes . . . . .	59
3.16	Use generics . . . . .	59
3.17	Specific Spot . . . . .	60
3.18	Packages . . . . .	60
3.19	Packages style . . . . .	60
3.20	Namespaces . . . . .	62
3.21	Automatic namespace creation . . . . .	62
3.22	Lollipop interface . . . . .	63
3.23	Changing arrows direction . . . . .	63
3.24	Association classes . . . . .	65
3.25	Association on same classe . . . . .	66
3.26	Skinparam . . . . .	66
3.27	Skinned Stereotypes . . . . .	67
3.28	Color gradient . . . . .	67
3.29	Help on layout . . . . .	68
3.30	Splitting large files . . . . .	69
3.31	Extends and implements . . . . .	70
3.32	Inline style of relations (Linking or arrow) . . . . .	70
3.33	Change relation, linking or arrow color and style . . . . .	71
3.34	Arrows from/to class members . . . . .	72
<b>4</b>	<b>Object Diagram</b>	<b>74</b>
4.1	Definition of objects . . . . .	74
4.2	Relations between objects . . . . .	74
4.3	Associations objects . . . . .	74
4.4	Adding fields . . . . .	75
4.5	Common features with class diagrams . . . . .	75
4.6	Map table or associative array . . . . .	76
<b>5</b>	<b>Activity Diagram (legacy)</b>	<b>78</b>
5.1	Simple Action . . . . .	78
5.2	Label on arrows . . . . .	78
5.3	Changing arrow direction . . . . .	78
5.4	Branches . . . . .	79



5.5	More on Branches . . . . .	80
5.6	Synchronization . . . . .	81
5.6.1	# Long action description . . . . .	82
5.7	Notes . . . . .	82
5.8	Partition . . . . .	83
5.9	Skinparam . . . . .	84
5.10	Octagon . . . . .	85
5.11	Complete example . . . . .	85
<b>6</b>	<b>Activity Diagram (new)</b>	<b>88</b>
6.1	Simple action . . . . .	88
6.2	Start/Stop/End . . . . .	88
6.3	Conditional . . . . .	89
6.3.1	Several tests (horizontal mode) . . . . .	90
6.3.2	Several tests (vertical mode) . . . . .	91
6.4	Conditional with stop on an action [kill, detach] . . . . .	91
6.5	Repeat loop . . . . .	93
6.6	Break on a repeat loop [break] . . . . .	94
6.7	While loop . . . . .	95
6.8	Parallel processing . . . . .	96
6.9	Notes . . . . .	97
6.10	Colors . . . . .	98
6.11	Lines without arrows . . . . .	98
6.12	Arrows . . . . .	99
6.13	Connector . . . . .	100
6.14	Color on connector . . . . .	100
6.15	Grouping . . . . .	101
6.16	Swimlanes . . . . .	102
6.17	Detach or kill [detach, kill] . . . . .	103
6.18	SDL (Specification and Description Language) . . . . .	105
6.19	Complete example . . . . .	106
6.20	Condition Style . . . . .	108
6.20.1	Inside style (by default) . . . . .	108
6.20.2	Diamond style . . . . .	109
6.20.3	InsideDiamond (or <i>Foo1</i> ) style . . . . .	110
6.21	Condition End Style . . . . .	111
6.21.1	Diamond style (by default) . . . . .	111
6.21.2	Horizontal line (hline) style . . . . .	112
<b>7</b>	<b>Component Diagram</b>	<b>114</b>
7.1	Components . . . . .	114
7.2	Interfaces . . . . .	114
7.3	Basic example . . . . .	115
7.4	Using notes . . . . .	115
7.5	Grouping Components . . . . .	116
7.6	Changing arrows direction . . . . .	117
7.7	Use UML2 notation . . . . .	118
7.8	Use UML1 notation . . . . .	119
7.9	Use rectangle notation (remove UML notation) . . . . .	119
7.10	Long description . . . . .	120
7.11	Individual colors . . . . .	120
7.12	Using Sprite in Stereotype . . . . .	120
7.13	Skinparam . . . . .	121
7.14	Specific SkinParameter . . . . .	122
7.14.1	componentStyle . . . . .	122
<b>8</b>	<b>Deployment Diagram</b>	<b>125</b>
8.1	Declaring element . . . . .	125
8.2	Declaring element (using short form) . . . . .	127



8.3	Linking or arrow	128
8.4	Change arrow color and style	130
8.5	Nestable elements	131
8.6	Packages and nested elements	132
8.6.1	Example with one level	132
8.6.2	Other example	133
8.6.3	Full nesting	134
8.7	Round corner	137
8.8	Alias	138
8.8.1	Simple alias with as	138
8.8.2	Examples of long alias	138
8.9	Type of arrow head or 'O' arrow	140
8.9.1	Type of arrow head	140
8.9.2	Type of 'O' arrow or circle arrow	143
8.10	Specific SkinParameter	144
8.10.1	roundCorner	144
<b>9</b>	<b>State Diagram</b>	<b>146</b>
9.1	Simple State	146
9.2	Change state rendering	146
9.3	Composite state	147
9.3.1	Internal sub-state	147
9.3.2	Sub-state to sub-state	148
9.4	Long name	149
9.5	History [[H], [H*]]	150
9.6	Fork [fork, join]	151
9.7	Concurrent state [--,   ]	152
9.7.1	Horizontal separator --	152
9.7.2	Vertical separator	153
9.8	Conditional [choice]	154
9.9	Stereotypes full example [choice, fork, join, end]	154
9.10	Point [entryPoint, exitPoint]	155
9.11	Pin [inputPin, outputPin]	156
9.12	Expansion [expansionInput, expansionOutput]	157
9.13	Arrow direction	158
9.14	Change line color and style	159
9.15	Note	159
9.16	Note on link	160
9.17	More in notes	160
9.18	Inline color	161
9.19	Skinparam	162
9.20	Changing style	163
<b>10</b>	<b>Timing Diagram</b>	<b>165</b>
10.1	Declaring participant	165
10.2	Binary and Clock	165
10.3	Adding message	166
10.4	Relative time	166
10.5	Anchor Points	167
10.6	Participant oriented	168
10.7	Setting scale	168
10.8	Initial state	168
10.9	Intricated state	169
10.10	Hidden state	170
10.11	Hide time axis	170
10.12	Using Time and Date	171
10.13	Adding constraint	172
10.14	Highlighted period	172
10.15	Adding texts	173



10.16	Complete example	174
10.17	Digital Example	175
10.18	Adding color	176
<b>11</b>	<b>Display JSON Data</b>	<b>178</b>
11.1	Complex example	178
11.2	Highlight parts	179
11.3	JSON basic element	180
11.3.1	Synthesis of all JSON basic element	180
11.4	JSON array or table	181
11.4.1	Array type	181
11.4.2	Minimal array or table	181
11.4.3	Number array	181
11.4.4	String array	181
11.4.5	Boolean array	181
11.5	JSON numbers	182
11.6	JSON strings	182
11.6.1	JSON Unicode	182
11.6.2	JSON two-character escape sequence	182
11.7	Minimal JSON examples	183
<b>12</b>	<b>Network diagram (nwdiag)</b>	<b>185</b>
12.1	Simple diagram	185
12.2	Define multiple addresses	185
12.3	Grouping nodes	186
12.4	Extended Syntax	187
12.5	Using Sprite on nwdiag	188
12.6	Using OpenIconic on nwdiag	189
12.7	Same nodes on more than two networks	190
12.8	Peer networks	191
12.9	Peer networks and group	191
12.10	Add title, header, footer or legend on network diagram	192
12.11	Change width of the networks	193
<b>13</b>	<b>Salt (Wireframe)</b>	<b>196</b>
13.1	Basic widgets	196
13.2	Using grid [[]]	196
13.3	Group box [^]	197
13.4	Using separator [..., ==, ~~, --]	197
13.5	Tree widget [T]	198
13.6	Tree table [T]	198
13.7	Enclosing brackets [{, }]	200
13.8	Adding tabs [/]	200
13.9	Using menu [*]	201
13.10	Advanced table	202
13.11	Scroll Bars [S, SI, S-]	202
13.12	Colors	203
13.13	Pseudo sprite [<<, >>]	204
13.14	OpenIconic	204
13.15	Include Salt "on activity diagram"	205
13.16	Include salt "on while condition of activity diagram"	208
<b>14</b>	<b>Archimate Diagram</b>	<b>209</b>
14.1	Archimate keyword	209
14.2	Defining Junctions	209
14.3	Example 1	210
14.4	Example 2	212
14.5	List possible sprites	212
14.6	ArchiMate Macros	212



<b>15 Gantt Diagram</b>	<b>214</b>
15.1 Declaring tasks	214
15.1.1 Duration	214
15.1.2 Start	214
15.1.3 End	214
15.1.4 Start/End	215
15.2 One-line declaration (with the and conjunction)	215
15.3 Adding constraints	215
15.4 Short names	216
15.5 Customize colors	216
15.6 Completion status	216
15.7 Milestone	217
15.7.1 Relative milestone (use of constraints)	217
15.7.2 Absolute milestone (use of fixed date)	217
15.7.3 Milestone of maximum end of tasks	217
15.8 Hyperlinks	218
15.9 Calendar	218
15.10 Coloring days	218
15.11 Changing scale	219
15.11.1 Daily ( <i>by default</i> )	219
15.11.2 Weekly	219
15.11.3 Monthly	220
15.12 Close day	220
15.13 Simplified task succession	221
15.14 Separator	222
15.15 Working with resources	222
15.16 Complex example	223
15.17 Comments	223
15.18 Using style	223
15.19 Add notes	224
15.20 Pause tasks	227
15.21 Change link colors	227
15.22 Tasks or Milestones on the same line	228
15.23 Highlight today	228
15.24 Task between two milestones	228
15.25 Grammar and verbal form	229
15.26 Add title, header, footer, caption or legend on gantt diagram	229
15.27 Removing Foot Boxes	229
<b>16 MindMap</b>	<b>232</b>
16.1 OrgMode syntax	232
16.2 Multilines	232
16.3 Colors	233
16.3.1 With inline color	233
16.3.2 With style color	234
16.4 Removing box	235
16.5 Arithmetic notation	235
16.6 Markdown syntax	236
16.7 Changing style	236
16.8 Changing diagram direction	237
16.9 Complete example	238
16.10 Word Wrap	239
<b>17 Work Breakdown Structure (WBS)</b>	<b>241</b>
17.1 OrgMode syntax	241
17.2 Change direction	241
17.3 Arithmetic notation	242
17.4 Removing box	243
17.5 Colors (with inline or style color)	243





17.6	Using style	244
17.7	Word Wrap	245
<b>18</b>	<b>Maths</b>	<b>247</b>
18.1	Standalone diagram	247
18.2	How is this working?	248
<b>19</b>	<b>Entity Relationship Diagram</b>	<b>249</b>
19.1	Information Engineering Relations	249
19.2	Entities	249
19.3	Complete Example	250
<b>20</b>	<b>Common commands</b>	<b>252</b>
20.1	Comments	252
20.2	Footer and header	252
20.3	Zoom	252
20.4	Title	253
20.5	Caption	254
20.6	Legend the diagram	254
20.7	Appendice: Examples on all diagram	255
20.7.1	Activity	255
20.7.2	Archimate	256
20.7.3	Class	256
20.7.4	Component, Deployment, Use-Case	257
20.7.5	Gantt project planning	257
20.7.6	Object	258
20.7.7	MindMap	258
20.7.8	Network (nwdiag)	259
20.7.9	Sequence	260
20.7.10	State	260
20.7.11	Timing	261
20.7.12	Work Breakdown Structure (WBS)	262
20.7.13	Wireframe (SALT)	262
20.8	Appendice: Examples on all diagram with style	263
20.8.1	Activity	264
20.8.2	Archimate	265
20.8.3	Class	267
20.8.4	Component, Deployment, Use-Case	268
20.8.5	Gantt project planning	269
20.8.6	Object	270
20.8.7	MindMap	272
20.8.8	Network (nwdiag)	273
20.8.9	Sequence	275
20.8.10	State	276
20.8.11	Timing	277
20.8.12	Work Breakdown Structure (WBS)	279
20.8.13	Wireframe (SALT)	280
<b>21</b>	<b>Creole</b>	<b>282</b>
21.1	Emphasized text	282
21.2	List	282
21.3	Escape character	283
21.4	Horizontal lines	283
21.5	Headings	284
21.6	Legacy HTML	284
21.7	Code	285
21.8	Table	286
21.8.1	Build a table	286
21.8.2	Add color on cells or lines	287



21.8.3	Add color on border	287
21.8.4	No border or same color as the background	287
21.8.5	Bold header or not	288
21.9	Tree	288
21.10	Special characters	290
21.11	OpenIconic	290
21.12	Appendix: Examples of "Creole List" on all diagrams	291
21.12.1	Activity	291
21.12.2	Class	292
21.12.3	Component, Deployment, Use-Case	293
21.12.4	Gantt project planning	294
21.12.5	Object	294
21.12.6	MindMap	295
21.12.7	Network (nwdiag)	295
21.12.8	Note	295
21.12.9	Sequence	296
21.12.10	State	296
21.13	Appendix: Examples of "Creole horizontal lines" on all diagrams	296
21.13.1	Activity	296
21.13.2	Class	297
21.13.3	Component, Deployment, Use-Case	298
21.13.4	Gantt project planning	298
21.13.5	Object	298
21.13.6	MindMap	299
21.13.7	Network (nwdiag)	300
21.13.8	Note	300
21.13.9	Sequence	300
21.13.10	State	301
21.14	Style equivalent (between Creole and HTML)	301
<b>22</b>	<b>Defining and using sprites</b>	<b>303</b>
22.1	Changing colors	304
22.2	Encoding Sprite	304
22.3	Importing Sprite	304
22.4	Examples	304
22.5	StdLib	305
22.6	Listing Sprites	306
<b>23</b>	<b>Skinparam command</b>	<b>307</b>
23.1	Usage	307
23.2	Nested	307
23.3	Black and White	307
23.4	Shadowing	308
23.5	Reverse colors	308
23.6	Colors	309
23.7	Font color, name and size	310
23.8	Text Alignment	310
23.9	Examples	311
23.10	List of all skinparam parameters	314
<b>24</b>	<b>Preprocessing</b>	<b>317</b>
24.1	Migration notes	317
24.2	Variable definition	317
24.3	Boolean expression	318
24.3.1	Boolean representation [0 is false]	318
24.3.2	Boolean operation and operator [&&,   , ()]	318
24.3.3	Boolean builtin functions [%false(), %true(), %not(<exp>)]	318
24.4	Conditions [!if, !else, !elseif, !endif]	318
24.5	While loop [!while, !endwhile]	319



24.6	Procedure [!procedure, !endprocedure]	320
24.7	Return function [!function, !endfunction]	321
24.8	Default argument value	322
24.9	Unquoted procedure or function [!unquoted]	323
24.10	Keywords arguments	323
24.11	Including files or URL [!include, !include_many, !include_once]	324
24.12	Including Subpart [!startsub, !endsub, !includesub]	325
24.13	Builtin functions [%]	325
24.14	Logging [!log]	326
24.15	Memory dump [!memory_dump]	326
24.16	Assertion [!assert]	327
24.17	Building custom library [!import, !include]	327
24.18	Search path	328
24.19	Argument concatenation [##]	328
24.20	Dynamic invocation [%invoke_procedure(), %call_user_func()]	328
24.21	Evaluation of addition depending of data types [+]	329
24.22	Preprocessing JSON	330
<b>25</b>	<b>Unicode</b>	<b>331</b>
25.1	Examples	331
25.2	Charset	333
<b>26</b>	<b>Standard Library</b>	<b>334</b>
26.1	Amazon Labs Library	334
26.2	AWS library	334
26.3	Azure library	335
26.4	Cloud Insight	336
26.5	Elastic library	337
26.6	Tupadr3 library	337
26.7	Google Material Icons	339
26.8	Office	340
26.9	ArchiMate	341
26.10	Kubernetes	343
26.11	Miscellaneous	344