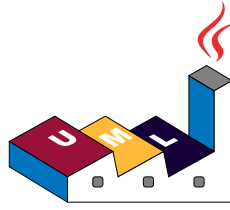


# PlantUML 을사용해서 UML 그리기



## PlantUML 언어참조가이드

(Version 1.2019.9)

**PlantUML** 은 다이어그램을 빠르게 작성하기 위한 오픈소스 프로젝트입니다.

- 시퀀스 다이어그램
- 유즈 케이스 다이어그램
- 클래스 다이어그램
- 액티비티 다이어그램
- 컴포넌트 다이어그램
- 상태 다이어그램
- 객체 다이어그램
- Deployment diagram
- Timing diagram

다음의 UML 이외의 다이어그램도 지원함:

- 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

간단하고 직관적인 언어를 사용해 다이어그램을 정의할 수 있습니다.

# 1 시퀀스 다이어그램

## 1.1 기본예제

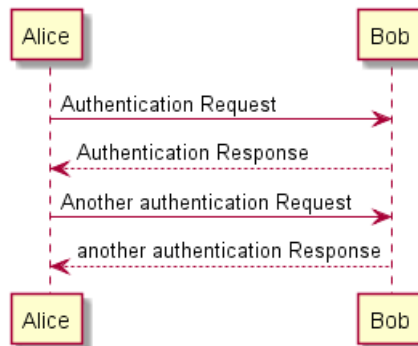
시퀀스 -> 는 두 참여자들 사이의 메시지를 그리기 위해 사용된다. 참여자들은 명시적으로 선언하지 않아도 된다.

점선화살표를 만들기 위해서는 --> 를 사용한다.

또한 <-과 <--를 사용할 수 있다. 출력되는 그림은 변경되지 않지만, 가독성을 향상 시키는데 사용할 수 있다. 이는 시퀀스 다이어그램에만 적용되며, 다른 다이어그램에는 다른 규칙이 적용된다.

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

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



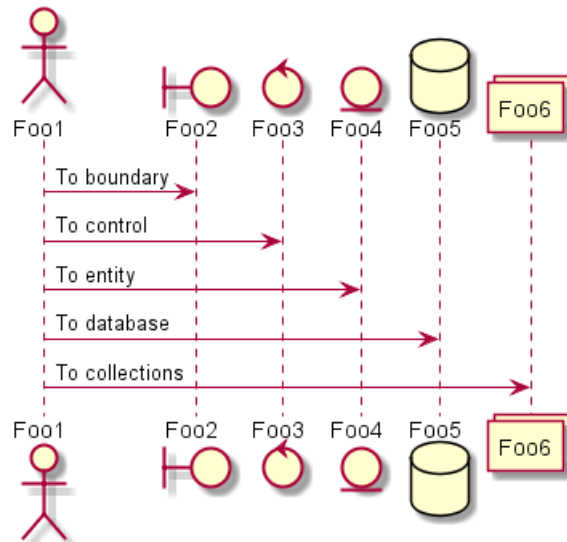
## 1.2 참여자 (participant) 선언

participant 키워드를 이용하여 참여자의 순서를 바꿀 수 있다.

또한, 참여자 선언에 다음과 같은 키워드를 사용할 수 있다.

- actor
- boundary
- control
- entity
- database

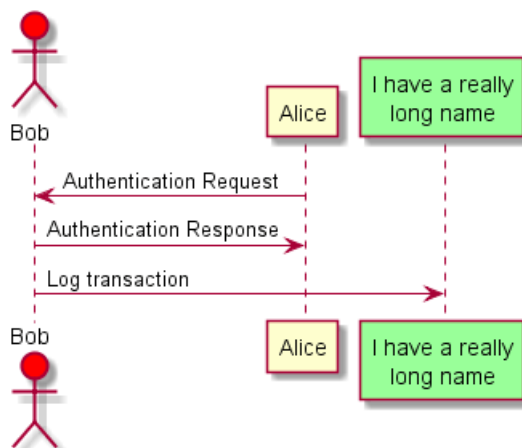
```
@startuml
actor Foo1
boundary Foo2
control Foo3
entity Foo4
database Foo5
collections Foo6
Foo1 -> Foo2 : To boundary
Foo1 -> Foo3 : To control
Foo1 -> Foo4 : To entity
Foo1 -> Foo5 : To database
Foo1 -> Foo6 : To collections
@enduml
```



as 키워드를 이용하여 참여자의 이름을 변경할 수 있다.  
 또한, 참여자 (actor, participant) 의 배경색을 변경할 수도 있다.

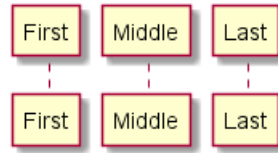
```
@startuml
actor Bob #red
' The only difference between actor
'and participant is the drawing
participant Alice
participant "I have a really\nlong name" as L #99FF99
/' You can also declare:
    participant L as "I have a really\nlong name" #99FF99
    '/
```

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



order 키워드를 이용하여, 참여자의 출력 순서를 지정할 수 있다.

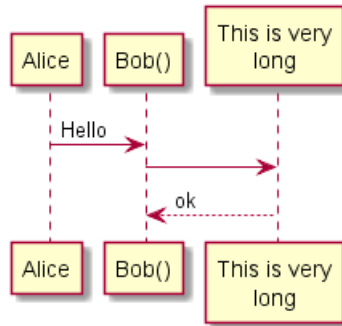
```
@startuml
participant Last order 30
participant Middle order 20
participant First order 10
@enduml
```



### 1.3 참여자에서특수문자사용하기

따옴표를 사용하여 참여자를 정의할 수 있다. 그리고 "as" 키워드를 사용하여 참여자를 별칭으로 사용할 수도 있다.

```
@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
```

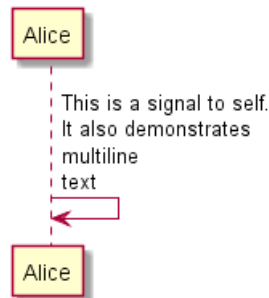


### 1.4 자신에게메시지보내기

참여자 는 자기 자신에게 메시지를 보낼 수 있다.

\n 을 이용해서 여러 줄로 쓰는 것도 가능하다

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



### 1.5 화살표스타일변경

다음 방법으로 화살표 스타일을 바꿀 수 있다:

- 끝부분에 x 를 추가하여 메시지가 전달되지 않았음을 표시할 수 있다.
- < 나 > 대신에 \ 나 / 를 사용해서
- 아래쪽이나 위쪽 화살표만 표시한다.

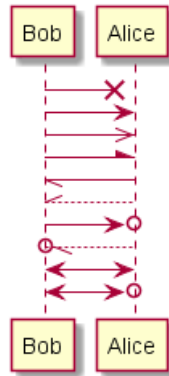
- {\$>\$} 를두번사용하여화살표모양을얇게표시할수있다. (예. >>)
- - 대신 -- 를사용해서점선화살표를표시한다.
- 화살표다음에"o" 추가도가능하다.
- 양쪽끝에화살표추가도가능하다.

```

@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
    
```

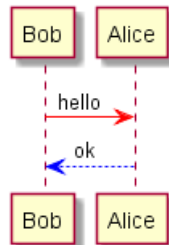


### 1.6 화살표색상변경

다음의표기법을이용해서각각화살표의색상을바꿀수있다.

```

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



### 1.7 메시지순서에번호매기기

autonumber 키워드는메시지에자동으로번호를매길때에사용한다.

```

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

```
Bob <- Alice : Authentication Response
@enduml
```



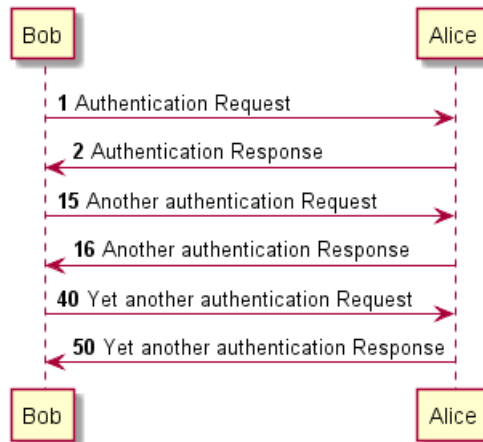
autonumber 시작번호의형태로 표시하면 특정번호로 시작할 수 있으며, autonumber 시작번호 증가값으로 표시할 경우 증가값을 조정하는 것도 가능하다.

```
@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
```



쌍따옴표를 이용하여 표시형식을 바꿀 수도 있다.

표시형식은 자바 클래스 DecimalFormat 을 사용한다. (0 은 숫자를 의미하며, # 은 숫자로 표시하되, 빈자리 이면 0 으로 채우라는 뜻이다).

몇가지 html 태그를 사용할 수 있다.

```
@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
```

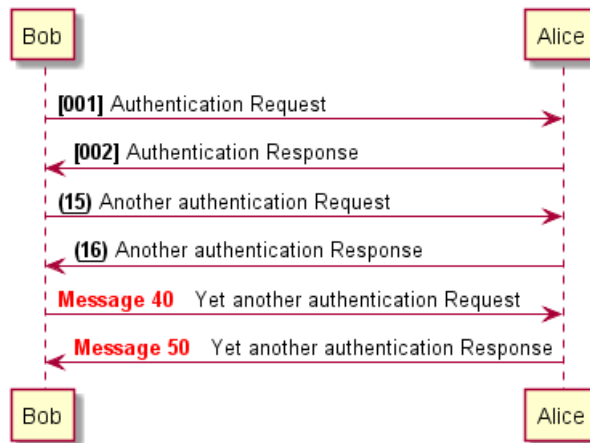


```

Bob <- Alice : Yet another authentication Response

@enduml

```



또한, `autonumber stop` 키워드를 이용하여 번호매김을 일시정지할 수 있으며, `autonumber resume` 증가값 표시형식 키워드를 이용하여 계속해서 번호를 매길 수 있다.

```

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

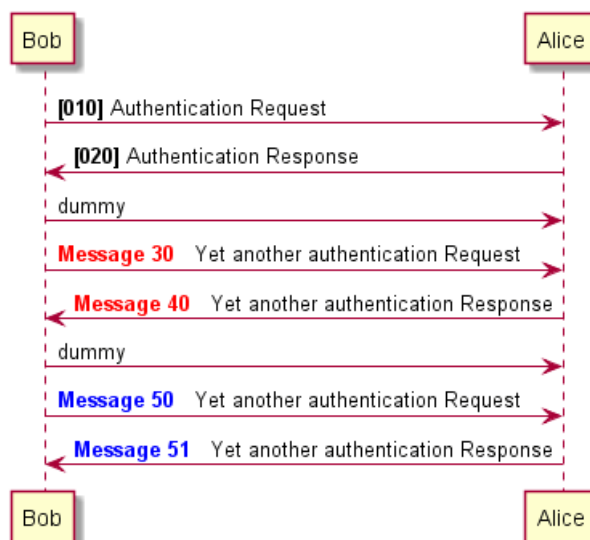
autonumber stop
Bob -> Alice : dummy

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

```



## 1.8 페이지제목, 머리말과꼬리말

title 키워드를 이용하여 페이지에 제목을 추가할 수 있다.

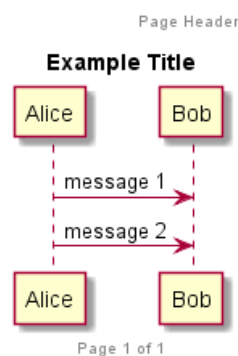
또한, header 와 footer 를 이용하여, 각각 머리말과 꼬리말을 표시할 수도 있다.

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

title Example Title

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

@enduml
```



## 1.9 다이어그램 분리

newpage 키워드를 이용하여, 다이어그램을 여러개의 이미지로 분리할 수 있다.

newpage 키워드 뒤에 바로 새로 생성되는 페이지의 제목을 넣을 수 있다.

여러 페이지에 걸쳐있는 긴 다이어그램을 출력할 때 유용하다.

(주: 예제에서 첫 번째 페이지만 표시되었지만, 실제로 잘 동작하는 기능이다.)

```
@startuml

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

newpage

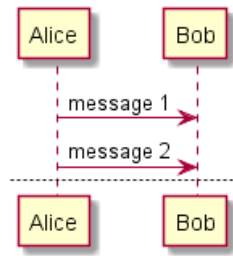
Alice -> Bob : message 3
Alice -> Bob : message 4

newpage A title for the\nlast page

Alice -> Bob : message 5
Alice -> Bob : message 6

@enduml
```





## 1.10 메세지그룹화

다음과같은키워드들을사용하여메세지를그룹화할수있다:

- alt/else
- opt
- loop
- par
- break
- critical
- group, 화면에보여질텍스트

헤더에표시될텍스트를추가할수있다. (group 제외).

end 키워드는그룹을닫는데사용한다.

또한, 그룹을중첩해서만들수도있다.

```

@startuml
Alice -> Bob: Authentication Request

alt successful case

Bob -> Alice: Authentication Accepted

else some kind of failure

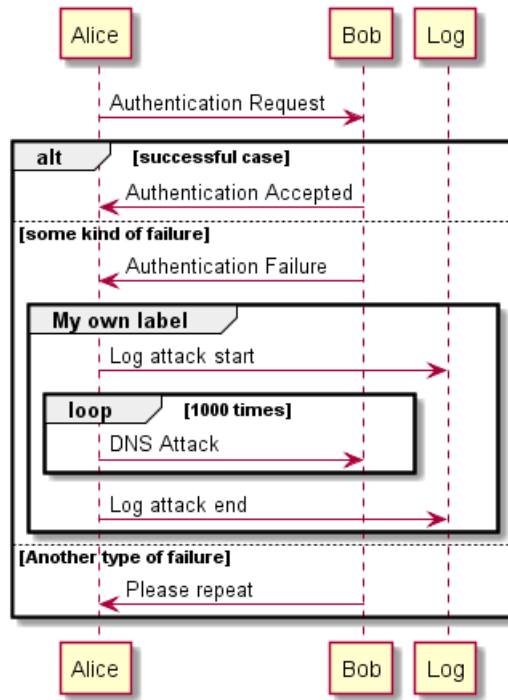
Bob -> Alice: Authentication Failure
group My own label
Alice -> Log : Log attack start
  loop 1000 times
    Alice -> Bob: DNS Attack
  end
Alice -> Log : Log attack end
end

else Another type of failure

  Bob -> Alice: Please repeat

end
@enduml

```



### 1.11 메시지에노트추가하기

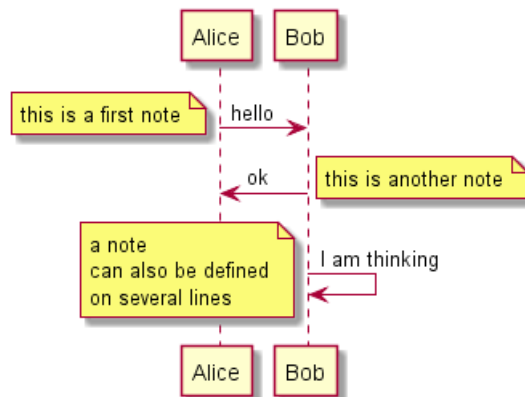
메시지다음에 `note left` 나 `note right` 키워드를 이용하여, 메시지에노트를추가할수있다. 또한, 한번에여러줄의노트를추가하는경우에는 `end note` 를 이용하여, 노트의끝을표시해주어야한다.

```

@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
    
```



## 1.12 다른형태의노트들

note left of, note right of, note over 키워드를 이용하여 참여자의 상대적인 위치에 노트를 추가할 수도 있다.

노트의 배경색을 변경함으로써, 노트를 강조하는 것도 가능하다.

한번에 여러 줄의 노트를 추가하는 경우에는, end note 를 이용하여 노트의 끝을 표시해주어야 한다.

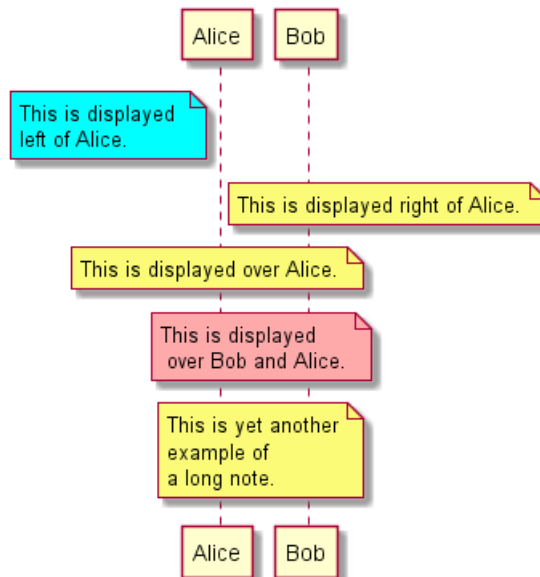
```
@startuml
participant Alice
participant Bob
note left of Alice #aqua
This is displayed
left of Alice.
end note
```

```
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
```

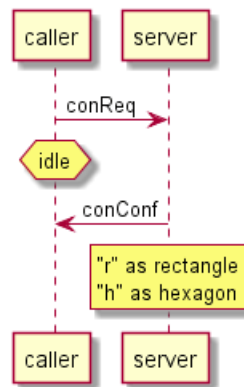


## 1.13 노트모양바꾸기

hnote 와 rnote 키워드를 이용하여, 노트의 모양을 바꿀 수 있다.

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

```
endrnote
@enduml
```



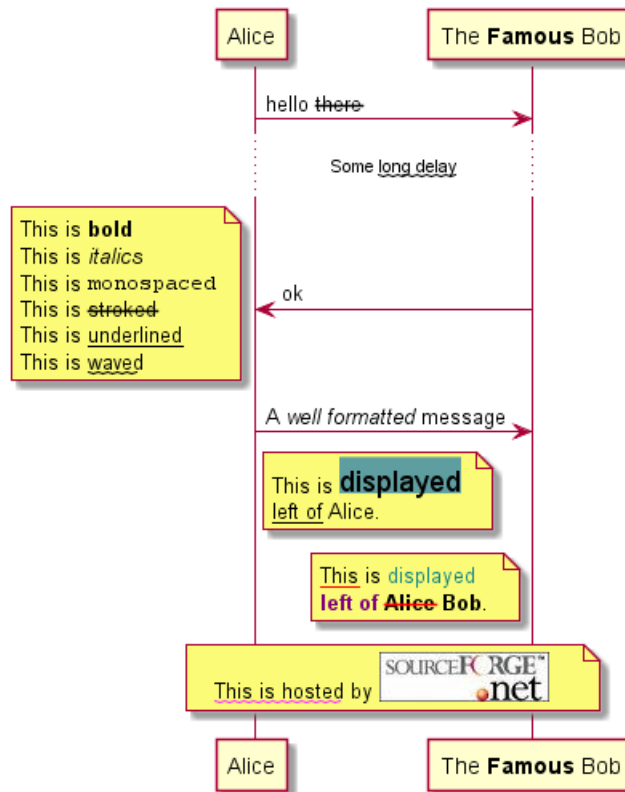
## 1.14 Creole 과 HTML

creole 문법을 사용할 수도 있다:

```
@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
```



## 1.15 구분자

== 구분자를 이용하여, 다이어그램을 논리적인 단계로 구분하여 나눌 수 있다.

```
@startuml
```

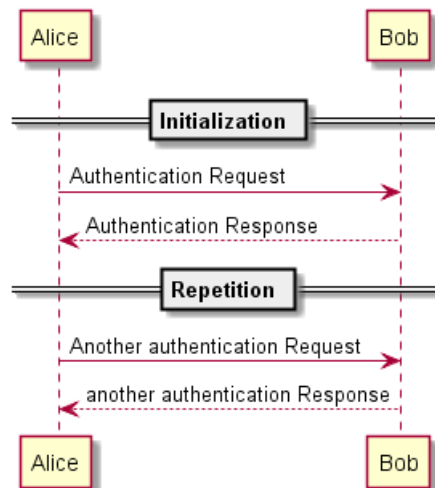
```
== Initialization ==
```

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

```
== Repetition ==
```

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

```
@enduml
```



## 1.16 참조

ref over 키워드를 이용하여, 다이어그램에 참조를 표시할 수 있다.

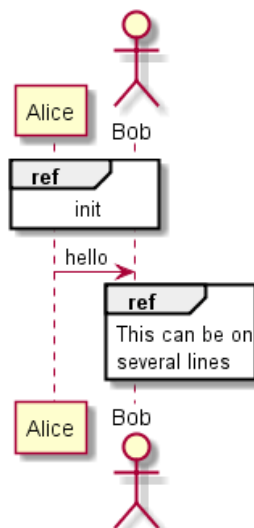
```

@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
  
```



## 1.17 지연

... 을 이용하여, 다이어그램에 지연상태를 나타낼 수 있으며, 그 위에 메시지를 추가할 수도 있다.

```

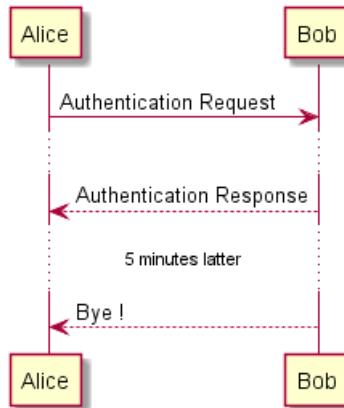
@startuml
  
```

```

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

@enduml

```



## 1.18 공백

||| 을 이용하여 다이어그램에 공백을 나타낼 수 있으며, 공백에 얼마만큼의 픽셀을 사용할 것인지 숫자로 명시할 수도 있다.

```

@startuml

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

@enduml

```



## 1.19 생명선활성화및비활성화

activate 와 deactivate 는 참여자의 활성화 여부를 표현하는데 사용한다.

참여자가 활성화되면, 참여자의 생명선이 나타난다.

activate 와 deactivate 는 바로 이전의 메시지에 적용된다.

destroy 는 참여자의 생명선이 끝났음을 표현한다.

```

@startuml
participant User

User -> A: DoWork
activate A

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

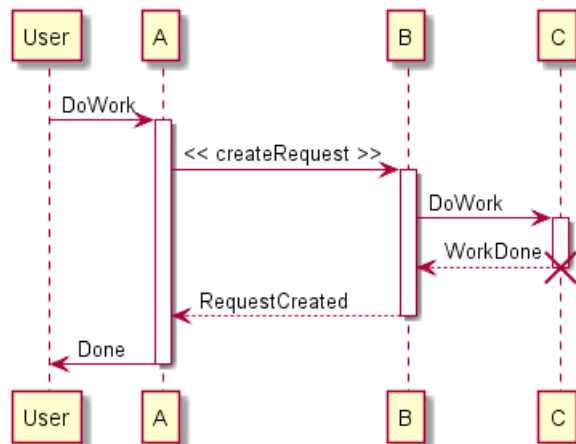
B -> C: DoWork
activate C
C --> B: WorkDone
destroy C

B --> A: RequestCreated
deactivate B

A -> User: Done
deactivate A

@enduml

```



생명선은 중첩해서 사용할 수 있으며, 생명선에 색을 넣을 수도 있다.

```

@startuml
participant User

User -> A: DoWork
activate A #FFBBBB

A -> A: Internal call
activate A #DarkSalmon

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

B --> A: RequestCreated

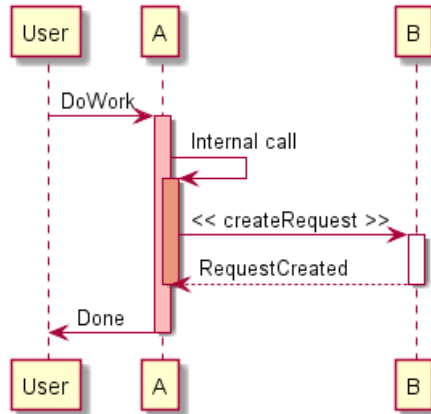
```





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

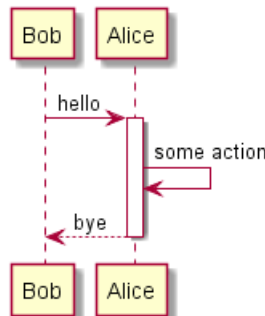
@enduml



### 1.20 리턴

리턴메시지를 생성하는 return 명령이 추가되었다. 리턴되는 지점은 가장 최근에 생명선을 활성화 시킨 지점의 출발점이다. 문법은 간단히 return 꼬리표이며, 꼬리표는 기존의 메시지와 마찬가지로 임의의 문자열을 쓸 수 있다.

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



### 1.21 참여자 생성

해당 메시지가 실제로 객체를 생성한다는 걸 강조하기 위해, 참여자가 첫 번째 메시지를 수신하기 전에 create 키워드를 사용할 수 있다.

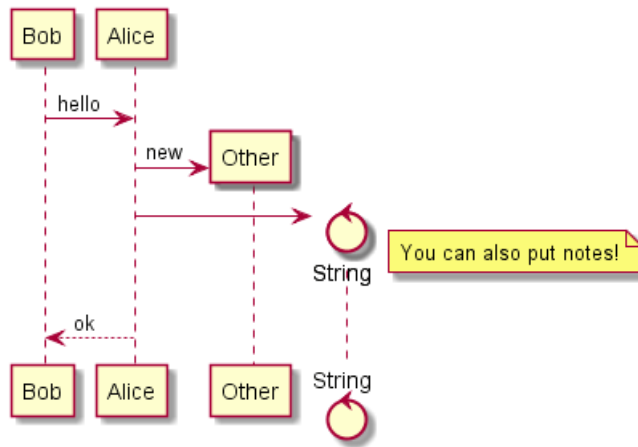
```
@startuml
Bob -> Alice : hello

create Other
Alice -> Other : new

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

```
Alice --> Bob : ok
```

```
@enduml
```



## 1.22 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 >>
```

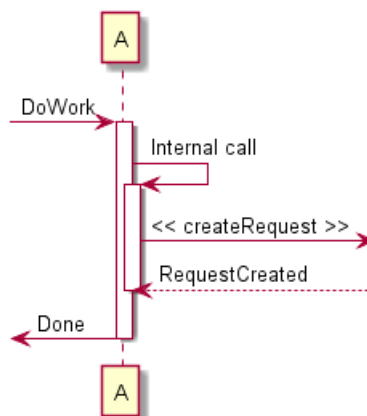
```
A<--] : RequestCreated
```

```
deactivate A
```

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

```
deactivate A
```

```
@enduml
```



You can also have the following syntax:

```
@startuml
```

```
[-> Bob
```



```

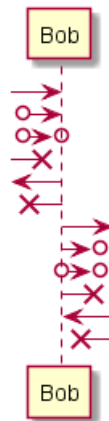
[o-> Bob
[o->o Bob
[x-> Bob

[<- Bob
[x<- Bob

Bob ->]
Bob ->o]
Bob o->o]
Bob ->x]

Bob <-]
Bob x<-]
@enduml

```



### 1.23 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).

```

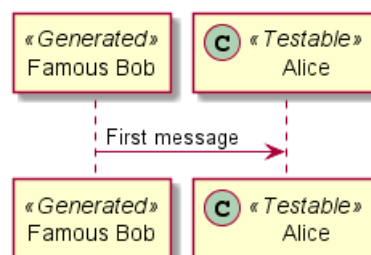
@startuml

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

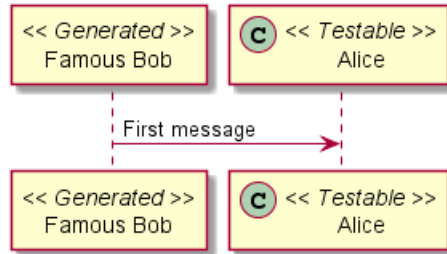
```



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

Bob->Alice: First message

@enduml

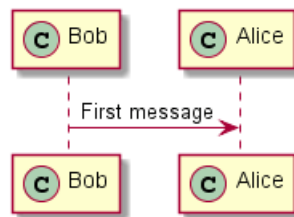


@startuml

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

Bob->Alice: First message

@enduml



### 1.24 More information on titles

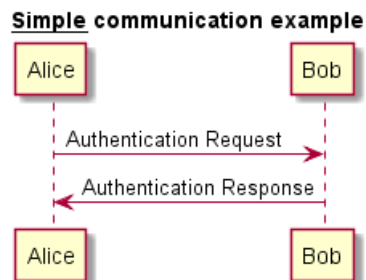
You can use creole formatting in the title.

@startuml

```
title __Simple__ **communication** example
```

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

@enduml



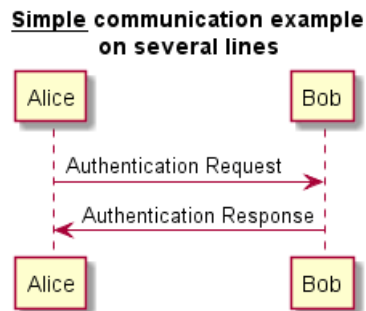
You can add newline using \n in the title description.

@startuml

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

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

```
@enduml
```



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

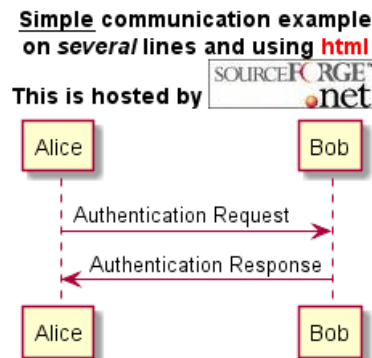
```
@startuml
```

```
title
```

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

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

```
@enduml
```



## 1.25 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.

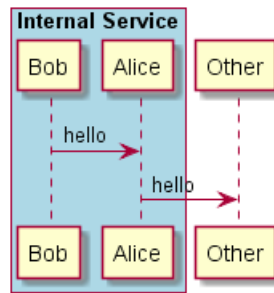
```
@startuml
```

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

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



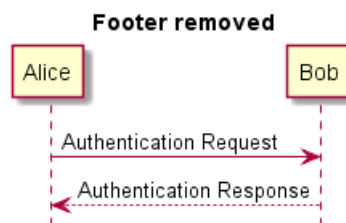
```
Alice -> Other : hello
@enduml
```



## 1.26 Removing Footer

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

```
@startuml
hide footbox
title Footer removed
Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response
@enduml
```



## 1.27 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

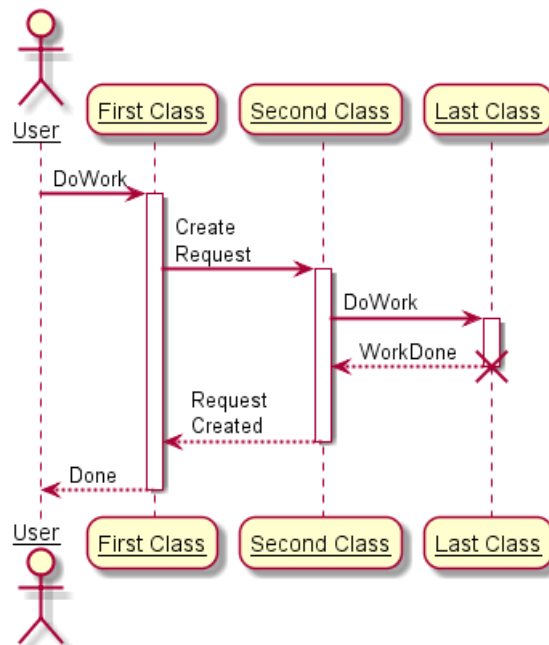
B -> C: DoWork
activate C
C --> B: WorkDone
destroy C

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml

```



```

@startuml
skinparam backgroundColor #EEEEBC
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

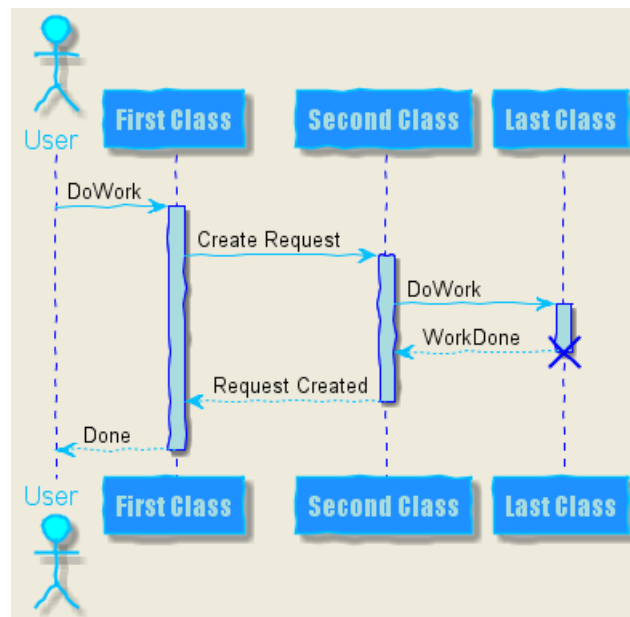
B -> C: DoWork
activate C
C --> B: WorkDone
destroy C

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml

```



## 1.28 Changing padding

It is possible to tune some padding settings.

```

@startuml
skinparam ParticipantPadding 20
skinparam BoxPadding 10

box "Foo1"
participant Alice1

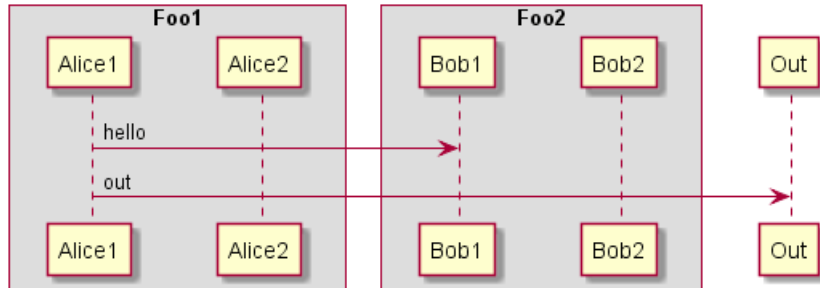
```



```

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

```



## 2 유즈케이스 다이어그램

Let's have few examples :

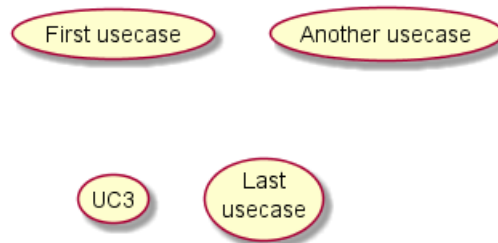
Note that you can disable the shadowing using the skinparam shadowing false command.

### 2.1 유즈케이스

유즈케이스는 (두개의괄호는원어로보이기때문에) 괄호로싸서표현합니다.

아니면 usecase 키워를이용하여정의할수있습니다. 그리고 as 키워드를이용하여별칭을정의할수있습니다. 별칭은관계를정의할때사용됩니다.

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



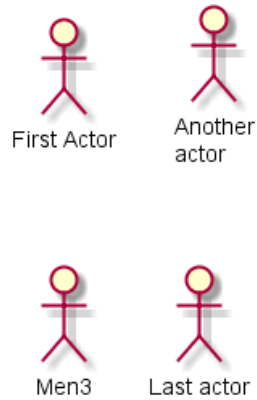
### 2.2 Actors

Actor are enclosed using between two points.

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

We will see later that the actor definitions are optional.

```
@startuml
:First Actor:
:Another\nactor: as Men2
actor Men3
actor :Last actor: as Men4
@enduml
```



## 2.3 유즈케이스종류

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

You can also use the following separators: -- .. == \_\_. And you can put titles within the separators.

```
@startuml
```

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

```
--
```

```
Several separators are possible.
```

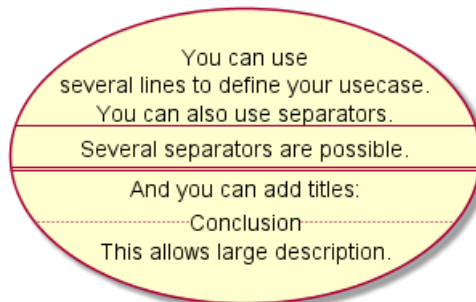
```
==
```

```
And you can add titles:
```

```
..Conclusion..
```

```
This allows large description."
```

```
@enduml
```



## 2.4 기본예제

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
```

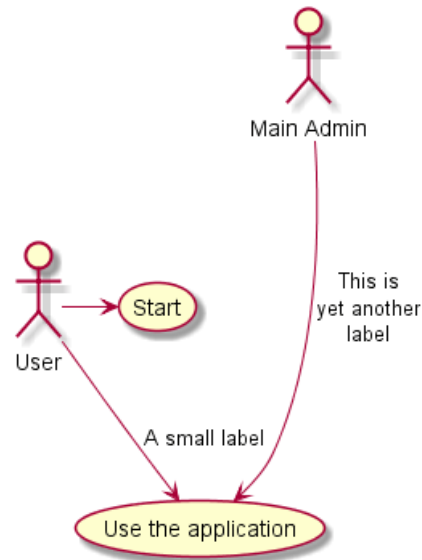
```
User -> (Start)
```

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

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



```
@enduml
```



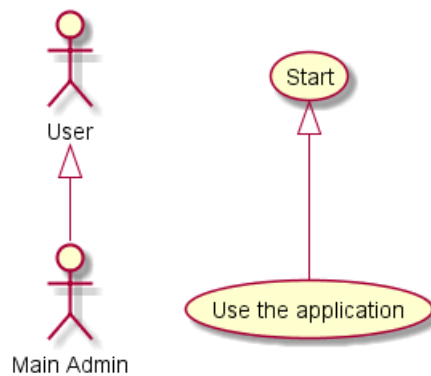
## 2.5 Extension

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
```



## 2.6 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)
```



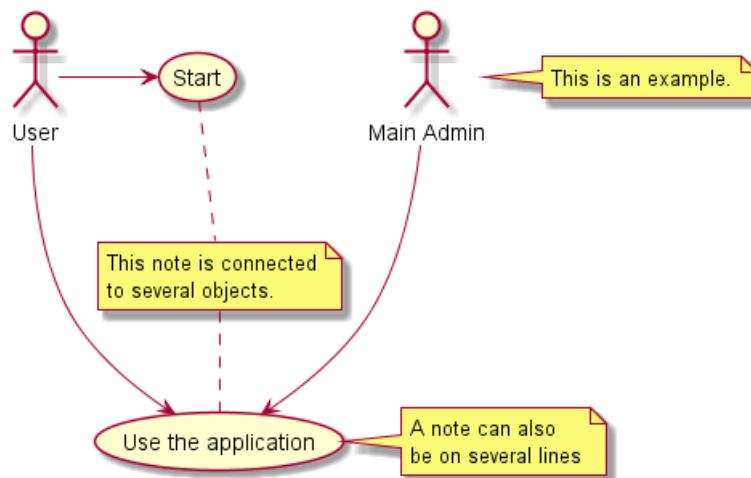
```
User -> (Start)
User --> (Use)
```

```
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
```



## 2.7 Stereotypes

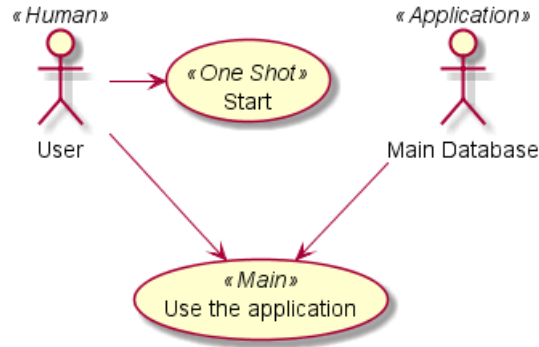
You can add stereotypes while defining actors and use cases using << and >>.

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

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

```
MySQL --> (Use)
```

```
@enduml
```

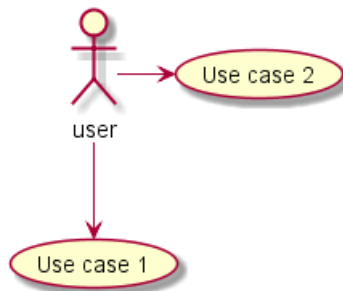


### 2.8 화살표방향변경

기본적으로, 클래스간의 링크는 대쉬 2 개 -- 로 표시하고 수직방향이다. 다음처럼 대쉬 1 개 (혹은 점) 을 넣어서 수평방향 링크를 사용할 수 있다:

```

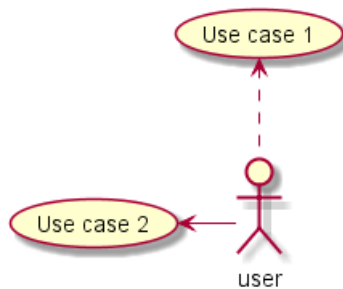
@startuml
:user: --> (Use case 1)
:user: -> (Use case 2)
@enduml
  
```



You can also change directions by reversing the link:

```

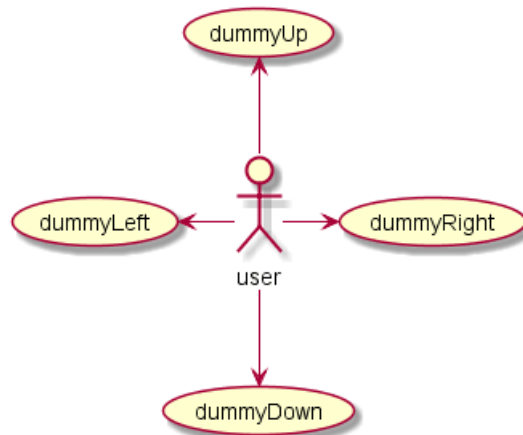
@startuml
(Use case 1) <.. :user:
(Use case 2) <- :user:
@enduml
  
```



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
  
```



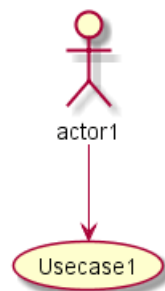
You can shorten the arrow by using only the first character of the direction (for example, `-d-` instead of `-down-`) or the two first characters (`-do-`).

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

## 2.9 Splitting diagrams

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

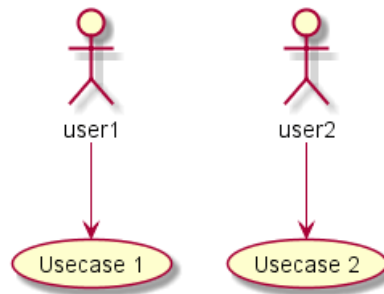
```
@startuml
:actor1: --> (Usecase1)
newpage
:actor2: --> (Usecase2)
@enduml
```



## 2.10 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
```



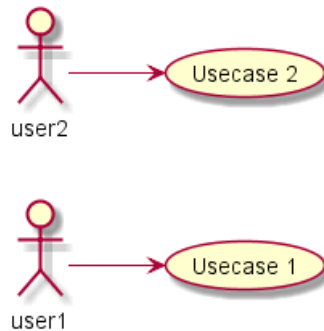
You may change to **left to right** using the left to right direction command. The result is often better with this direction.

```

@startuml

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

@enduml
  
```



## 2.11 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
  
```





```

}

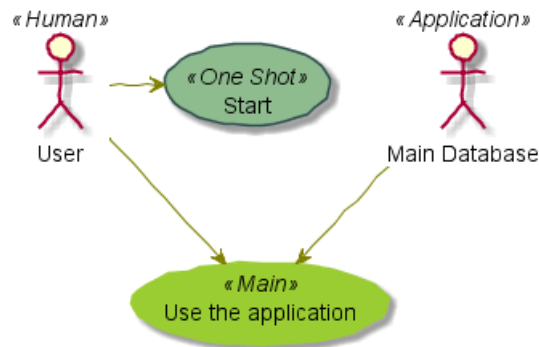
User << Human >>
:Main Database: as MySQL << Application >>
(Start) << One Shot >>
(Use the application) as (Use) << Main >>

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

MySQL --> (Use)

@enduml

```

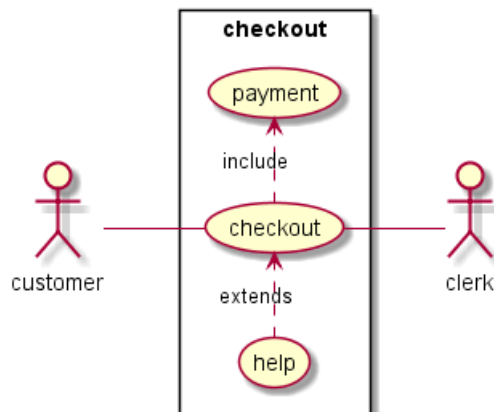


## 2.12 Complete example

```

@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 클래스 다이어그램

### 3.1 클래스관계

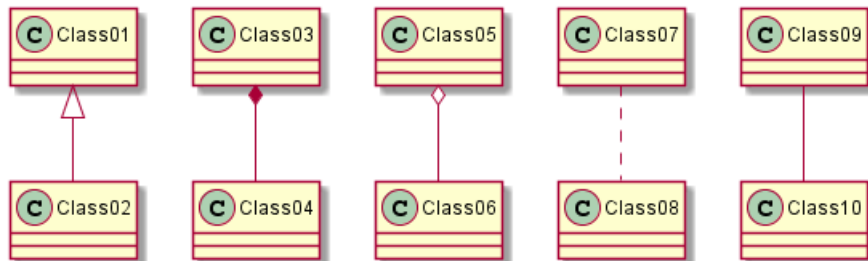
클래스관계는 다음과 같은 부호를 사용합니다.

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

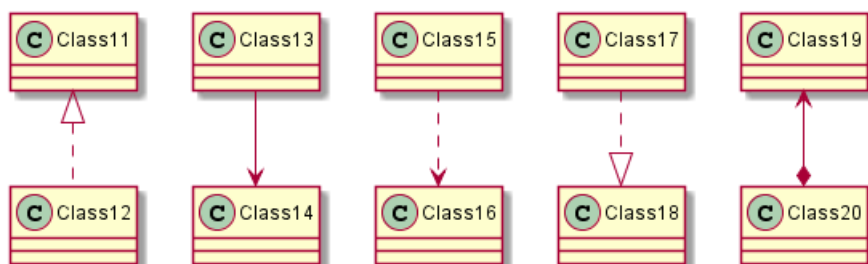
-- 는 .. 점선으로 대체 가능합니다.

이러한 규칙들로 다음과 같은 다이어그램을 그리는 것이 가능합니다.

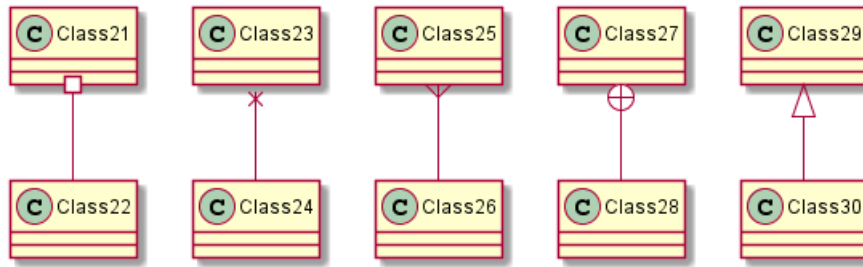
```
@startuml
Class01 <|-- Class02
Class03 *-- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10
@enduml
```



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



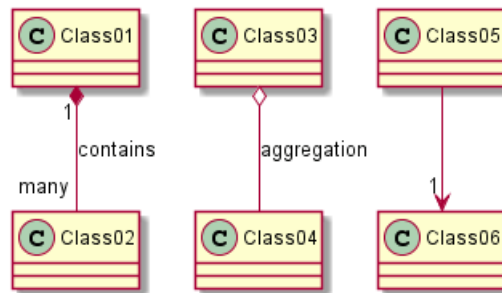
```
@startuml
Class21 #-- Class22
Class23 x-- Class24
Class25 }-- Class26
Class27 +-- Class28
Class29 ^-- Class30
@enduml
```



### 3.2 관계를 나타내기 위한 레이블

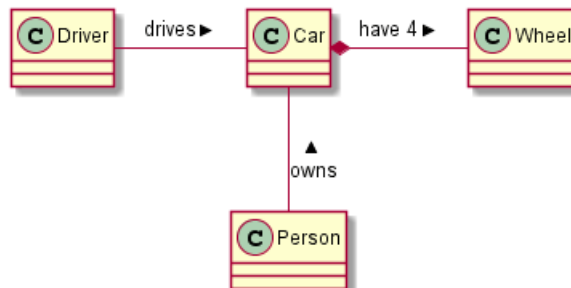
관계에서 레이블을 추가하기 위해서는 뒤에 : 를 붙이고 레이블을 작성하면 됩니다.  
 관계 차수를 나타내기 위해서는 " " 를 이용하여 관계의 양쪽 끝에 작성하면 됩니다.

```
@startuml
Class01 "1" *-- "many" Class02 : contains
Class03 o-- Class04 : aggregation
Class05 --> "1" Class06
@enduml
```



< 또는 > 을 사용하여 객체가 다른 객체에 대한 흐름 관계를 더 자세히 설명할 수 있습니다.

```
@startuml
class Car
Driver - Car : drives >
Car *- Wheel : have 4 >
Car -- Person : < owns
@enduml
```



### 3.3 Adding methods

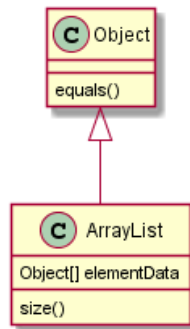
To declare fields and methods, you can use the symbol `:` followed by the field's or method's name.

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

```
@startuml
Object <|-- ArrayList

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

@enduml
```



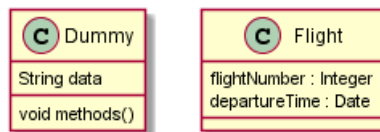
It is also possible to group between brackets `{}` all fields and methods.

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

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

class Flight {
    flightNumber : Integer
    departureTime : Date
}

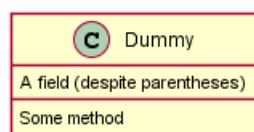
@enduml
```



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
```

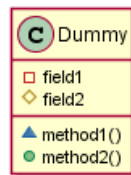


### 3.4 Defining visibility

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

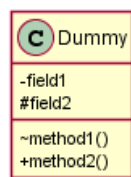
Character	Icon for field	Icon for method	Visibility
-	□	■	private
#	◇	◆	protected
~	△	▲	package private
+	○	●	public

```
@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
```



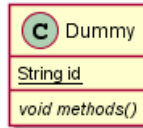
### 3.5 Abstract and Static

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
```





### 3.6 Advanced class body

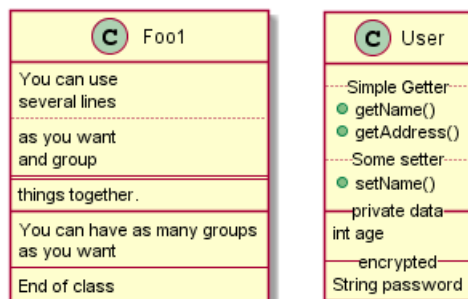
By default, methods and fields are automatically regrouped by PlantUML. You can use separators to define your own way of ordering fields and methods. The following separators are possible : -- .. == \_\_.

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
```



### 3.7 Notes and stereotypes

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



You can also define notes using `note left of`, `<code>note right of</code>`, `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.

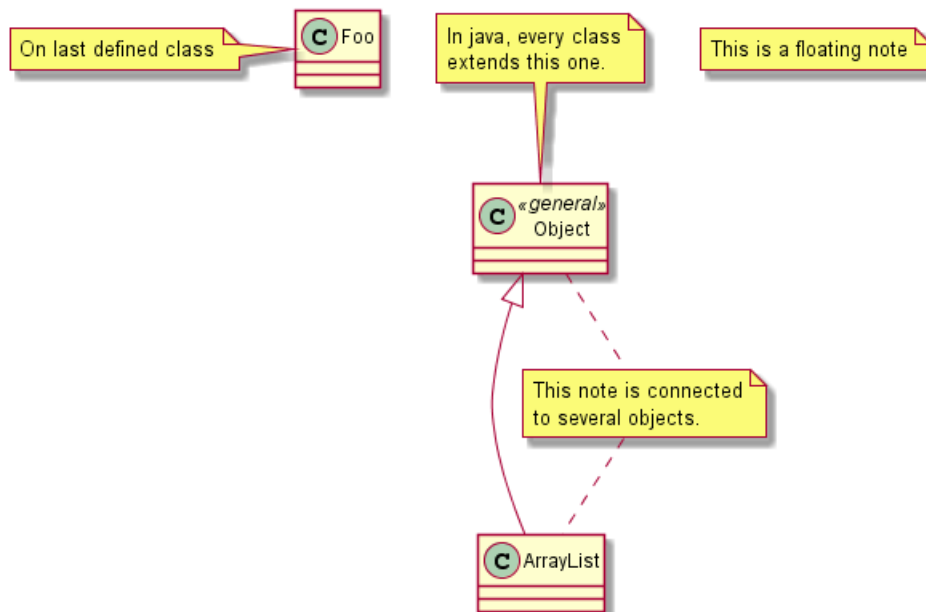
```
@startuml
class Object << general >>
Object <|--- ArrayList

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
```



### 3.8 More on notes

It is also possible to use few html tags 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`.

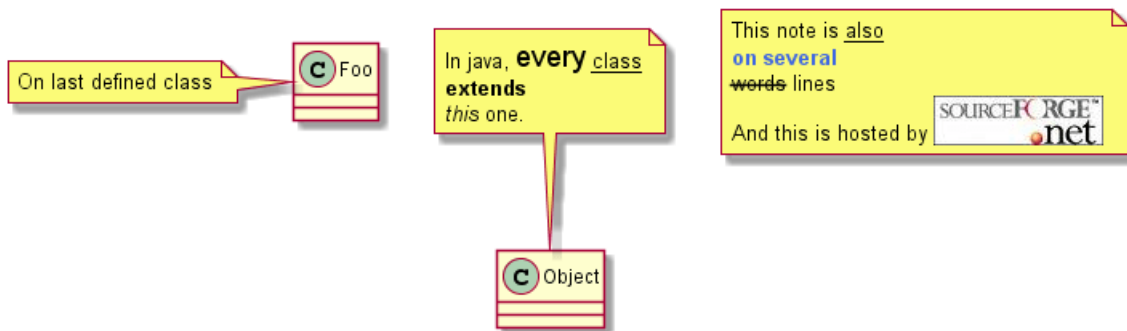
```
@startuml

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
```



### 3.9 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.

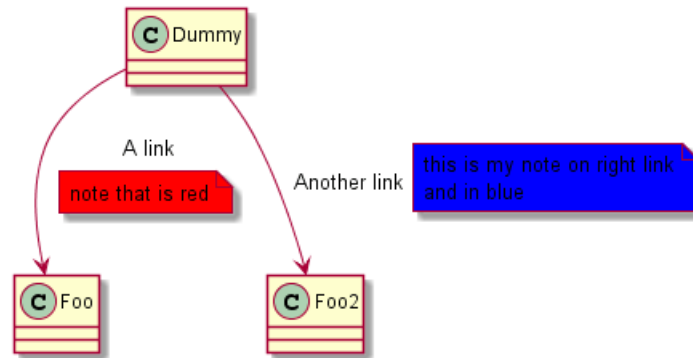
```
@startuml

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
```





### 3.10 Abstract class and interface

<a name="enum" id="enum">You can declare a class as abstract using "abstract" or "abstract class" keywords.</a>

The class will be printed in *italic*.</p>

You can use the interface, annotation and enum keywords too.</p>

@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()
}
  
```

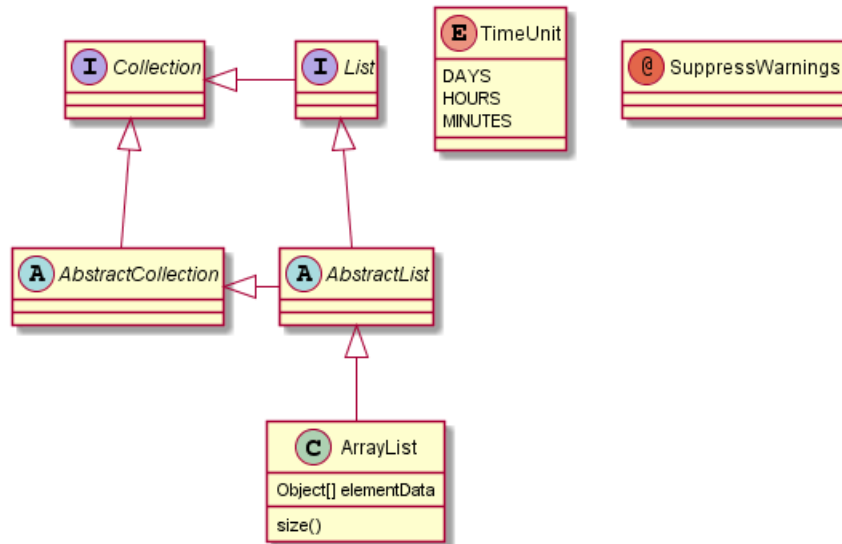
```

enum TimeUnit {
  DAYS
  HOURS
  MINUTES
}
  
```

```

annotation SuppressWarnings
  
```

@enduml



### 3.11 Using non-letters

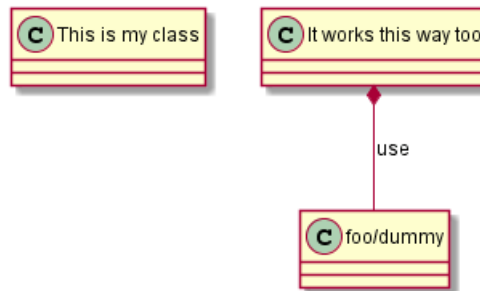
If you want to use non-letters in the class (or enum...) display, you can either :

- 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
  
```



### 3.12 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.

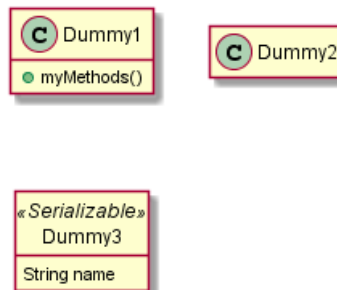
```
@startuml
class Dummy1 {
+myMethods()
}

class Dummy2 {
+hiddenMethod()
}

class Dummy3 <<Serializable>> {
String name
}

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

@enduml
```



### 3.13 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 come classes after file inclusion.

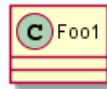
```
@startuml
class Foo1
class Foo2

Foo2 *-- Foo1

hide Foo2

@enduml
```

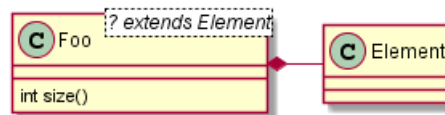




### 3.14 Use generics

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

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



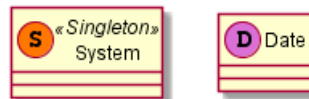
It is possible to disable this drawing using skinparam genericDisplay old command.

### 3.15 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
```



### 3.16 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
}
@enduml
```

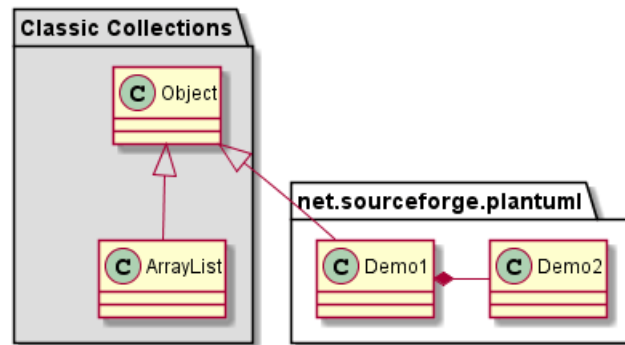


```

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

@enduml

```



### 3.17 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
}

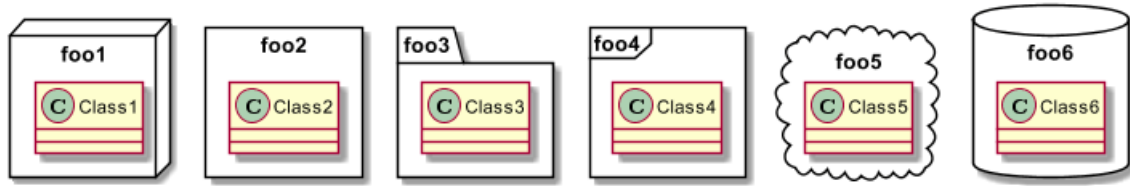
package foo4 <<Frame>> {
  class Class4
}

package foo5 <<Cloud>> {
  class Class5
}

package foo6 <<Database>> {
  class Class6
}

@enduml

```



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

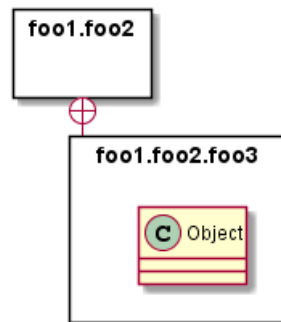
```
@startuml
skinparam packageStyle rectangle

package foo1.foo2 {
}

package foo1.foo2.foo3 {
    class Object
}

foo1.foo2 +-- foo1.foo2.foo3

@enduml
```



### 3.18 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 {
```



```

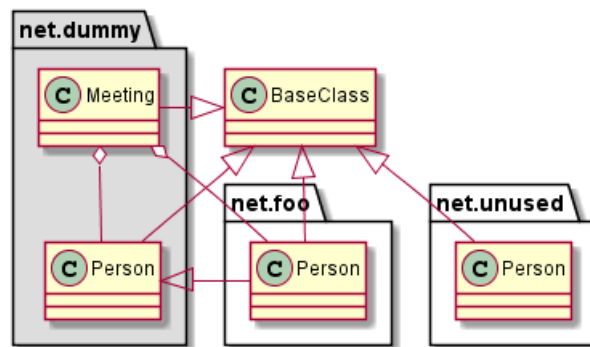
net.dummy.Person <|-- Person
.BaseClass <|-- Person

net.dummy.Meeting o-- Person
}

BaseClass <|-- net.unused.Person

@enduml

```



### 3.19 Automatic namespace creation

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

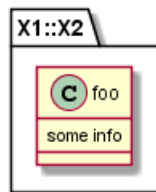
```

@startuml

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

@enduml

```



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

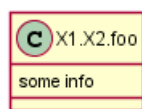
```

@startuml

set namespaceSeparator none
class X1.X2.foo {
    some info
}

@enduml

```

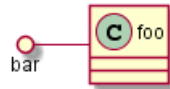


### 3.20 Lollipop interface

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

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

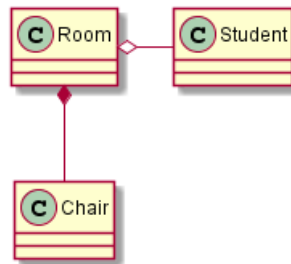
```
@startuml
class foo
bar ()- foo
@enduml
```



### 3.21 Changing arrows direction

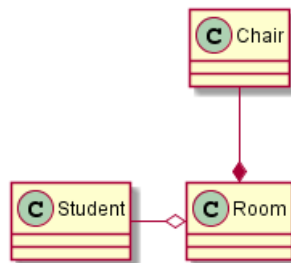
By default, links between classes have two dashes -- and are vertically oriented. It is possible to use horizontal link by putting a single dash (or dot) like this:

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



You can also change directions by reversing the link:

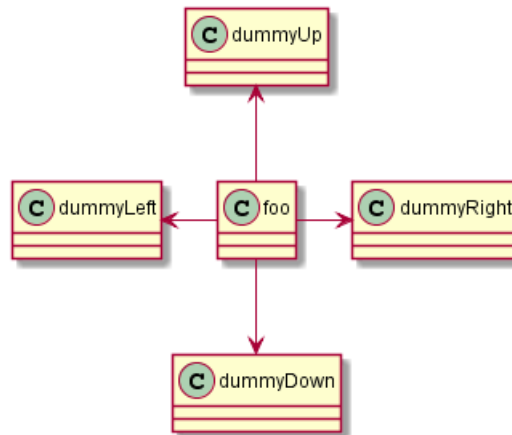
```
@startuml
Student -o Room
Chair ---* Room
@enduml
```



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.

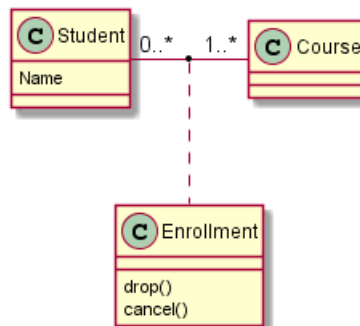
### 3.22 Association classes

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

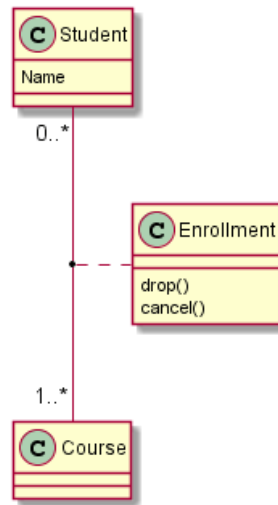
class Enrollment {
    drop()
    cancel()
}
@enduml
  
```



```

}
@enduml

```



### 3.23 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
```

```

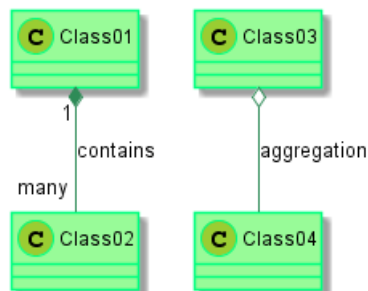
skinparam class {
  BackgroundColor PaleGreen
  ArrowColor SeaGreen
  BorderColor SpringGreen
}
skinparam stereotypeCBackgroundColor YellowGreen

```

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

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

```
@enduml
```



### 3.24 Skinned Stereotypes

You can define specific color and fonts for stereotyped classes.

```

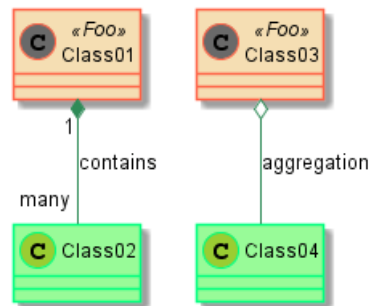
@startuml
skinparam class {
BackgroundColor PaleGreen
ArrowColor SeaGreen
BorderColor SpringGreen
BackgroundColor<<Foo>> Wheat
BorderColor<<Foo>> Tomato
}
skinparam stereotypeCBackgroundColor YellowGreen
skinparam stereotypeCBackgroundColor<< Foo >> DimGray

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

Class03 o-- Class04 : aggregation

@enduml

```



### 3.25 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 {

```

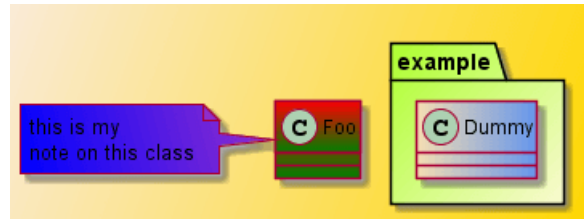


```

class Dummy
}

@enduml

```



### 3.26 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.

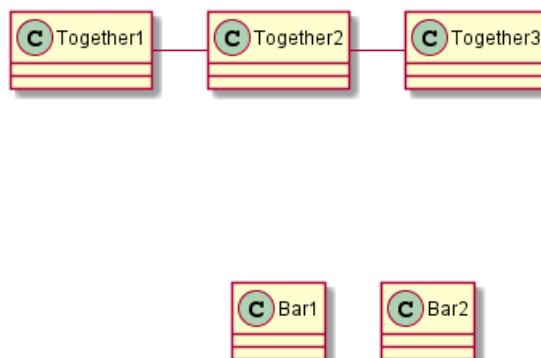
```

@startuml

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

@enduml

```



### 3.27 대용량파일분할하기

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

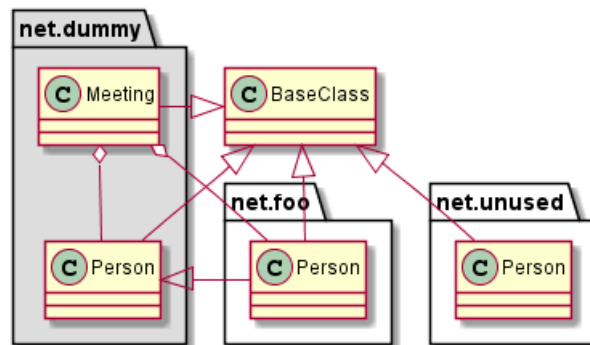
namespace net.dummy #DDDDDD {
.BaseClass <|-- Person
Meeting o-- Person

.BaseClass <|-- Meeting
}

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

net.dummy.Meeting o-- Person
}

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



## 4 Activity Diagram

### 4.1 Simple Activity

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 Activity"
"First Activity" --> (*)
@enduml
```

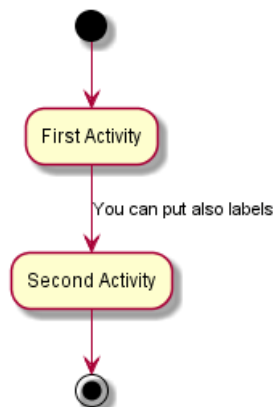


### 4.2 화살표라벨

기본적으로 화살표는 마지막으로 사용한 액티비티에서 시작한다.

화살표에 라벨을 붙이려면 화살표 정의 바로 다음에 대괄호를 사용한다.

```
@startuml
(*) --> "First Activity"
-->[You can put also labels] "Second Activity"
--> (*)
@enduml
```



### 4.3 Changing arrow direction

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

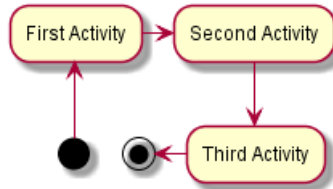
- -down-> (default arrow)
- -right-> or ->



```

• -left->
• -up->
@startuml
(*) -up-> "First Activity"
-right-> "Second Activity"
--> "Third Activity"
-left-> (*)
@enduml

```



#### 4.4 Branches

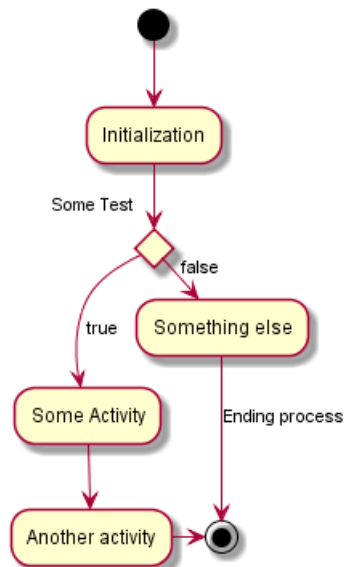
You can use if/then/else keywords to define branches.

```

@startuml
(*) --> "Initialization"

if "Some Test" then
  -->[true] "Some Activity"
  --> "Another activity"
-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"

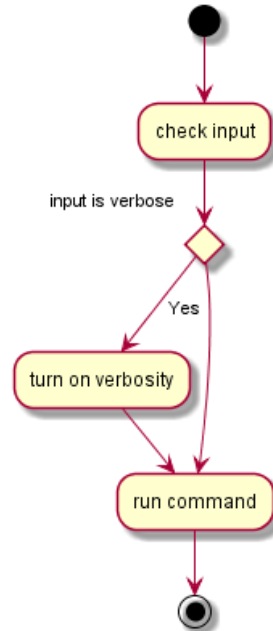
```



```

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

```



## 4.5 브랜치에뎀붙임

기본적으로 브랜치는 마지막에 정의한 액티비티와 연결된다. 하지만 이를 오버라이드하여 다른 연결 관계를 `if` 키워드로 정의할 수 있다.

브랜치내 브랜치 정도 가능하다.

```

@startuml
(*) --> if "Some Test" then
    -->[true] "activity 1"
    if "" then
-> "activity 3" as a3
    else
if "Other test" then
    -left-> "activity 5"
    else
    --> "activity 6"
    endif
    endif
else
    -->[false] "activity 2"
endif

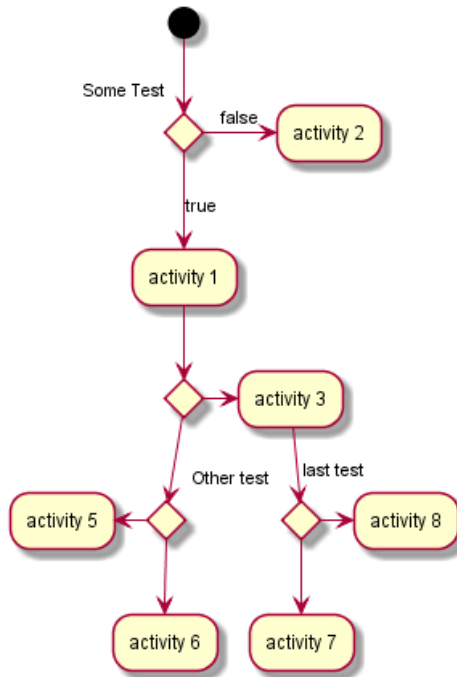
```



```

a3 --> if "last test" then
  --> "activity 7"
else
  -> "activity 8"
endif
@enduml

```



## 4.6 Synchronization

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

```

@startuml

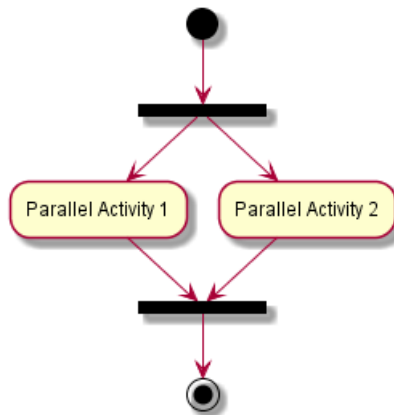
(*) --> ===B1===
--> "Parallel Activity 1"
--> ===B2===

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

--> (*)

@enduml

```



## 4.7 Long activity description

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

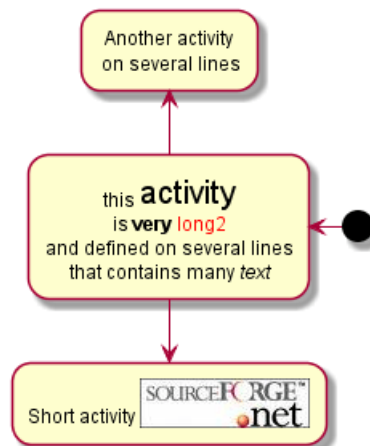
You can also give a short code to the activity with the `as` keyword. This code can be used latter in the diagram description.

```

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

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

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



## 4.8 Notes

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

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

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

```

@startuml
  
```

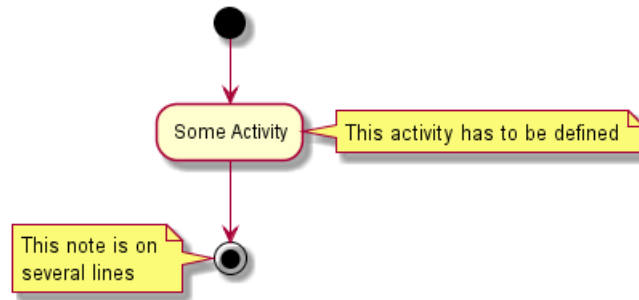


```

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

@enduml

```



## 4.9 Partition

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

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

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

```

@startuml

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

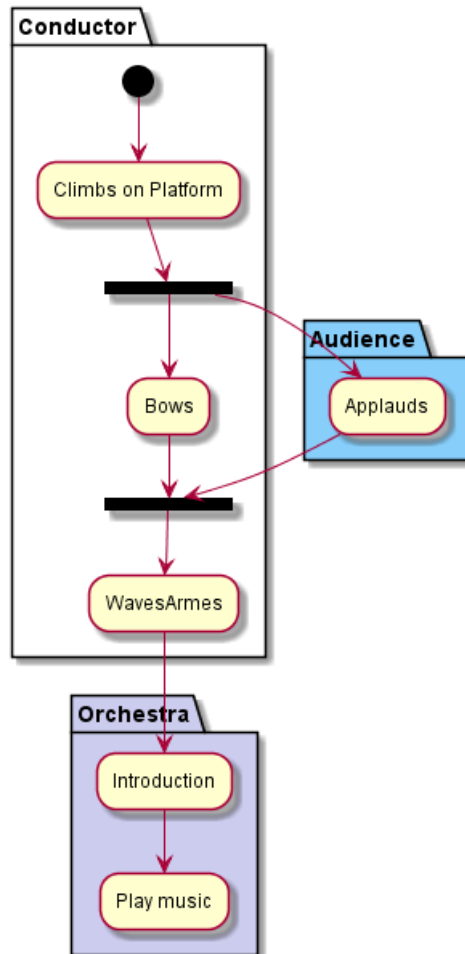
partition Audience #LightSkyBlue {
  === S1 === --> Applauds
}

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

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

@enduml

```



## 4.10 Skinparam

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

You can use this command :

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

You can define specific color and fonts for stereotyped activities.

```
@startuml
```

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

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

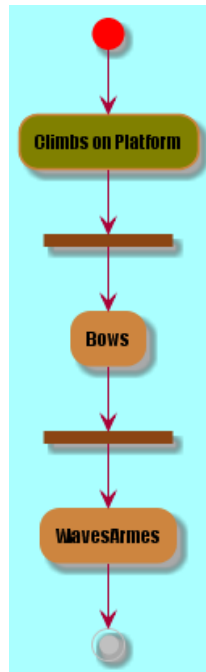


```

--> === S1 ===
--> Bows
--> === S2 ===
--> WavesArmes
--> (*)

@enduml

```



## 4.11 Octagon

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

```

@startuml
'Default is skinparam activityShape roundBox
skinparam activityShape octagon

(*) --> "First Activity"
"First Activity" --> (*)

@enduml

```



## 4.12 Complete example

```

@startuml
title Servlet Container

(*) --> "ClickServlet.handleRequest()"

```



```
--> "new Page"

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

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

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

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

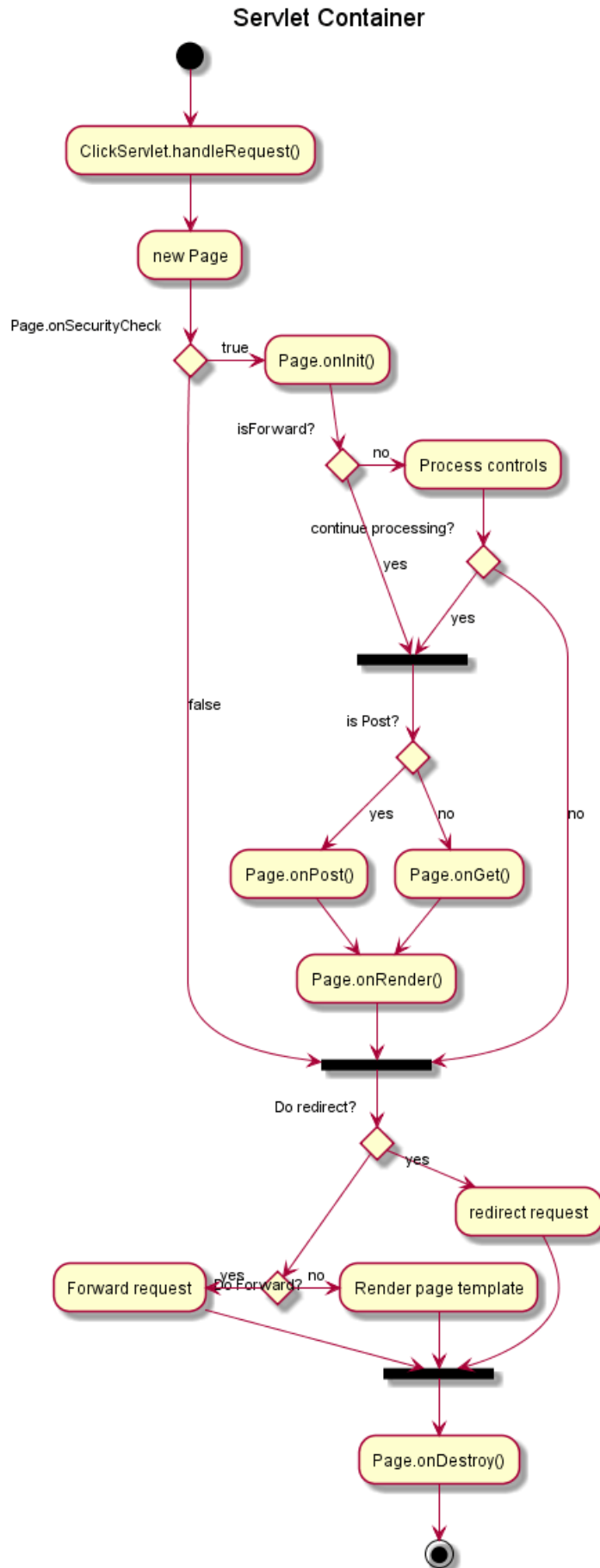
    if "is Post?" then
      -->[yes] "Page.onPost()"
      --> "Page.onRender()" as render
      --> ===REDIRECT_CHECK===
      else
      -->[no] "Page.onGet()"
      --> render
      endif

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

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

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

@enduml
```



## 5 Activity Diagram (beta)

기존의 activity diagram 문법은 몇몇 제약과 단점이 있다. (예를 들면, 유지보수가 어렵다.)

그래서 완전 새로운 문법과 구현이 베타 버전으로 고안되었고, 우리는 더 나은 포맷과 문법으로 정의할 수 있었다.

이 새로운 구현의 또 다른 장점은 (시퀀스 다이어그램과 같이) Graphviz 를 설치할 필요 없이 수행된다는 것이다. 새로운 구문이 이전 구문을 대체할 것이다. 그러나 호환성을 보장하기 위해 이전 구문이 여전히 인식될 것이다. 새로운 구문으로 이전을 권장한다.

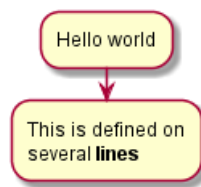
### 5.1 Simple Activity

Activities label starts with : and ends with ;.

Text formatting can be done using creole wiki syntax.

They are implicitly linked in their definition order.

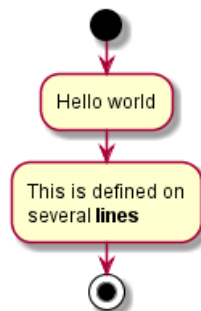
```
@startuml
:Hello world;
:This is defined on
several lines;
@enduml
```



### 5.2 Start/Stop

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
```

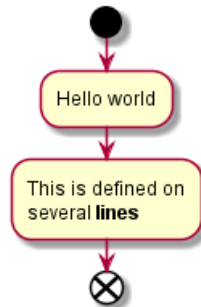




```

several **lines**;
end
@enduml

```



### 5.3 Conditional

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

```

@startuml

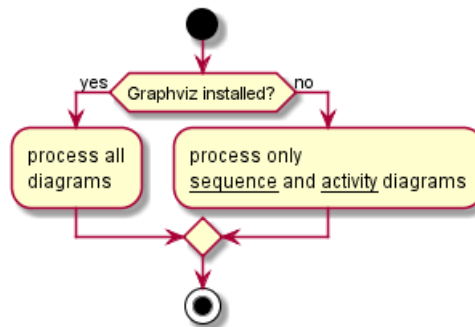
start

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

stop

@enduml

```



You can use the `elseif` keyword to have several tests :

```

@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)

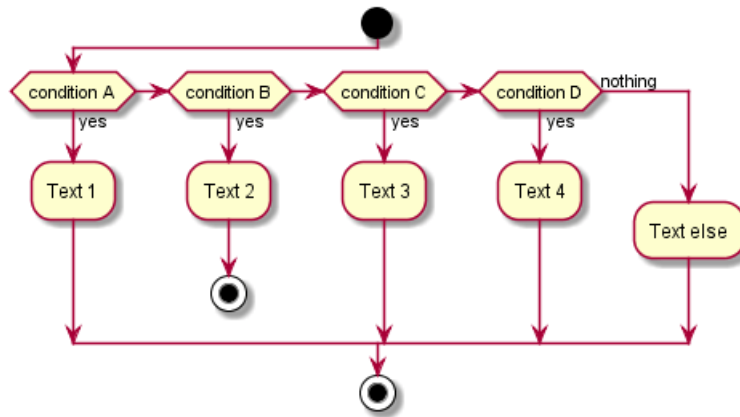
```



```

:Text else;
endif
stop
@enduml

```



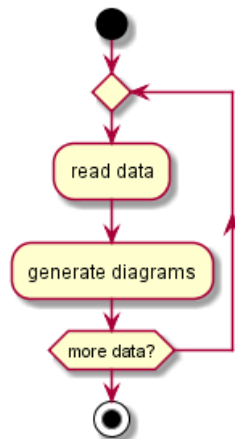
## 5.4 Repeat loop

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

```

@startuml
start
repeat
:read data;
:generate diagrams;
repeat while (more data?)
stop
@enduml

```



## 5.5 While loop

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

```

@startuml
start

```



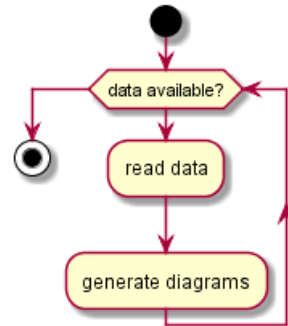
```

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

stop

@enduml

```

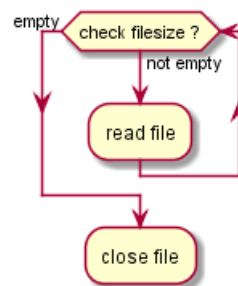


It is possible to provide a label after the endwhile keyword, or using the is keyword.

```

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

```



## 5.6 Parallel processing

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

```

@startuml

start

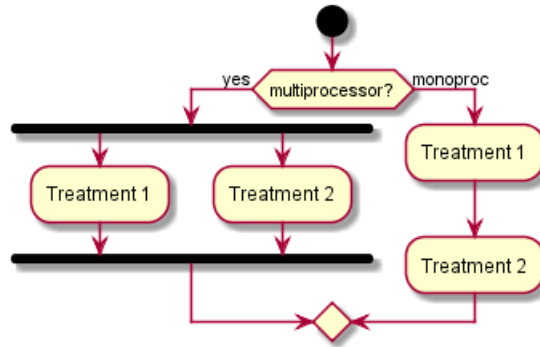
if (multiprocessor?) then (yes)
  fork
  :Treatment 1;
  fork again
  :Treatment 2;
  end fork
else (monoproc)
  :Treatment 1;
  :Treatment 2;
endif

end

```



@enduml



### 5.7 Notes

Text formatting can be done using creole wiki syntax.

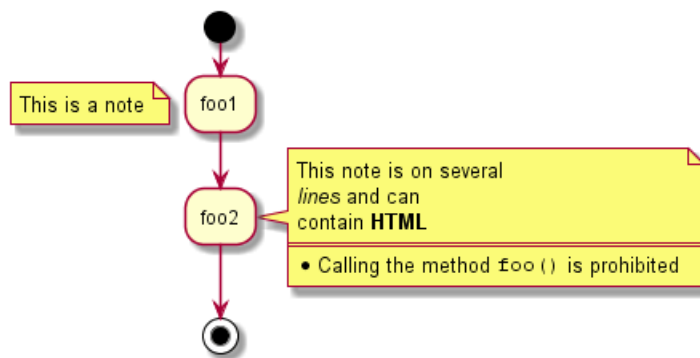
A note can be floating, using floating keyword.

@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



### 5.8 Colors

You can specify a color for some activities.

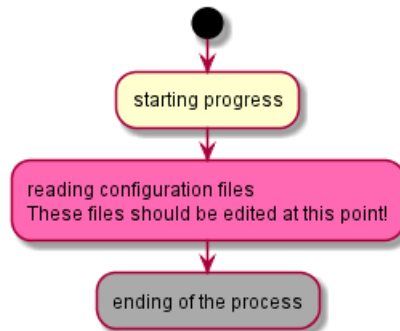
@startuml

```

start
:starting progress;
    
```

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

@enduml
```

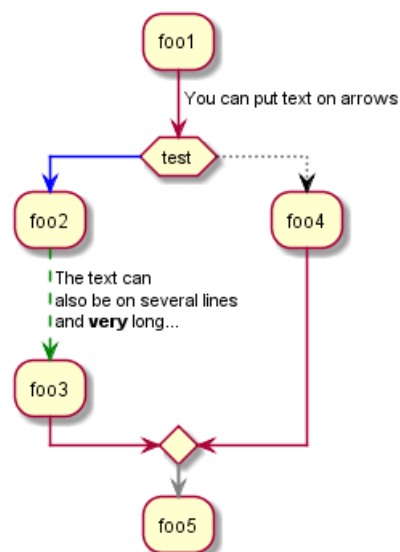


## 5.9 Arrows

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.

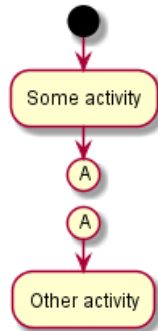
```
@startuml
:foo1;
-> You can put text on arrows;
if (test) then
-[#blue]->
:foo2;
-[#green,dashed]-> The text can
also be on several lines
and very long...;
:foo3;
else
-[#black,dotted]->
:foo4;
endif
-[#gray,bold]->
:foo5;
@enduml
```



## 5.10 Connector

You can use parentheses to denote connector.

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

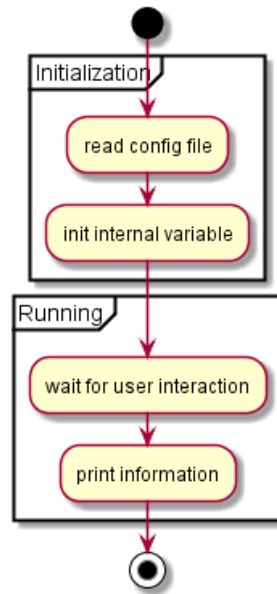


## 5.11 Grouping

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
```



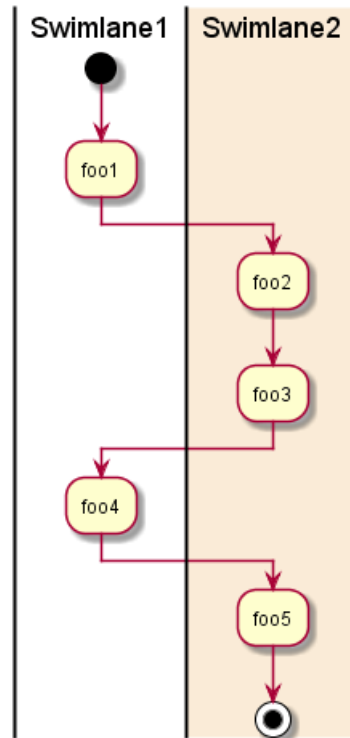
## 5.12 Swimlanes

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
  
```



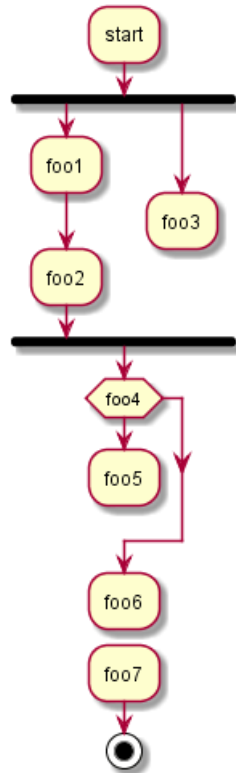
### 5.13 Detach

It's possible to remove an arrow using the detach keyword.

```

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





## 5.14 SDL

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

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

```

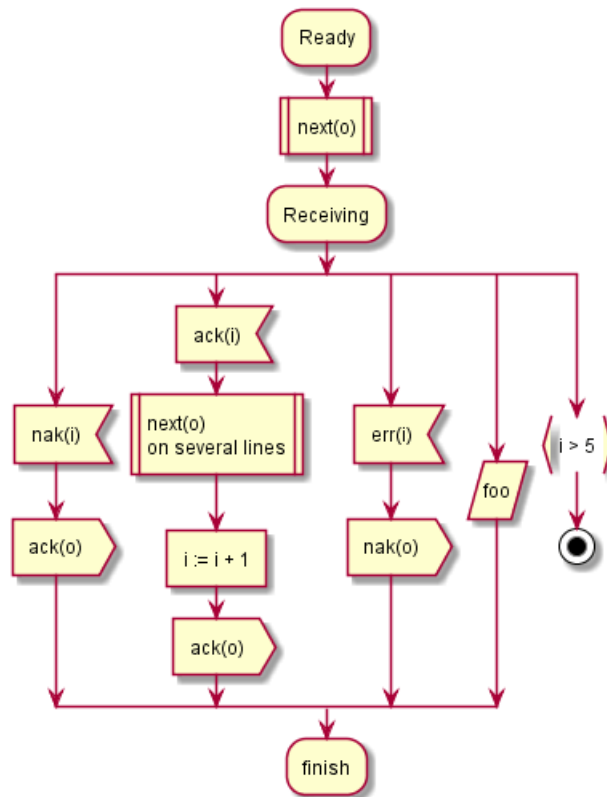
@startuml
:Ready;
:next(o)|
:Receiving;
split
:nak(i)<
:ack(o)>
split again
:ack(i)<
:next(o)
on several lines|
:i := i + 1]
:ack(o)>
split again
:err(i)<
:nak(o)>
split again
:foo/
split again
:i > 5}
  
```



```

stop
end split
:finish;
@enduml

```



## 5.15 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
endif
:Page.onRender();
endif
else (false)
endif

if (do redirect?) then (yes)

```

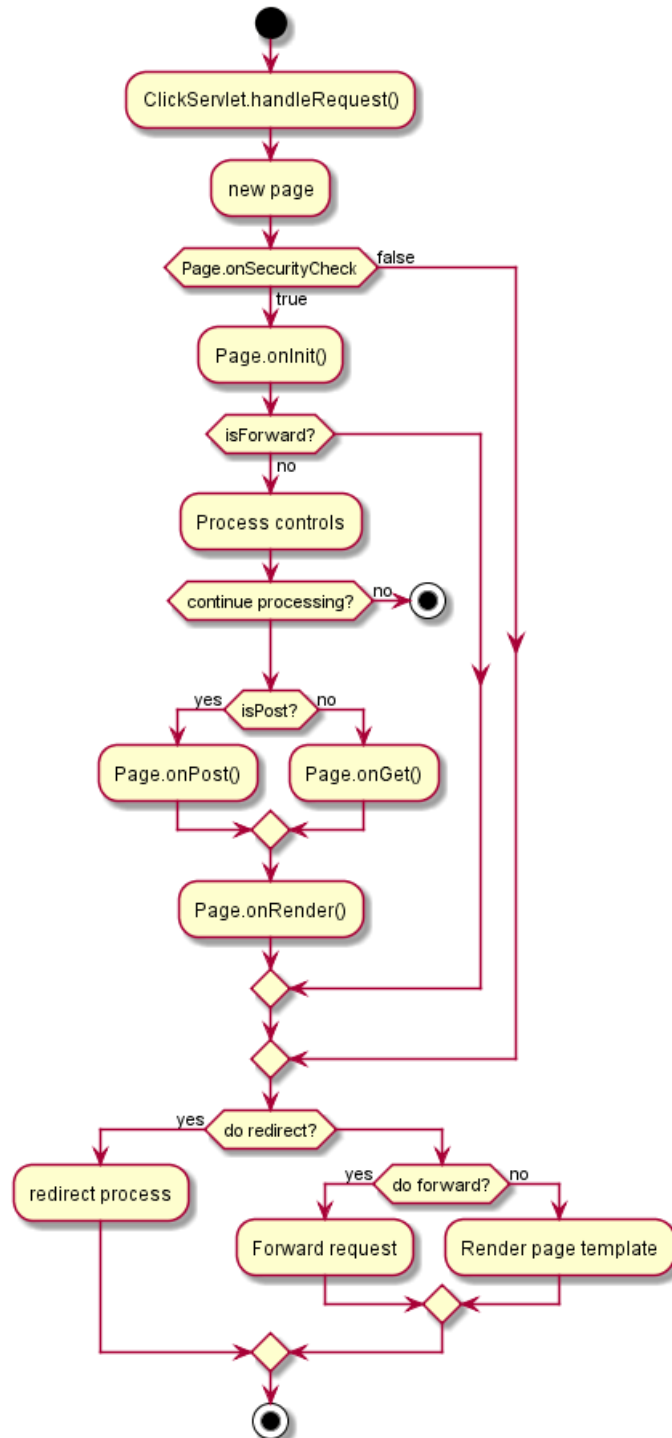
```

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

stop

@enduml

```



## 6 컴포넌트 다이어그램

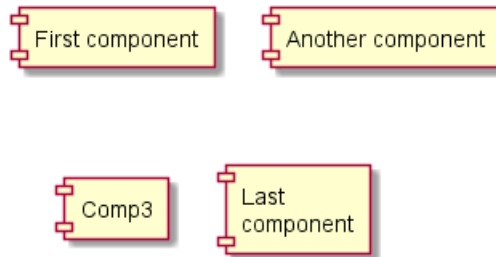
몇가지예제를보자. :

### 6.1 컴포넌트

컴포넌트는 반드시 대괄호 [] 로 둘러싸여야 한다.

컴포넌트를 정의할 때 `component` 키워드도 사용할 수 있다. `as` 키워드를 이용해서 별명을 정의할 수도 있다. 이 별명은 뒤에서 관계를 정의할 때 사용된다.

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



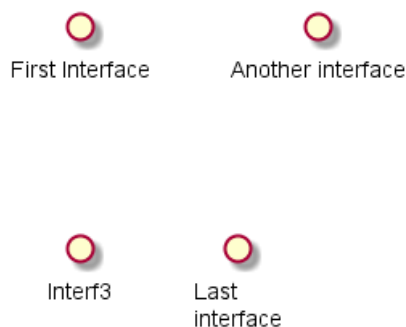
### 6.2 인터페이스

인터페이스는 () 기호로 정의될 수 있다 (이 기호가 원처럼 보이기 때문이다).

`interface` 키워드도 인터페이스를 정의하는데 사용할 수 있다. `as` 키워드를 이용해서 별명을 정의할 수도 있다. 이 별명은 뒤에서 관계를 정의할 때 사용된다.

인터페이스를 정의하는 일은 선택 (`optional`) 이라는 것을 뒤에서 확인할 것이다.

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



### 6.3 기본예제

요소들간의연결은점선 (..), 실선 (--), 그리고화살표 (-->) 기호들의조합으로생성된다.

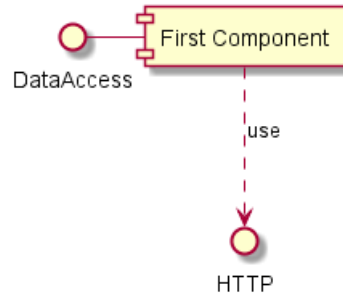
```
@startuml
```

```

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

```

```
@enduml
```



### 6.4 메모사용하기

각오브젝트에관련된메모를정의하기위해 note left of, note right of, note top of, note bottom of 키워드들을사용할수있다.

메모는또한 note 키워드를통해단독으로정의될수도있고, 다른오브젝트들에 .. 기호로연결된다.

```
@startuml
```

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

```

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

```

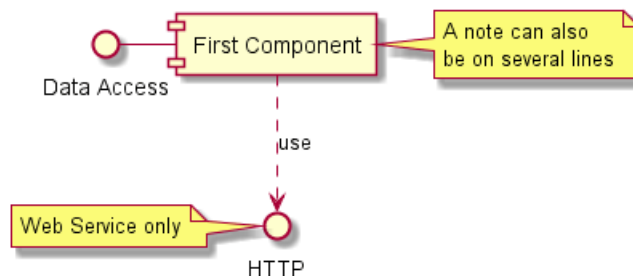
```
note left of HTTP : Web Service only
```

```

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

```

```
@enduml
```



### 6.5 컴포넌트를그룹으로나누기

컴포넌트들과인터페이스들을그룹으로나누기위해여러가지키워드를사용할수있다:

- package



```

    • node
    • folder
    • frame
    • cloud
    • database

@startuml

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

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

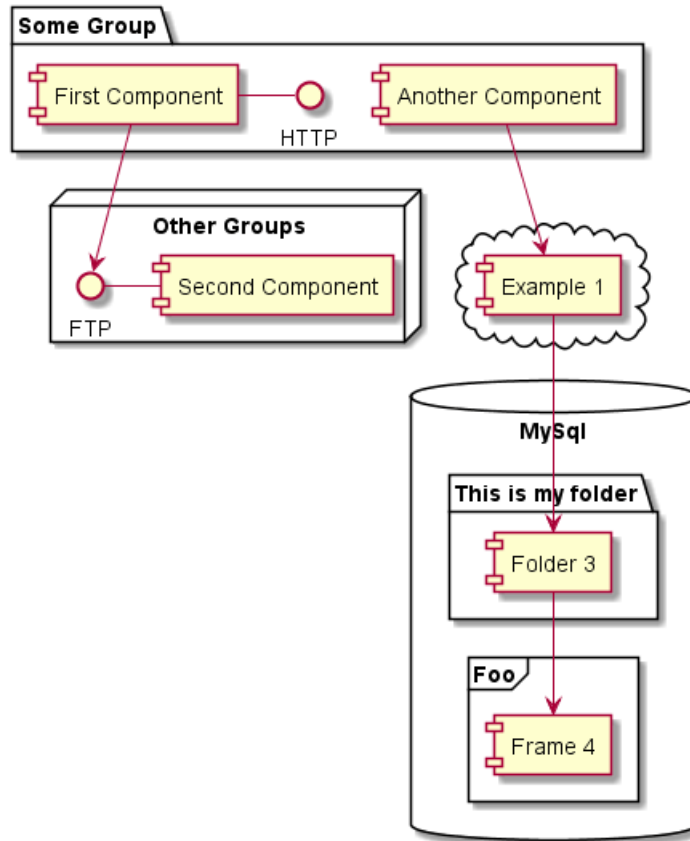
cloud {
    [Example 1]
}

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

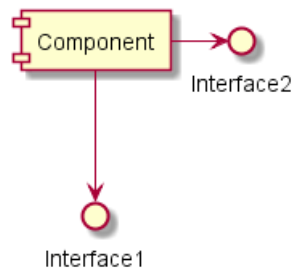
```



### 6.6 화살표방향바꾸기

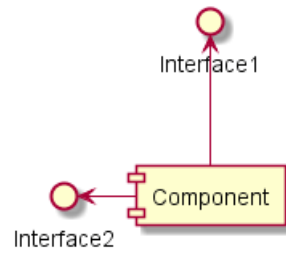
기본적으로 클래스들간의 연결은 두개의 대시를 -- 갖고 방향은 수직방향이다. 다음처럼 한개의 대시 (혹은 점) 를 넣어 수평방향 연결을 사용할 수 있다:

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



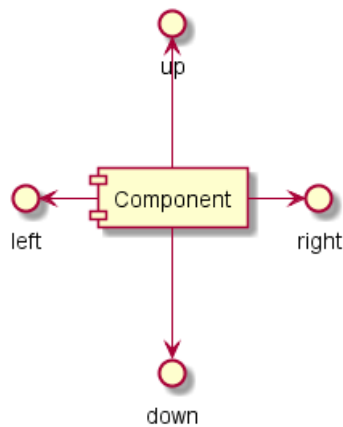
연결을 반전시켜 방향을 바꿀 수도 있다:

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



화살표안에 left, right, up, down 키워드를 추가하여 방향을 바꾸는 것도 가능하다:

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



방향을 의미하는 단어의 첫 번째 글자만 사용해서 화살표를 짧게 할 수 있다. (예를 들면, -down- 대신 -d-) 또는 두 글자를 사용해도 된다. (-do-).

이 기능을 남용하지 말아야 한다는 것을 명심하자: 그래야 별다른 수정 없이도 *GraphViz* 가 좋은 결과를 보여준다.

## 6.7 Use UML2 notation

The skinparam componentStyle uml2 command is used to switch to UML2 notation.

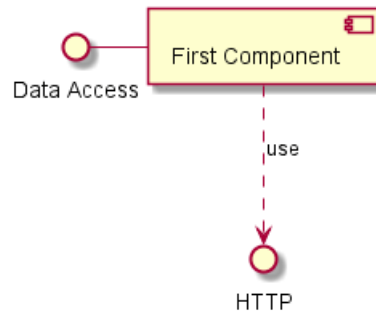
```
@startuml
skinparam componentStyle uml2

interface "Data Access" as DA

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

@enduml
```

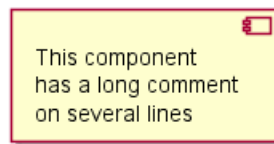




## 6.8 Long description

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
```



## 6.9 Individual colors

You can specify a color after component definition.

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



## 6.10 Using Sprite in Stereotype

You can use sprites within stereotype components.

```
@startuml
sprite $businessProcess [16x16/16] {
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFF0FFFFF
FFFFFFFFF0FFFFF
FF0000000000FFF
FF0000000000FFF
FF0000000000FFF
FFFFFFFFF0FFFFF
FFFFFFFFF0FFFFF
}
```



```

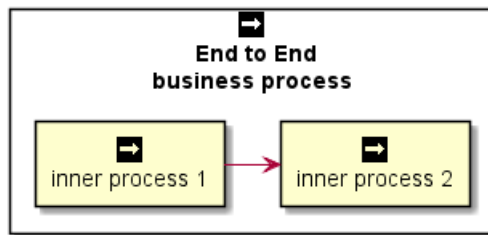
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
}

```

```

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

```



## 6.11 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
```

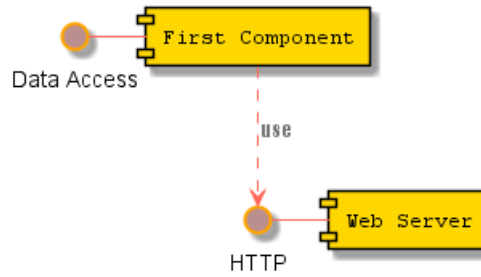
```
DA - [First Component]
```

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



```
HTTP - [Web Server] << Apache >>
```

```
@enduml
```



```
@startuml
```

```
[AA] <<static lib>>
```

```
[BB] <<shared lib>>
```

```
[CC] <<static lib>>
```

```
node node1
```

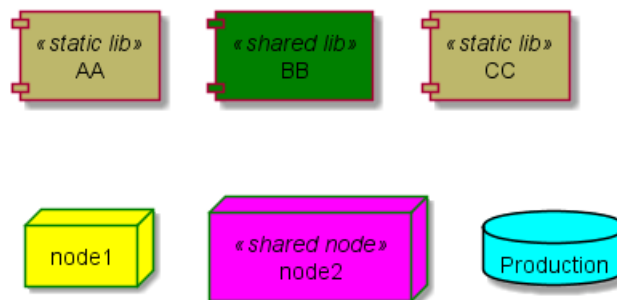
```
node node2 <<shared node>>
```

```
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 상태 다이어그램

### 7.1 간단한 상태

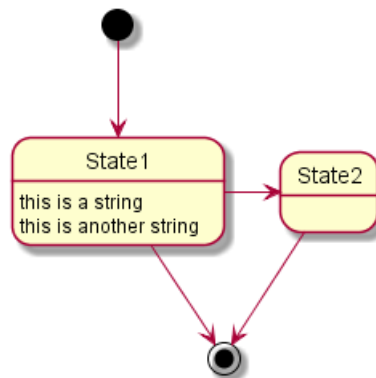
[\*] 을사용해서시작점과종료점을그린다.

--> 를사용해서화살표를그린다.

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

State1 -> State2
State2 --> [*]

@enduml
```

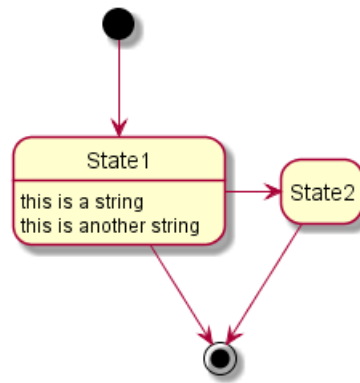


### 7.2 Change state rendering

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
```



### 7.3 상태수정

물론 상태는 수정될 수 있다. state 키워드와 브라켓을 정의해야 한다.

```

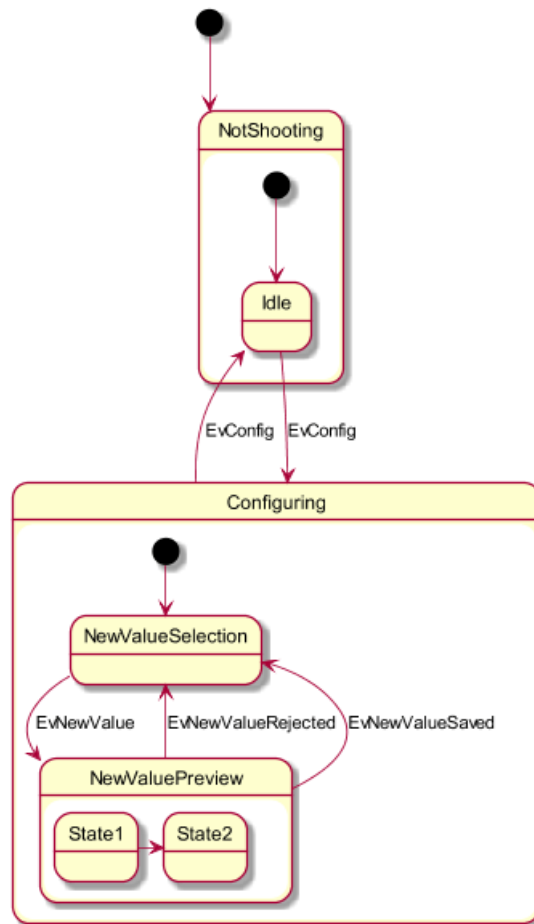
@startuml
scale 350 width
[*] --> NotShooting

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
  
```



## 7.4 긴이름

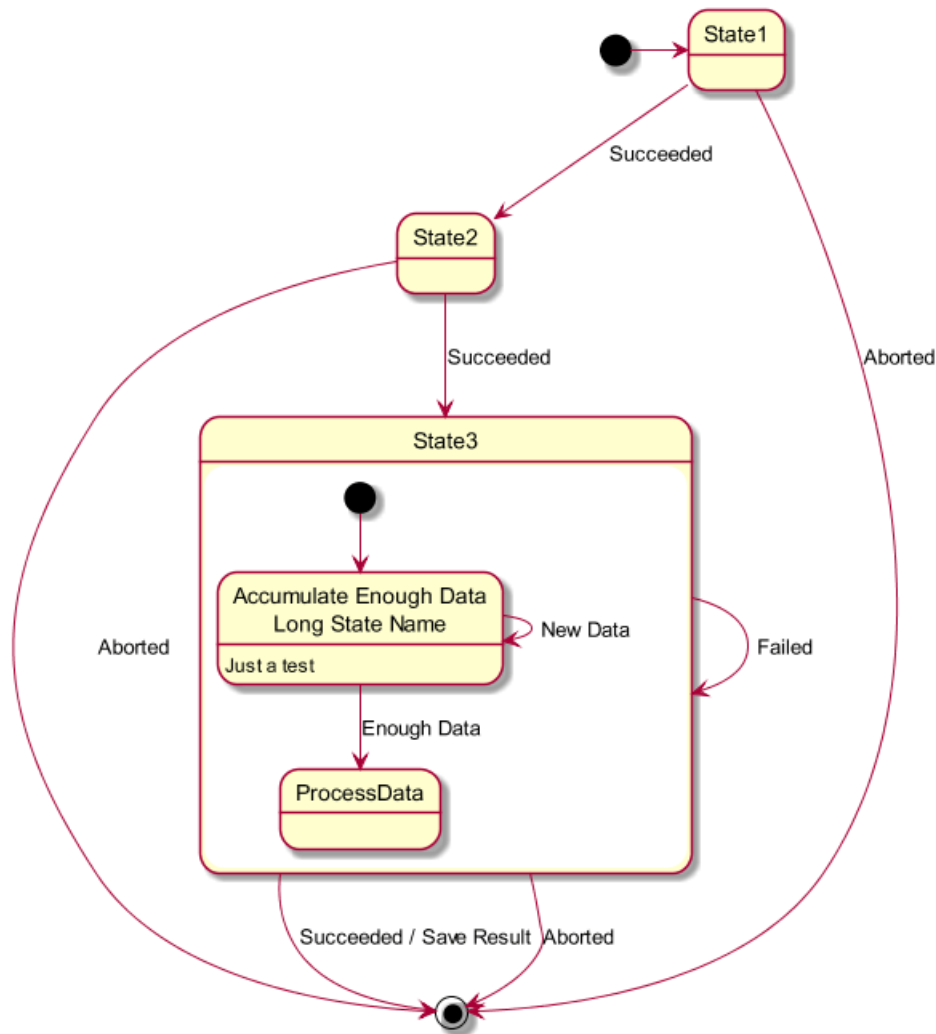
state 키워드를 사용하면 상태들을 길게 기술할 수 있다.

```

@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
  
```



## 7.5 Concurrent state

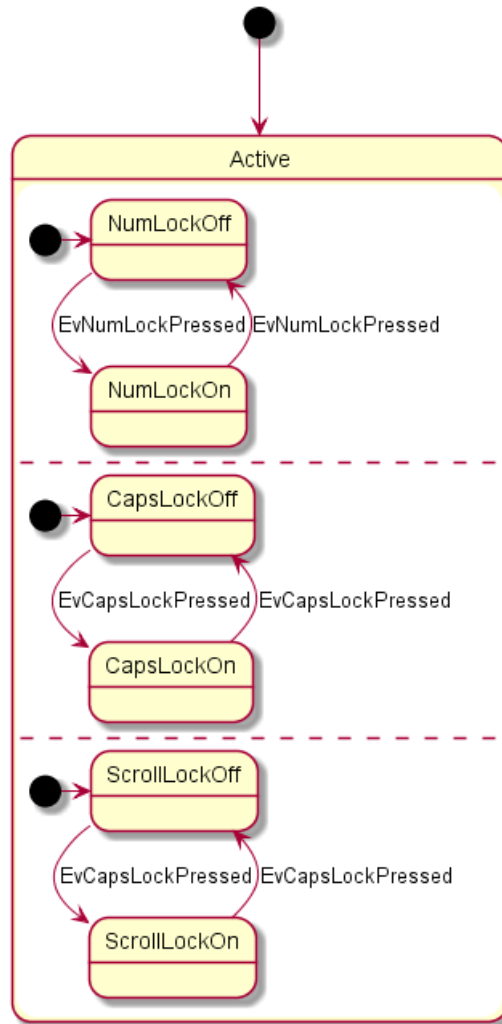
You can define concurrent state into a composite state using either -- or || symbol as separator.

```
@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
```





## 7.6 Arrow direction

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

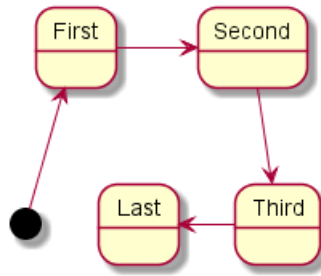
- `-down->` (default arrow)
- `-right->` or `->`
- `-left->`
- `-up->`

```
@startuml
```

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

```
@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.

## 7.7 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
```

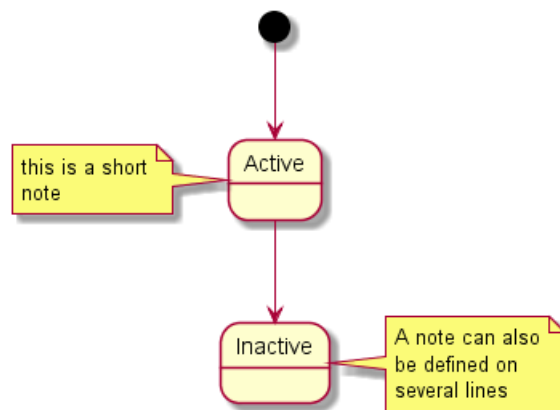
```
[*] --> Active
Active --> Inactive
```

```
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
```



## 7.8 More in notes

You can put notes on composite states.

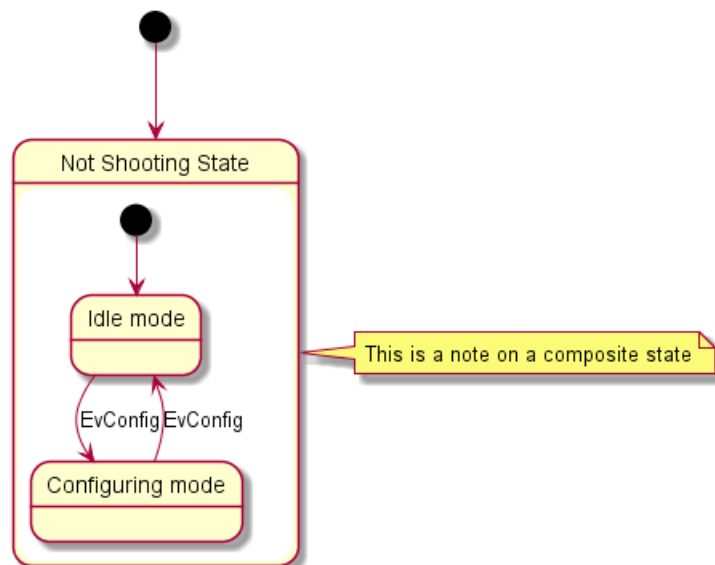
```
@startuml
```

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



## 7.9 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 states.

```
@startuml
skinparam backgroundColor LightYellow
skinparam state {
  StartColor MediumBlue
```

```

EndColor Red
BackgroundColor Peru
BackgroundColor<<Warning>> Olive
BorderColor Gray
FontName Impact
}

```

```

[*] --> NotShooting

```

```

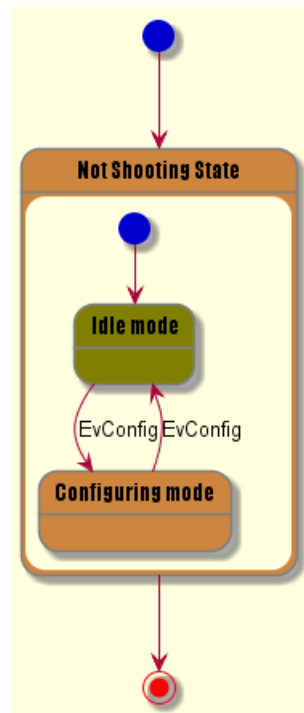
state "Not Shooting State" as NotShooting {
  state "Idle mode" as Idle <<Warning>>
  state "Configuring mode" as Configuring
  [*] --> Idle
  Idle --> Configuring : EvConfig
  Configuring --> Idle : EvConfig
}

```

```

NotShooting --> [*]
@enduml

```

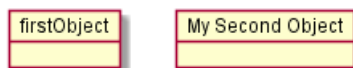


## 8 Object Diagram

### 8.1 Definition of objects

You define instance of objects using the object keywords.

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



### 8.2 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.

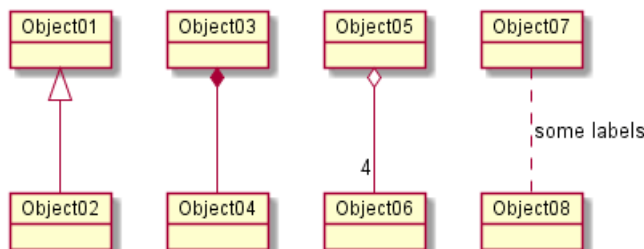
Knowing those rules, it is possible to draw the following drawings.

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

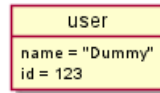
Object01 <|-- Object02
Object03 *-- Object04
Object05 o-- "4" Object06
Object07 .. Object08 : some labels
@enduml
```



### 8.3 Adding fields

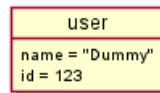
To declare fields, you can use the symbol : followed by the field's name.

```
@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
```



## 8.4 Common features with class diagrams

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

## 9 Timing 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.

### 9.1 Declaring participant

You declare participant using concise or robust keyword, depending on how you want them to be drawn.

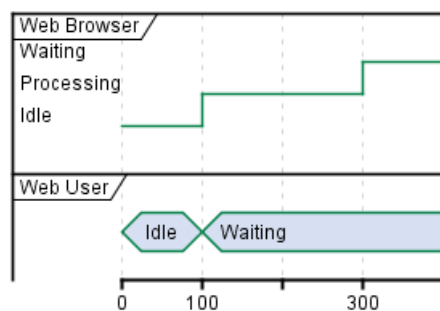
You define state change using the @ notation, and the is verb.

```
@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
```



### 9.2 Adding message

You can add message using the following syntax.

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

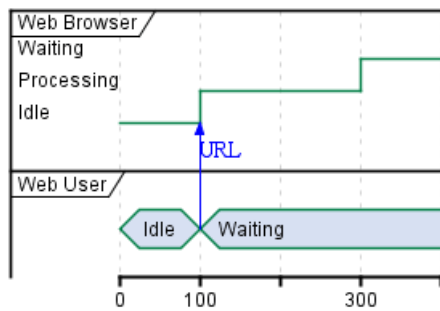
@0
WU is Idle
WB is Idle

@100
WU -> WB : URL
WU is Waiting
WB is Processing

@300
WB is Waiting
```



@enduml



### 9.3 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
```

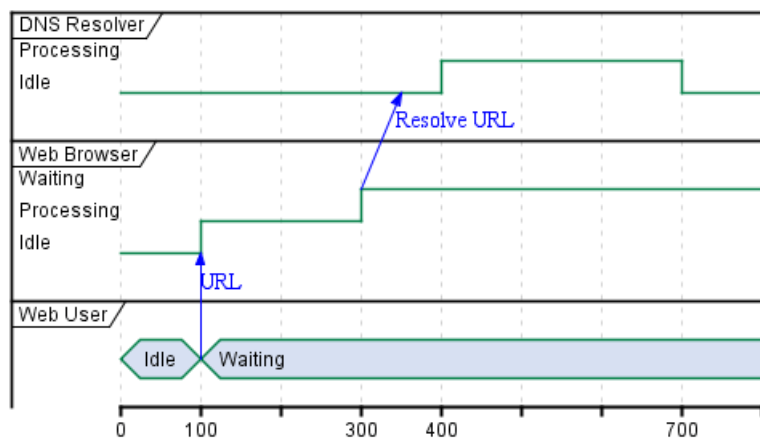
```
@0
WU is Idle
WB is Idle
DNS is Idle
```

```
@+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
```



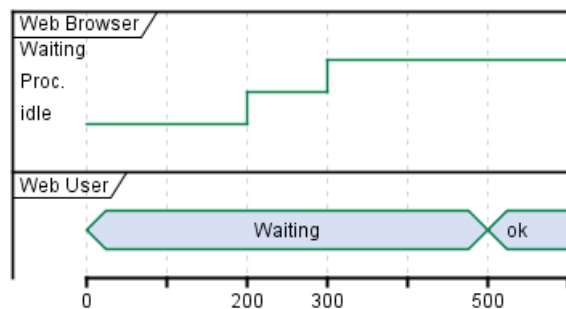
## 9.4 Participant oriented

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
```

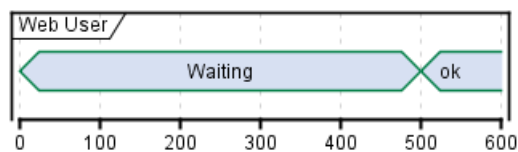


## 9.5 Setting scale

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
```



## 9.6 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
```

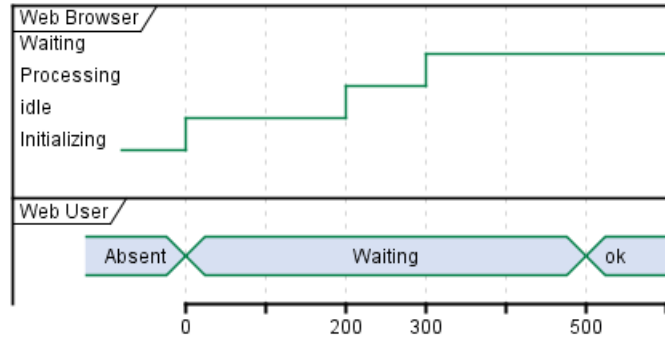




```

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

@WU
0 is Waiting
+500 is ok
@enduml
    
```

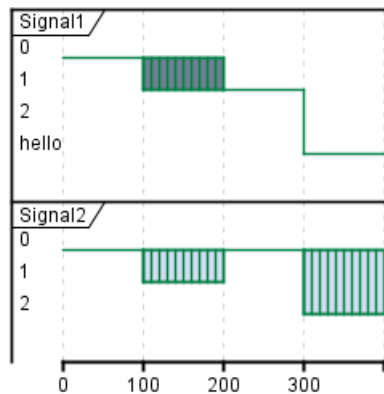


### 9.7 Intricated state

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
    
```



## 9.8 Hidden state

It is also possible to hide some state.

```
@startuml
concise "Web User" as WU

@0
WU is {-}

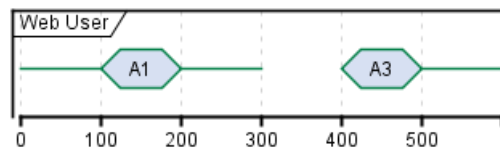
@100
WU is A1

@200
WU is {-}

@300
WU is {hidden}

@400
WU is A3

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



## 9.9 Adding constraint

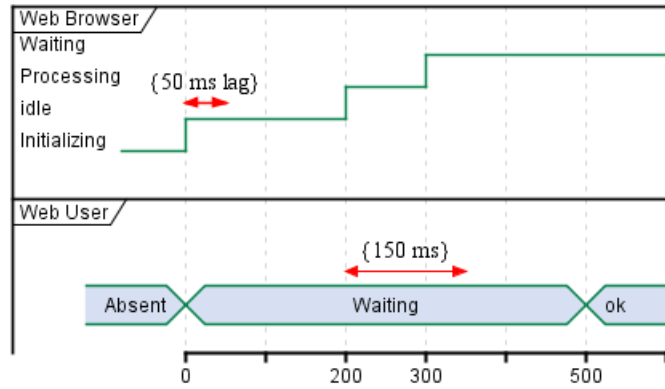
It is possible to display time constraints on the diagrams.

```
@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
```



### 9.10 Adding texts

You can optionally add a title, a header, a footer, a legend and a caption:

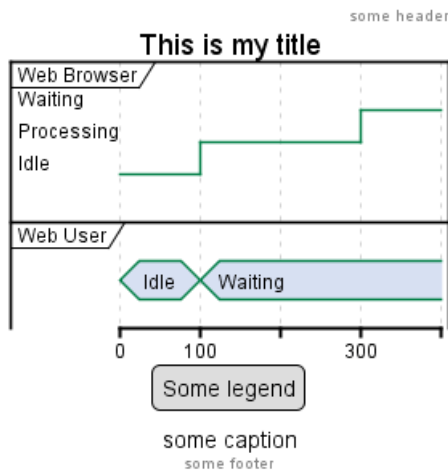
```
@startuml
Title This is my title
header: some header
footer: some footer
legend
Some legend
end legend
caption some caption
```

```
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 Gantt Diagram

This is only a proposal and subject to change.

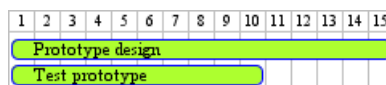
You are very welcome to create a new discussion on this future syntax. Your feedbacks, ideas and suggestions help us to find the right solution.

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

### 10.1 Declaring tasks

Tasks defined using square bracket. Their durations are defined using the last verb:

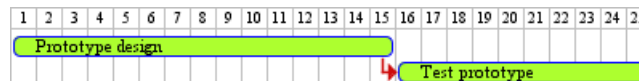
```
@startgantt
[Prototype design] lasts 15 days
[Test prototype] lasts 10 days
@endgantt
```



### 10.2 Adding constraints

It is possible to add constraints between task.

```
@startgantt
[Prototype design] lasts 15 days
[Test prototype] lasts 10 days
[Test prototype] starts at [Prototype design]'s end
@endgantt
```



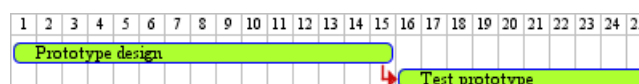
```
@startgantt
[Prototype design] lasts 10 days
[Code prototype] lasts 10 days
[Write tests] lasts 5 days
[Code prototype] starts at [Prototype design]'s end
[Write tests] starts at [Code prototype]'s start
@endgantt
```



### 10.3 Short names

It is possible to define short name for tasks with the as keyword.

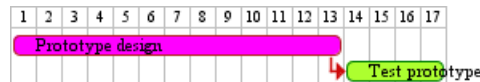
```
@startgantt
[Prototype design] as [D] lasts 15 days
[Test prototype] as [T] lasts 10 days
[T] starts at [D]'s end
@endgantt
```



## 10.4 Customize colors

It also possible to customize colors.

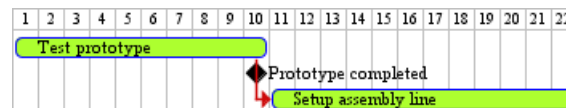
```
@startgantt
[Prototype design] lasts 13 days
[Test prototype] lasts 4 days
[Test prototype] starts at [Prototype design]'s end
[Prototype design] is colored in Fuchsia/FireBrick
[Test prototype] is colored in GreenYellow/Green
@endgantt
```



## 10.5 Milestone

You can define Milestones using the happens verb.

```
@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
```



## 10.6 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
```



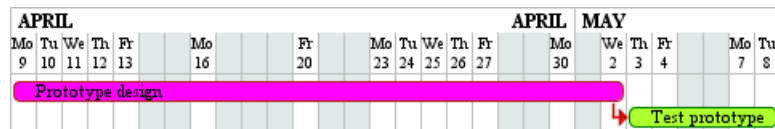
## 10.7 Close day

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
```



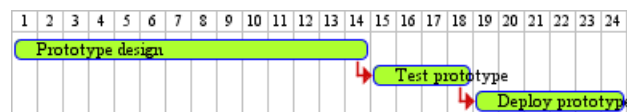
```
@endgantt
```



## 10.8 Simplified task succession

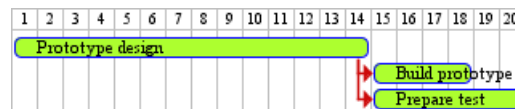
It's possible to use the then keyword to denote consecutive tasks.

```
@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
```



## 10.9 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
```



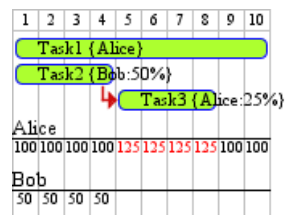
## 10.10 Working with resources

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
@endganttt
```



## 10.11 Complex example

It also possible to use the and conjunction.

You can also add delays in constraints.

```
@startganttt
```

```
[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 e
```

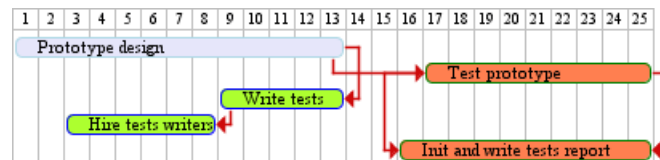
```
[Write tests] lasts 5 days and ends at [Prototype design]'s end
```

```
[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
```

```
@endganttt
```



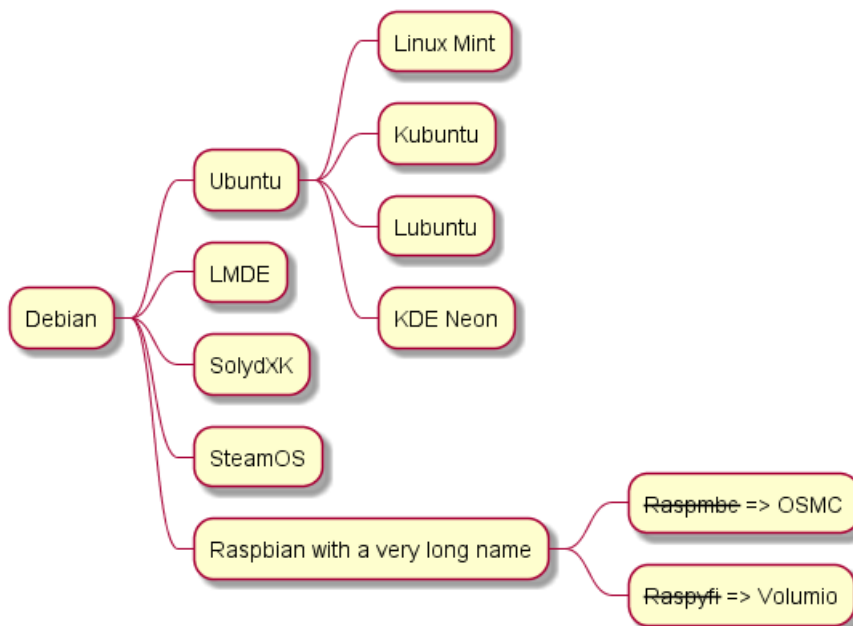
## 11 MindMap

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

### 11.1 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
```



### 11.2 Removing box

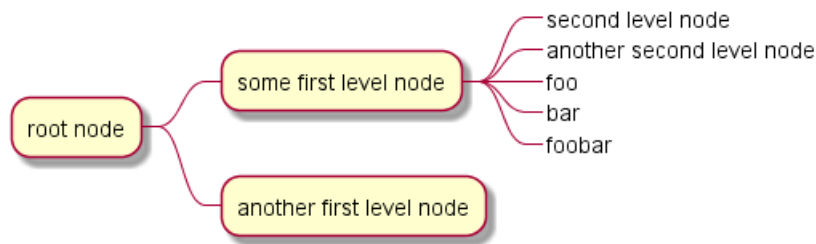
You can remove the box drawing using an underscore.

```
@startmindmap
* root node
** some first level node
***_ second level node
***_ another second level node
***_ foo
***_ bar
***_ foobar
** another first level node
```





```
@endmindmap
```

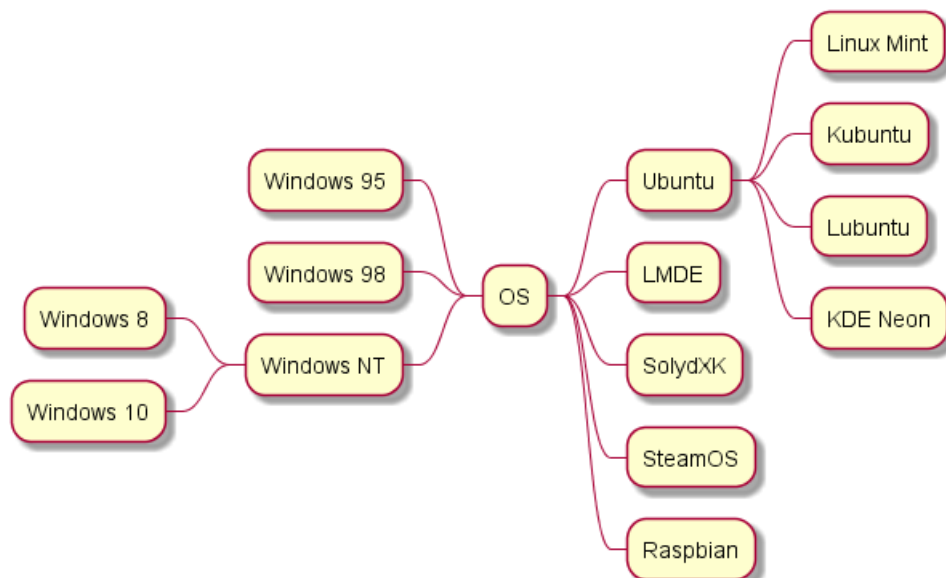


### 11.3 Arithmetic notation

You can use the following notation to choose diagram side.

```

@startmindmap
+ OS
++ Ubuntu
+++ Linux Mint
+++ Kubuntu
+++ Lubuntu
+++ KDE Neon
++ LMDE
++ SolydXK
++ SteamOS
++ Raspbian
-- Windows 95
-- Windows 98
-- Windows NT
--- Windows 8
--- Windows 10
@endmindmap
  
```



### 11.4 Markdown syntax

This syntax is compatible with Markdown

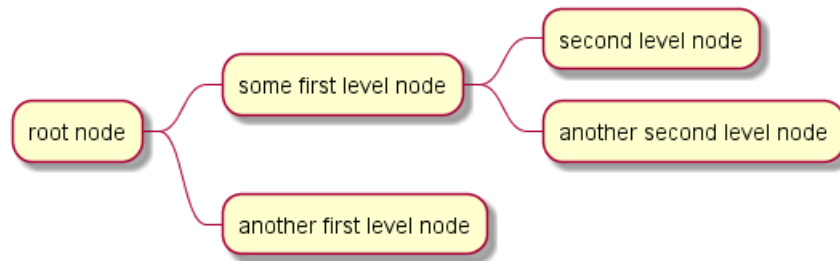
```
@startmindmap
```



```

* root node
* some first level node
* second level node
* another second level node
* another first level node
@endmindmap

```



## 11.5 Changing diagram direction

It is possible to use both sides of the diagram.

```

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

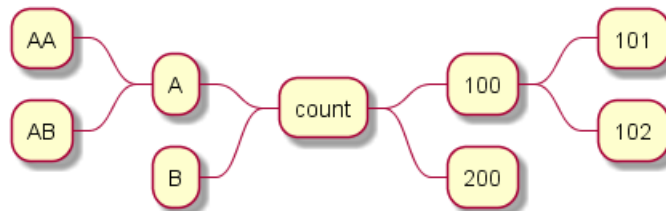
```

left side

```

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

```



## 11.6 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
    
```

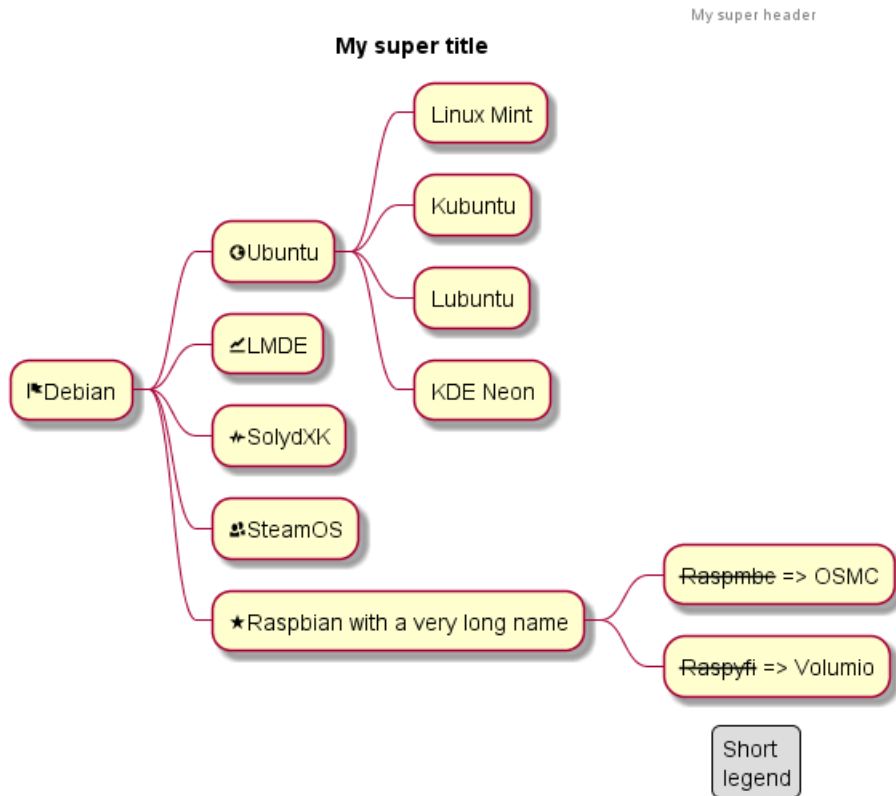


figure 1  
My super footer

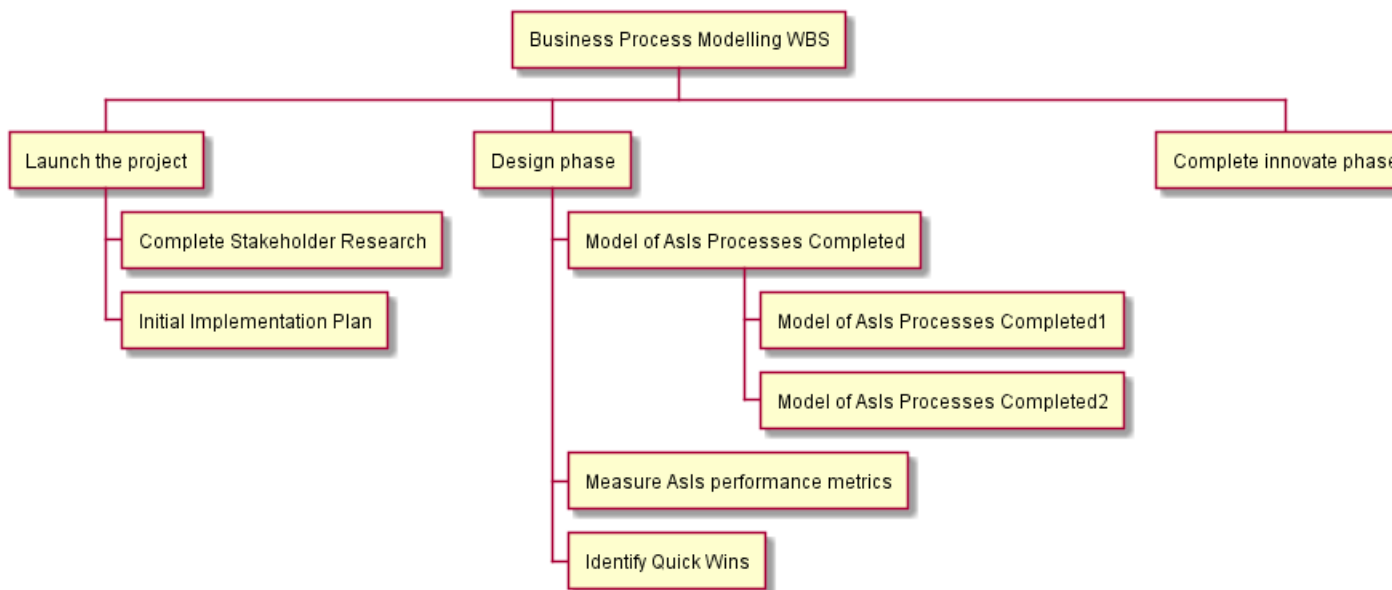
## 12 Work Breakdown Structure

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

### 12.1 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
```



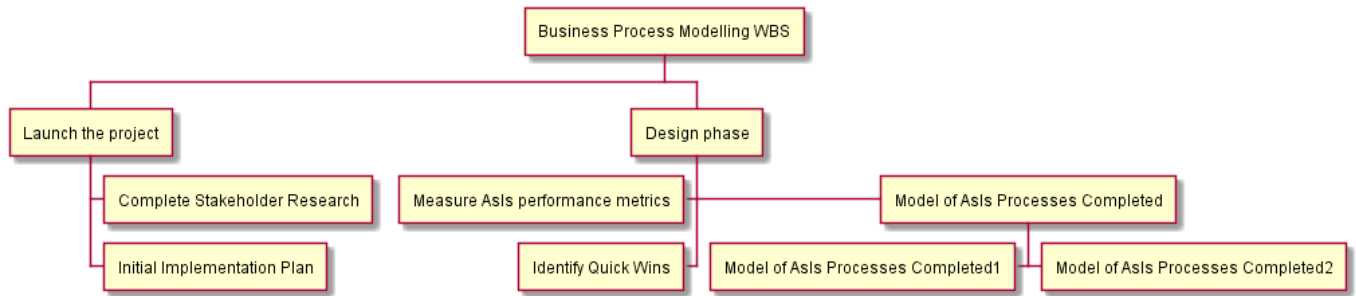
### 12.2 Change direction

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

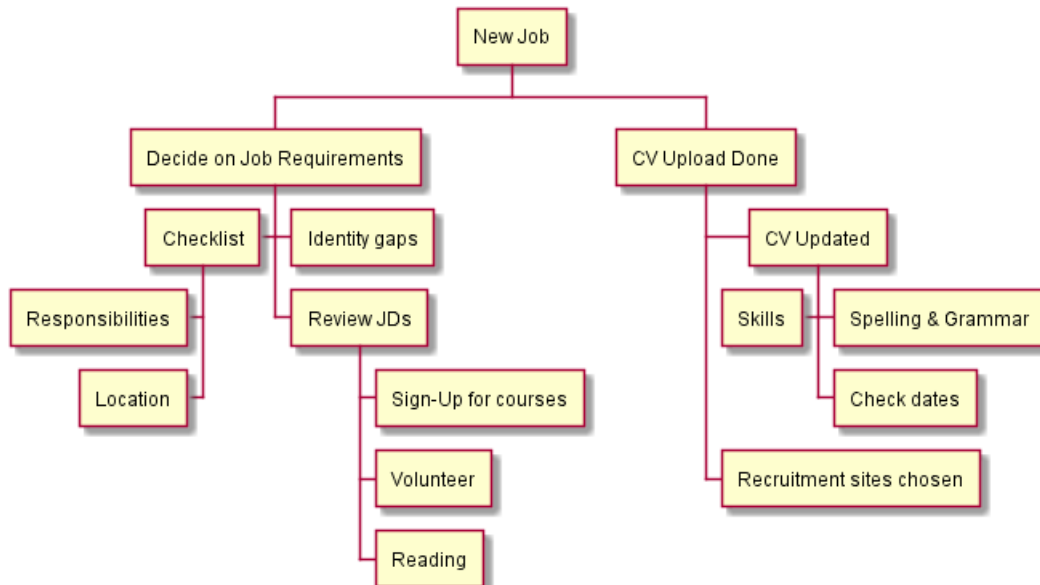


### 12.3 Arithmetic notation

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
    
```



You can use underscore \_ to remove box drawing.

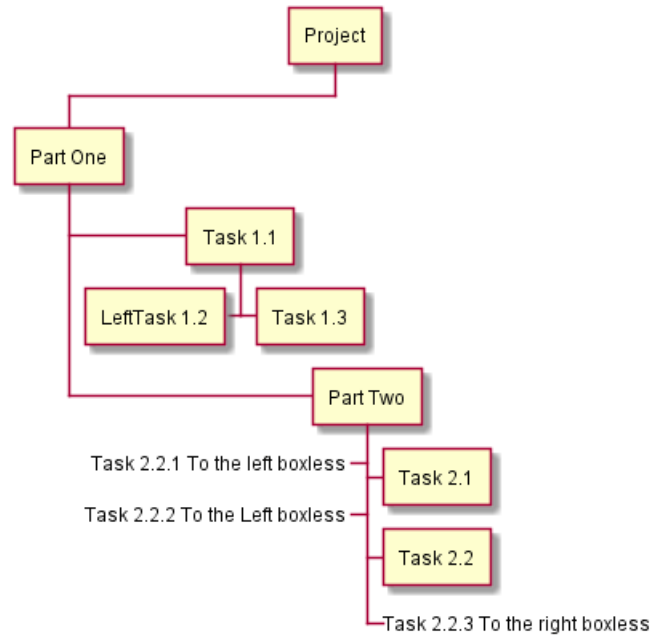
```

@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

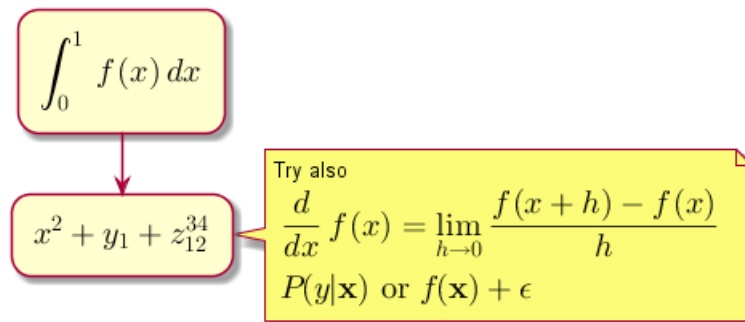
```



## 13 Maths

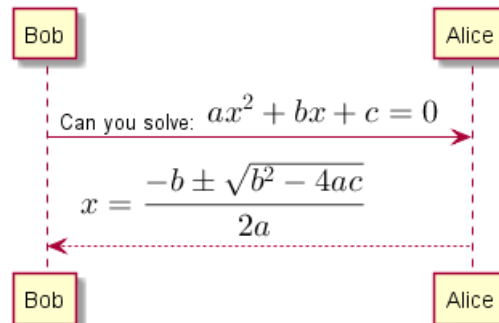
You can use AsciiMath or JLaTeXMath notation within PlantUML:

```
@startuml
:<math>int_0^1 f(x) dx</math>;
:<math>x^2+y_1+z_{12}^34</math>;
note right
Try also
<math>d/dxf(x)=lim_{h->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 = (-b+-sqrt(b^2-4ac))/(2a)</math>
@enduml
```



### 13.1 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
```



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

## 13.2 How is this working ?

To draw those formulas, PlantUML uses two OpenSource 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.

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*.



## 14 Common commands

### 14.1 Comments

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

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

### 14.2 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.

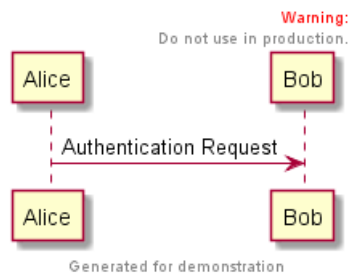
It is also possible to put some HTML into the header or footer.

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

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

```
center footer Generated for demonstration
```

```
@enduml
```



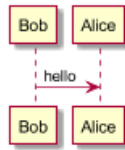
### 14.3 Zoom

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
```



## 14.4 Title

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

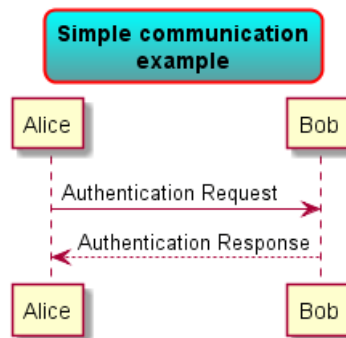
Some `skinparam` settings are available to put borders on the title.

```
@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.

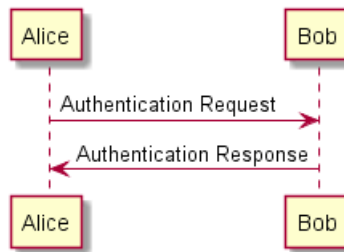
```
@startuml

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**



## 14.5 Caption

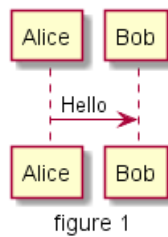
There is also a `caption` keyword to put a caption under the diagram.

```

@startuml

caption figure 1
Alice -> Bob: Hello

@enduml
  
```



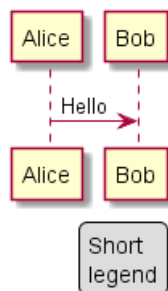
## 14.6 Legend the diagram

The `legend` and `end legend` are keywords is used to put a legend.

You can optionally specify to have `left`, `right`, `top`, `bottom` or `center` alignment for the legend.

```

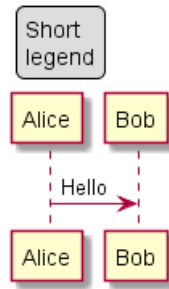
@startuml
Alice -> Bob : Hello
legend right
  Short
  legend
endlegend
@enduml
  
```



```

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

```
legend
endlegend
@enduml
```



## 15 Salt (wireframe)

Salt 그래픽인터페이스디자인을위한서브프로젝트이다.

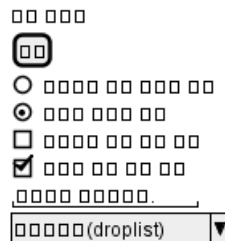
@startsalt 키워드를사용하거나, @startuml 아래에 salt 키워드를사용할수있다.

### 15.1 기본위젯

윈도우는반드시브라킷 ({, }) 으로감싸야한다. 그리고아래의형식으로사용한다.:

- 버튼 (Button) 은 [ 와 ].
- 라디오버튼 (Radio button) 은 ( 와 ).
- 체크박스 (Checkbox) 는 [ 와 ].
- 인풋박스 (User text area) 는 " .

```
@startuml
salt
{
  단순 텍스트
  [버튼]
  ( ) 체크되지 않은 라디오 버튼
  (X) 체크된 라디오 버튼
  [] 체크되지 않은 체크 박스
  [X] 체크된 않은 체크 박스
  "텍스트를 입력하세요. "
  ^드랍리스트(dropList)^
}
@enduml
```



이들은단순한샘플윈도우에대한토론을위해서사용된다.

### 15.2 그리드사용하기

테이블은 {를사용할때자동생성된다. 그리고컬럼을구분하기위해선 | 를사용해야한다.

예시:

```
@startsalt
{
  Login    | "MyName  "
  Password | "****   "
  [Cancel] | [ OK   ]
}
@sendsalt
```



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
```

### 15.3 Group box

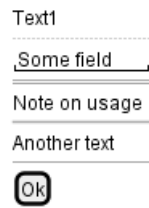
more info

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

### 15.4 Using separator

You can use several horizontal lines as separator.

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



## 15.5 Tree widget

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
```



## 15.6 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

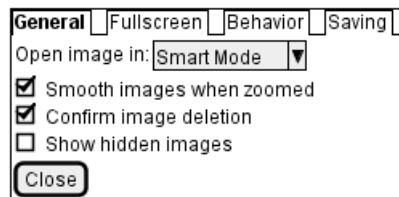
Superclass: java.lang.Object \_\_\_\_\_



## 15.7 Adding tabs

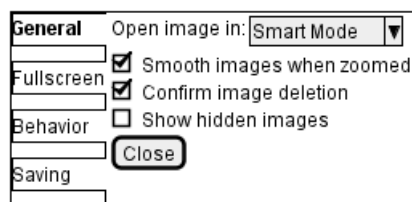
You can add tabs using {/ notation. Note that you can use HTML code to have bold text.

```
@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
```



## 15.8 Using menu

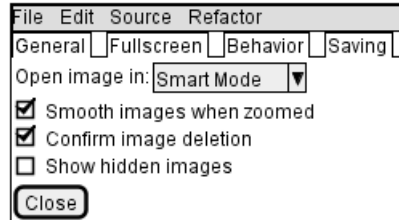
You can add a menu by using {\* notation.

```
@startsalt
{+
{* File | Edit | Source | Refactor }
{/ General | Fullscreen | Behavior | Saving }
{
{ Open image in: | ^Smart Mode^ }
[X] Smooth images when zoomed
```



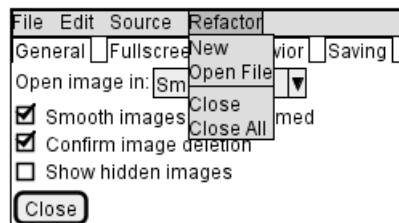


```
[X] Confirm image deletion
[ ] Show hidden images
}
[Close]
}
@endsalt
```



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
```



## 15.9 Advanced table

You can use two special notations for table :

- \* to indicate that a cell with span with left
- . to denote an empty cell

```
@startsalt
{#
. | Column 2 | Column 3
Row header 1 | value 1 | value 2
Row header 2 | A long cell | *
}
@endsalt
```

	Column 2	Column 3
Row header 1	value 1	value 2
Row header 2	A long cell	



## 15.10 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>`.

```
@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	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>	bold	code	expand-left	laptop	pencil	tablet
account-login	bolt	cog	expand-right	layers	person	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	text
align-right	browsers	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	calculator	credit-card	fork	loop-square	pulse	trash
arrow-circle-right	calendar	crop	fullscreen-enter	loop	puzzle-piece	underline
arrow-circle-top	camera-slr	dashboard	fullscreen-exit	magnifying-glass	question-mark	vertical-align-bottom
arrow-left	caret-bottom	data-transfer-download	globe	map-marker	rain	vertical-align-center
arrow-right	caret-left	data-transfer-upload	graph	map	random	vertical-align-top
arrow-thick-bottom	caret-right	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	cart	document	grid-two-up	media-record	resize-height	volume-low
arrow-thick-top	chat	dollar	hard-drive	media-skip-backward	resize-width	volume-off
arrow-top	check	double-quote-sans-left	header	media-skip-forward	rss-alt	warning
audio-spectrum	chevron-bottom	double-quote-sans-right	headphones	media-step-backward	rss	wifi
audio	chevron-left	double-quote-serif-left	heart	media-step-forward	script	wrench
badge	chevron-right	double-quote-serif-right	home	media-stop	share-boxed	x
ban	chevron-top	droplet	image	medical-cross	share	yen
bar-chart	circle-check	eject	inbox	menu	shield	zoom-in
basket	circle-x	elevator	infinity	microphone	signal	zoom-out
battery-empty	clipboard	ellipses	info	minus	signpost	
battery-full	clock	envelope-closed	italic	monitor	sort-ascending	
beaker	cloud-download	envelope-open	justify-center	moon	sort-descending	
	cloud-upload	euro	justify-left	move	spreadsheet	

## 15.11 Include Salt

see: <http://forum.plantuml.net/2427/salt-with-minimum-flowchat-capabilities?show=2427#q2427>

```
@startuml
(*) --> "
{{
salt
{+
<b>an example
choose one option
()one
```



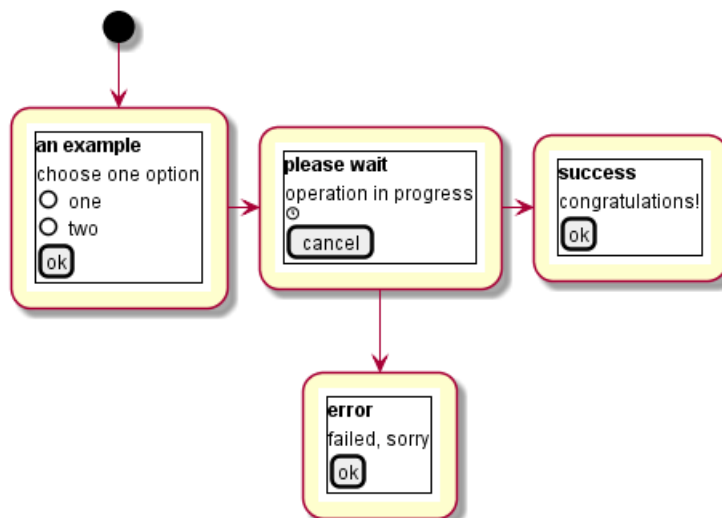
```

()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 function SALT($x)
"{{
salt
%invoke_void_func("_"+"$x)
}}" as $x
!endfunction

!function _choose()
{+
<b>an example
choose one option
()one
()two
[ok]
}
!endfunction

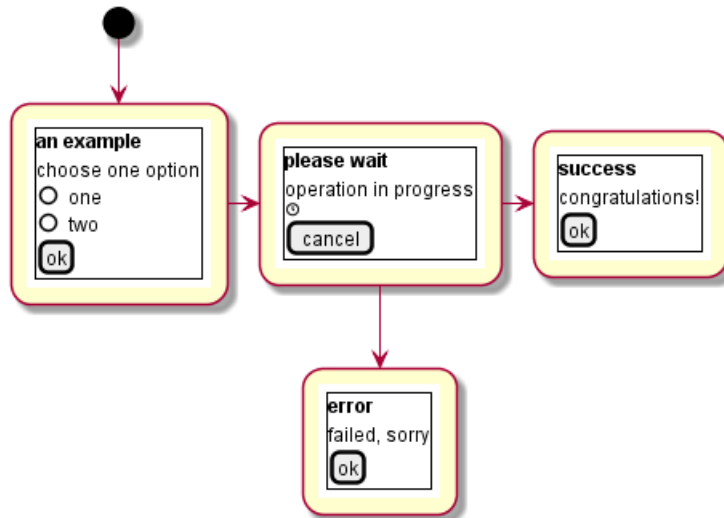
!function _wait()
{+
<b>please wait
operation in progress
<&clock>
[cancel]
}
!endfunction

!function _success()
{+
<b>success
congratulations!
[ok]
}
!endfunction

!function _error()
{+
<b>error
failed, sorry
[ok]
}
!endfunction

(*) --> SALT(choose)
-right-> SALT(wait)
wait -right-> SALT(success)
wait -down-> SALT(error)
@enduml

```



### 15.12 Scroll Bars

You can use "S" as scroll bar like in following examples:

```

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



```

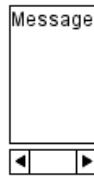
@startsalt
{SI
Message
.
.
.
.
}
@endsalt
    
```



```

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

```
.  
}  
@endsalt
```



## 16 Creole

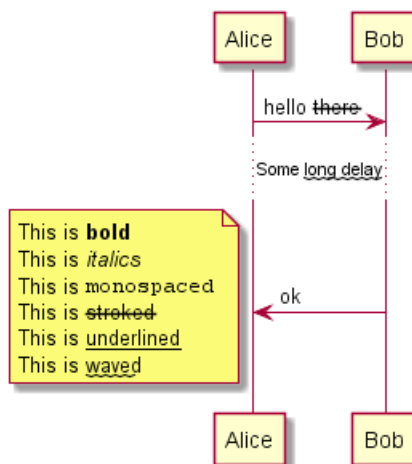
A light Creole engine has been integrated into PlantUML to have a standardized way of defining text style.

All diagrams are now supporting this syntax.

Note that ascending compatibility with HTML syntax is preserved.

### 16.1 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
```



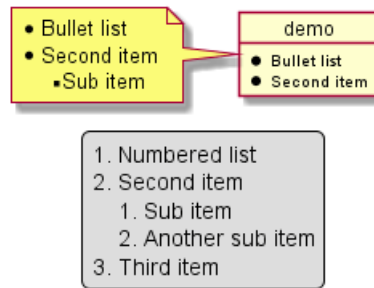
### 16.2 List

```
@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
```



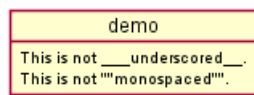
```
# Third item
end legend
@enduml
```



### 16.3 Escape character

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

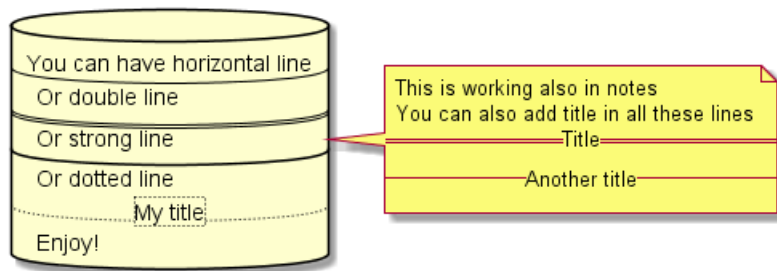
```
@startuml
object demo {
  This is not ~__underscored__.
  This is not ~""monospaced"".
}
@enduml
```



### 16.4 Horizontal lines

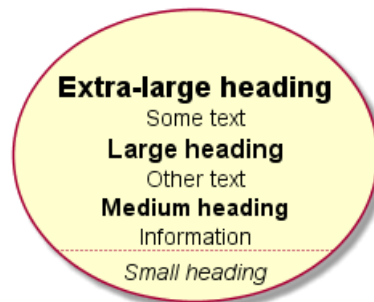
```
@startuml
database DB1 as "
You can have horizontal line
----
Or double line
====
Or strong line
----
Or dotted line
..My title..
Enjoy!
"
note right
  This is working also in notes
  You can also add title in all these lines
  ==Title==
  --Another title--
end note
@enduml
```





## 16.5 Headings

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



## 16.6 Legacy HTML

Some HTML tags are also working:

- `<b>` for bold text
- `<u>` or `<u:#AAAAAA>` or `<u:colorName>` for underline
- `<i>` for italic
- `<s>` or `<s:#AAAAAA>` or `<s:colorName>` for strike text
- `<w>` or `<w:#AAAAAA>` or `<w:colorName>` for wave underline text
- `<color:#AAAAAA>` or `<color:colorName>`
- `<back:#AAAAAA>` or `<back: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
```



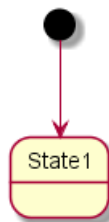
## 16.7 Table

It is possible to build table.

```
@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

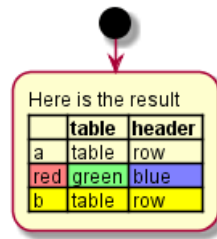
	table	header
a	table	row
b	table	row



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
```

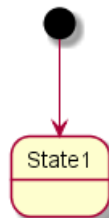
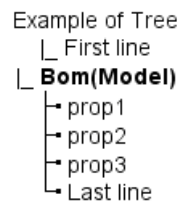




## 16.8 Tree

You can use |\_ characters to build a tree.

```
@startuml
skinparam titleFontSize 14
title
  Example of Tree
  |_ First line
  |_ Bom(Model)
  |_ prop1
  |_ prop2
  |_ prop3
  |_ Last line
end title
[*] --> State1
@enduml
```



## 16.9 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
```

## 16.10 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
```

### ♥Use of OpenIconic♥



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

```
@startuml
listopeniconic
@enduml
```

#### List Open Iconic

*Credit to*

<https://useiconic.com/open>

↪ account-login  
 ↪ account-logout  
 ↶ action-redo  
 ↷ action-undo  
 ≡ align-center  
 ≡ align-left  
 ≡ align-right  
 ⊙ aperture  
 ↓ arrow-bottom  
 ⊙ arrow-circle-bottom  
 ⊙ arrow-circle-left  
 ⊙ arrow-circle-right  
 ⊙ arrow-circle-top  
 ← arrow-left  
 → arrow-right  
 ↓ arrow-thick-bottom  
 ← arrow-thick-left  
 → arrow-thick-right  
 ↑ arrow-thick-top  
 ↑ arrow-top  
 🎧 audio-spectrum  
 🎵 audio  
 † badge  
 ⊙ ban  
 📊 bar-chart  
 🛒 basket  
 🔋 battery-empty  
 🔋 battery-full  
 🧴 beaker

🔔 bell  
 📶 bluetooth  
**B** bold  
 ⚡ bolt  
 📖 book  
 📌 bookmark  
 📦 box  
 📁 briefcase  
 £ british-pound  
 🌐 browser  
 🖱️ brush  
 🐛 bug  
 📣 bullhorn  
 📇 calculator  
 📅 calendar  
 📷 camera-slr  
 ▼ caret-bottom  
 ◀ caret-left  
 ▶ caret-right  
 ▲ caret-top  
 🛒 cart  
 💬 chat  
 ✓ check  
 ▼ chevron-bottom  
 ◀ chevron-left  
 ▶ chevron-right  
 ▲ chevron-top  
 ⊙ circle-check  
 ⊙ circle-x  
 📄 clipboard  
 🕒 clock  
 ☁ cloud-download  
 ☁ cloud-upload

☁ cloud  
 ☁ cloudy  
 ↻ code  
 ⚙ cog  
 ☰ collapse-down  
 ◀ collapse-left  
 ▶ collapse-right  
 ☰ collapse-up  
 ⚙ command  
 ◼ comment-square  
 Ⓞ compass  
 ⊙ contrast  
 ≡ copywriting  
 📇 credit-card  
 ✂ crop  
 📊 dashboard  
 ⬇ data-transfer-download  
 ⬆ data-transfer-upload  
 🗑 delete  
 📞 dial  
 📄 document  
 \$ dollar  
 " double-quote-sans-left  
 " double-quote-sans-right  
 " double-quote-serif-left  
 " double-quote-serif-right  
 💧 droplet  
 🗑 eject  
 ⬆ elevator  
 … ellipses  
 ✉ envelope-closed  
 ✉ envelope-open  
 € euro

≡ excerpt  
 ☰ expand-down  
 ◀ expand-left  
 ▶ expand-right  
 ☰ expand-up  
 🔗 external-link  
 👁 eye  
 🌊 eyedropper  
 📁 file  
 🔥 fire  
 🚩 flag  
 ⚡ flash  
 📁 folder  
 🍴 fork  
 🖥 fullscreen-enter  
 🖥 fullscreen-exit  
 🌐 globe  
 📊 graph  
 📊 grid-four-up  
 📊 grid-three-up  
 📊 grid-two-up  
 📀 hard-drive  
 📄 header  
 🎧 headphones  
 ♥ heart  
 🏠 home  
 🖼 image  
 📧 inbox  
 ∞ infinity  
 ⓘ info  
 / italic  
 ≡ justify-center  
 ≡ justify-left

≡ justify-right  
 ⌨ key  
 📖 laptop  
 📂 layers  
 💡 lightbulb  
 🔗 link-broken  
 🔗 link-intact  
 📋 list-rich  
 ≡ list  
 📍 location  
 🔒 lock-locked  
 🔓 lock-unlocked  
 ↻ loop-circular  
 ◻ loop-square  
 🔄 loop  
 🔍 magnifying-glass  
 📍 map-marker  
 🗺 map  
 ⏸ media-pause  
 ▶ media-play  
 ⏮ media-record  
 ⏪ media-skip-backward  
 ⏩ media-skip-forward  
 ⏮ media-step-backward  
 ⏭ media-step-forward  
 🛑 media-stop  
 ⚕ medical-cross  
 ≡ menu  
 🎤 microphone  
 - minus  
 📺 monitor  
 🌙 moon  
 + move

🎵 musical-note  
 📎 paperclip  
 ✎ pencil  
 👤 people  
 👤 person  
 📱 phone  
 🥧 pie-chart  
 📌 pin  
 🎮 play-circle  
 + plus  
 🔌 power-standby  
 🖨 print  
 📁 project  
 ⚡ pulse  
 🧩 puzzle-piece  
 ? question-mark  
 🌧 rain  
 ✖ random  
 🔄 reload  
 ↕ resize-both  
 ↕ resize-height  
 ↔ resize-width  
 📡 rss-alt  
 📡 rss  
 📜 script  
 📦 share-boxed  
 ➦ share  
 🛡 shield  
 📶 signal  
 📍 signpost  
 ↕ sort-ascending  
 ↕ sort-descending  
 📊 spreadsheet

★ star  
 ☀ sun  
 📱 tablet  
 🏷 tag  
 🏷 tags  
 🎯 target  
 📁 task  
 🖨 terminal  
 📄 text  
 👎 thumb-down  
 👍 thumb-up  
 ⌚ timer  
 ➦ transfer  
 🗑 trash  
underline  
 📏 vertical-align-bottom  
 📏 vertical-align-center  
 📏 vertical-align-top  
 📺 video  
 🔊 volume-high  
 🔊 volume-low  
 🔊 volume-off  
 ⚠ warning  
 📶 wifi  
 🛠 wrench  
 ✖ x  
 🗓 yen  
 🔍 zoom-in  
 🔍 zoom-out

## 17 Defining and using sprites

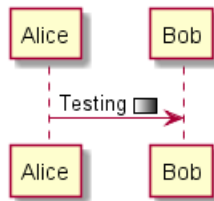
A *Sprite* is a small graphic element that can be used in diagrams.

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.

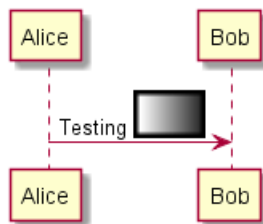
Then you can use the sprite using <\$XXX> where XXX is the name of the sprite.

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



## 17.1 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.

## 17.2 Importing Sprite

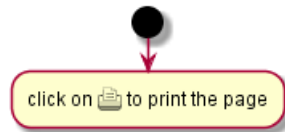
You can also launch the GUI to generate a sprite from an existing image.

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.

## 17.3 Examples

```
@startuml
sprite $printer [15x15/8z] N0tH3W0W208HxFz_kMAhj71HWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvF
start
:click on <$printer> to print the page;
@enduml
```



```
@startuml
sprite $bug [15x15/16z] PKzR2i0m2BFMi15p__FEjQEqB1z27aeqCqixa8S40T7C53cKpsHpaYPDJY_12MHM-BLRyywPhrrL
sprite $printer [15x15/8z] N0tH3W0W208HxFz_kMAhj71HWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvF
sprite $disk {
  444445566677881
  43600000009991
  4360000000ACA1
  5370000001A7A1
  53700000012B8A1
  538000000123B8A1
  638000001233C9A1
  634999AABBC99B1
  744566778899AB1
  7456AAAAA99AAB1
  8566AFC228AABB1
  8567AC8118BBBB1
  867BD4433BBBBB1
  39AAAAABBBBBBC1
}
```

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

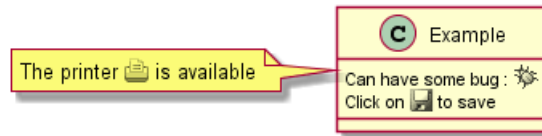
```
class Example {
  Can have some bug : <$bug>
  Click on <$disk> to save
}
```



```
note left : The printer <$printer> is available
```

```
@enduml
```

### Use of sprites (🖨️, 🐛...)



## 18 Skinparam command

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

Example:

```
skinparam backgroundColor transparent
```

### 18.1 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.

### 18.2 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
}
```

### 18.3 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
```





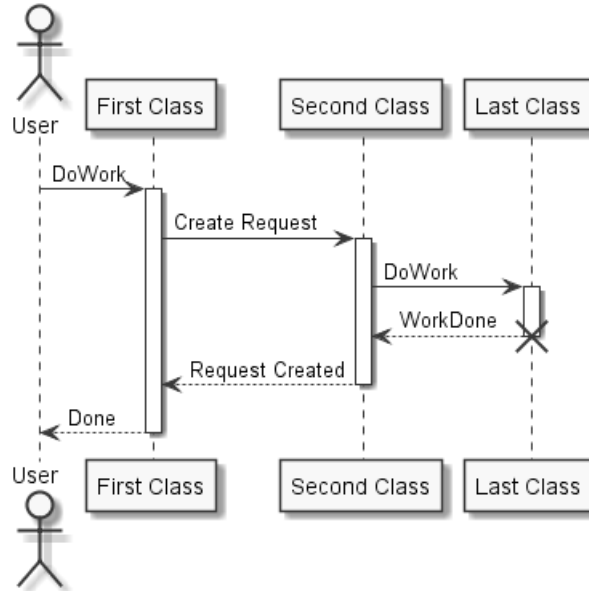
```

deactivate B

A --> User: Done
deactivate A

@enduml

```



### 18.4 Shadowing

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

```

@startuml

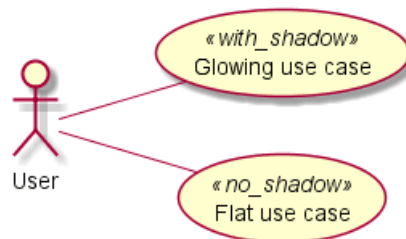
left to right direction

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

```



## 18.5 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

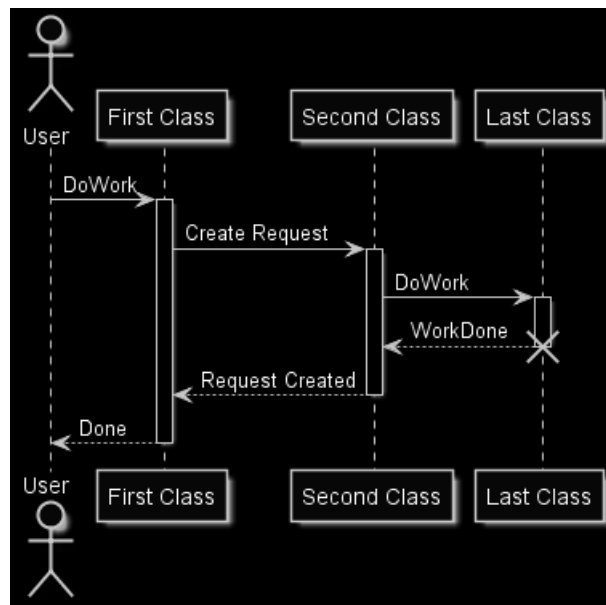
A -> B: Create Request
activate B

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

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml
```



## 18.6 Colors

You can use either standard color name or RGB code.

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.

## 18.7 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
```

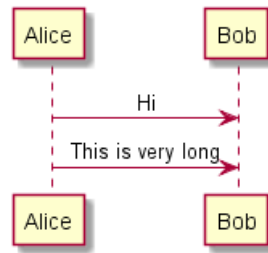
## 18.8 Text Alignment

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
```





## 18.9 Examples

```

@startuml
skinparam backgroundColor #EEEEBD
skinparam handwritten true

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

ParticipantBorderColor DeepSkyBlue
ParticipantBackgroundColor DodgerBlue
ParticipantFontName Impact
ParticipantFontSize 17
ParticipantFontColor #A9DCDF

ActorBackgroundColor aqua
ActorFontColor DeepSkyBlue
ActorFontSize 17
ActorFontName Aapex
}

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

User -> A: DoWork
activate A

A -> B: Create Request
activate B

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

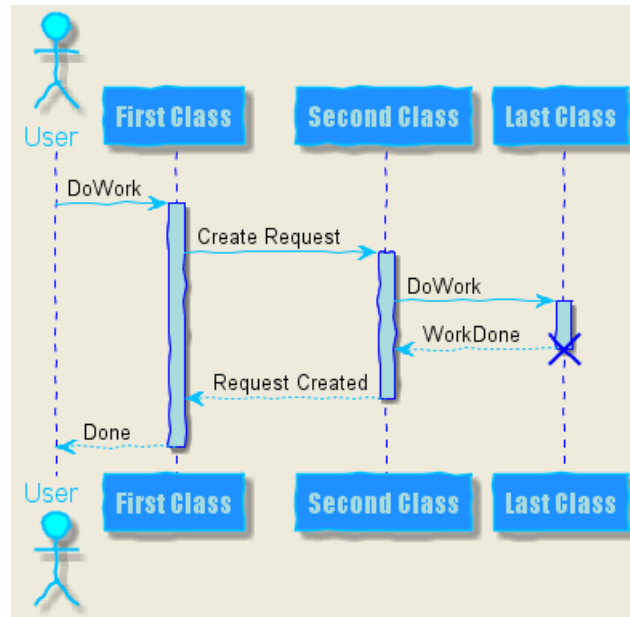
B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml

```





```

@startuml
skinparam handwritten true

skinparam actor {
  BorderColor black
  FontName Courier
  BackgroundColor<< Human >> Gold
}

skinparam usecase {
  BackgroundColor DarkSeaGreen
  BorderColor DarkSlateGray

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

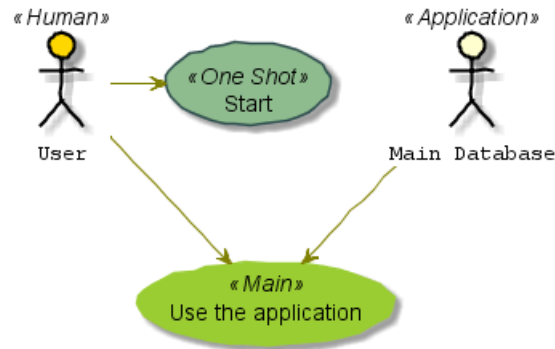
  ArrowColor Olive
}

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

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

MySql --> (Use)
@enduml

```



```

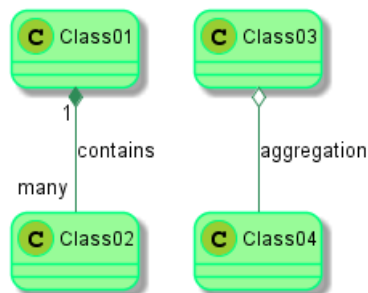
@startuml
skinparam roundcorner 20
skinparam class {
  BackgroundColor PaleGreen
  ArrowColor SeaGreen
  BorderColor SpringGreen
}
skinparam stereotypeCBackgroundColor YellowGreen

```

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

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

```
@enduml
```



```

@startuml
skinparam interface {
  backgroundColor RosyBrown
  borderColor orange
}

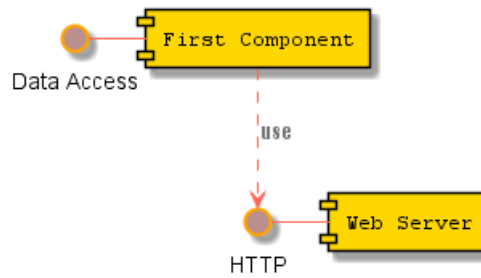
skinparam component {
  FontSize 13
  BackgroundColor<<Apache>> Red
  BorderColor<<Apache>> #FF6655
  FontName Courier
  BorderColor black
  BackgroundColor gold
  ArrowFontName Impact
  ArrowColor #FF6655
  ArrowFontColor #777777
}

```

```
() "Data Access" as DA
```

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

```
HTTP - [Web Server] << Apache >>
@enduml
```

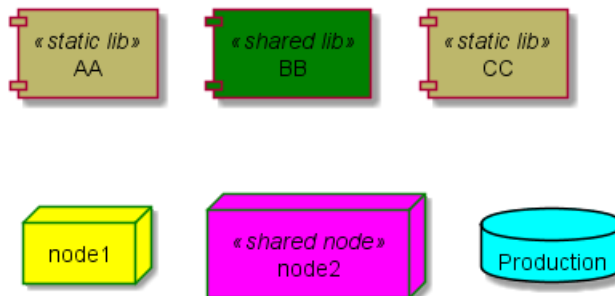


```
@startuml
[AA] <<static lib>>
[BB] <<shared lib>>
[CC] <<static lib>>

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
```



## 18.10 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:

```
@startuml
help skinparams
@enduml
```

That will give you the following result:

**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
- ArrowThickness
- ArtifactBackgroundColor





You can also view each skinparam parameters with its results displayed at <https://plantuml-documentation.readthedocs.io/en/latest/formatting/all-skin-params.html>.

## 19 Preprocessing

Some minor 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 !.

### 19.1 Migration notes

The actual 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` and variable definition instead. `!define` should be replaced by `return function` and `!definelong` should be replaced by `void function`.
- `!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.

### 19.2 Variable definition

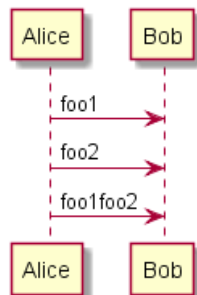
Although this is not mandatory, we highly suggest that variable names start with a \$. There are two types of data:

- Integer number
- String - 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"
!global $ef = $ab + $cd
```

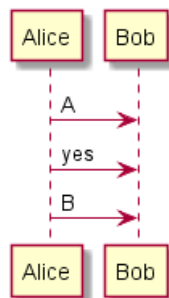
```
Alice -> Bob : $ab
Alice -> Bob : $cd
Alice -> Bob : $ef
@enduml
```



## 19.3 Conditions

- You can use expression in condition.
- *else* is 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
```



## 19.4 Void function

- Function names *must* start with a \$
- Argument names *must* start with a \$
- Void functions can call other void functions

Example:

```
@startuml
!function msg($source, $destination)
$source --> $destination
!endfunction

!function init_class($name)
class $name {
$addCommonMethod()
}
!endfunction

!function $addCommonMethod()
    toString()
    hashCode()
!endfunction

init_class("foo1")
init_class("foo2")
msg("foo1", "foo2")
@enduml
```





Variables defined in functions are **local**. It means that the variable is destroyed when the function ends.

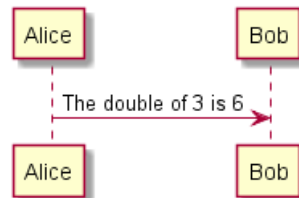
## 19.5 Return function

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 function
- from other void function
- Function name *should* start by a \$
- Argument names *should* start by a \$

```
@startuml
!function $double($a)
!return $a + $a
!endfunction
```

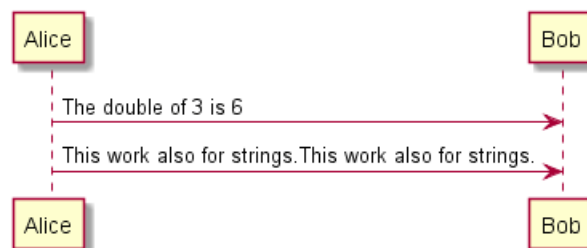
```
Alice -> Bob : The double of 3 is $double(3)
@enduml
```



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
```

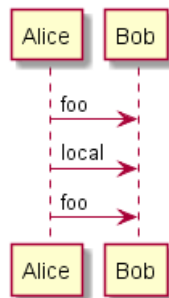


As in void function, variables are local by default (they are destroyed when the function is exited). However, you can access global variables from a function. However, you can use the `local` keyword to create a local variable if even a global variable exists with the same name.

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

!global $ijk = "foo"

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

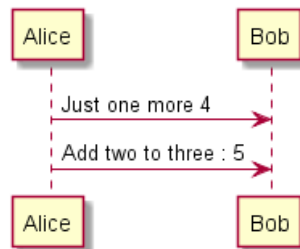


## 19.6 Default argument value

In both return and void 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
```

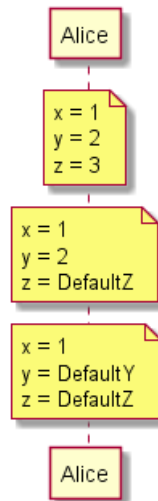


Only arguments at the end of the parameter list can have default values.

```
@startuml
!function defaulttest($x, $y="DefaultY", $z="DefaultZ")
note over Alice
  x = $x
  y = $y
  z = $z
end note
!endfunction
```



```
defaulttest(1, 2, 3)
defaulttest(1, 2)
defaulttest(1)
@enduml
```

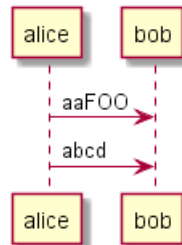


## 19.7 Unquoted function

By default, you have to put quotes when you call a function. It is possible to use the `unquoted` keyword to indicate that a function 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
```



## 19.8 Including files or URL

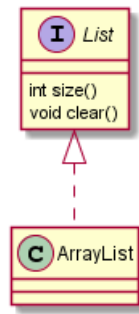
Use the `!include` directive to include file in your diagram. Using URL, you can also include file from Internet/Intranet.

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

!include List.iuml
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.

## 19.9 Including Subpart

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
```

## 19.10 Builtin functions

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

Name	Description	
%strlen	Calculate the length of a String	%
%substr	Extract a substring. Takes 2 or 3 arguments  %substr("abcdef", 3, 2)	"d
%strpos	Search a substring in a string	%strp
%intval	Convert a String to Int	%
%file_exists	Check if a file exists on the local filesystem	%file_exis
%function_exists	Check if a function exists	%function_e
%variable_exists	Check if a variable exists	%variable_
%set_variable_value	Set a global variable	%set_variable_valu
%get_variable_value	Retrieve some variable value	%get_variab.
%getenv	Retrieve environment variable value	%
%dirpath	Retrieve current dirpath	
%filename	Retrieve current filename	
%date	Retrieve current date. You can provide an optional format for the date	%date("y
%true	Return always true	
%false	Return always false	
%not	Return the logical negation of an expression	

## 19.11 Logging

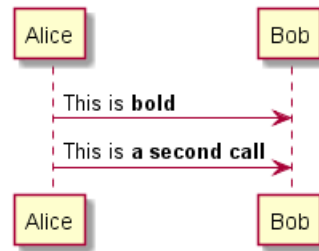
You can use !log to add some log output when generating the diagram. This has no impact at all on the diagram itself. However, those logs are printed in the command line's output stream. This could be useful for debug purpose.

```
@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
```







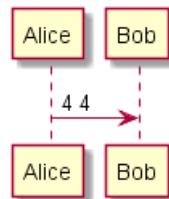
## 19.12 Memory dump

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
  
```



## 19.13 Assertion

You can put assertion in your diagram.

```

@startuml
Alice -> Bob : Hello
!assert %strpos("abcdef", "cd")==3 : "This always fail"
@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 <http://plantuml.com>

```
[From string (line 3) ]
@startuml
Alice -> Bob : Hello
!assert %strpos("abcdef", "cd")==3 : "This always fail"
Assertion error : This always fail
```

## 19.14 Building custom library

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

...
```

## 19.15 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.

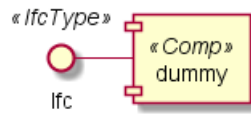
## 19.16 Argument concatenation

It is possible to append text to a macro argument using the `##` syntax.

```
@startuml
!unquoted function COMP_TEXTGENCOMP(name)
[name] << Comp >>
interface Ifc << IfcType >> AS name##Ifc
name##Ifc - [name]
!endfunction
```



```
COMP_TEXTGENCOMP(dummy)
@enduml
```



## 19.17 Dynamic function invocation

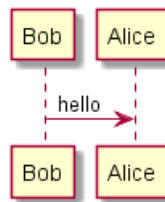
You can dynamically invoke a void function using the special `%invoke_void_func()` void function. This function takes as first argument the name of the actual void function to be called. The following argument are copied to the called function.

For example, you can have:

```
@startuml
!function $go()
  Bob -> Alice : hello
!endfunction

!$wrapper = "$go"

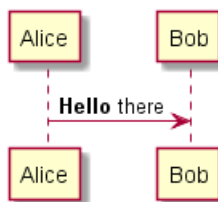
%invoke_void_func($wrapper)
@enduml
```



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
```



## 20 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*.

### 20.1 Examples

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

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

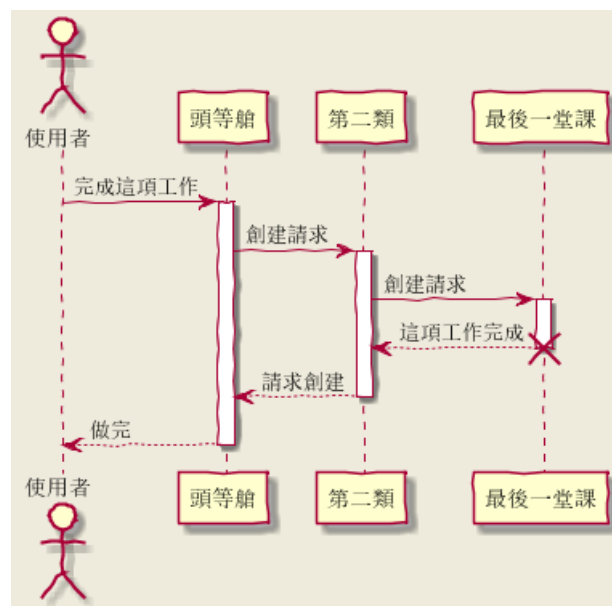
使用者 -> A: 完成這項工作
activate A

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
```



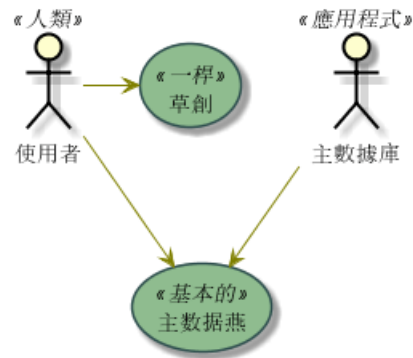
```
@startuml
```

```
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
```



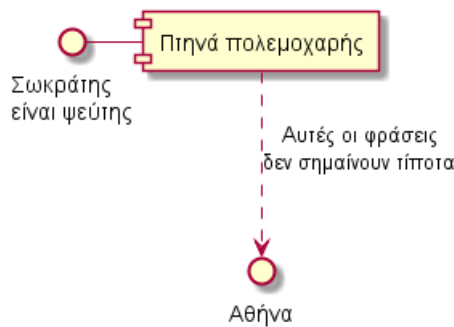
@startuml

() "Σωκράτης ψεύτης" as Σωκράτης

Σωκράτης - [Πτηνά πολεμοχαρής]

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

@enduml



## 20.2 Charset

The default charset used when *reading* the text files containing the UML text description is system dependent.

Normally, it should just be fine, but in some case, you may want to use another charset. For example, with the command line:

```
java -jar plantuml.jar -charset UTF-8 files.txt
```

Or, with the ant task:

```
<!-- Put images in c:/images directory -->
<target name="main">
<plantuml dir="./src" charset="UTF-8" />
```

Depending on your Java installation, the following charset should be available: ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, UTF-16.



## 21 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!

### 21.1 AWS library

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

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

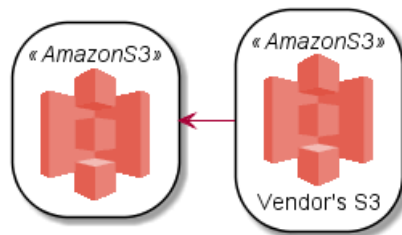
Use it by including the file that contains the sprite, eg: `!include <aws/Storage/AmazonS3/AmazonS3>`. When imported, you can use the sprite as normally you would, using `<$sprite_name>`.

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
```



### 21.2 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>
```

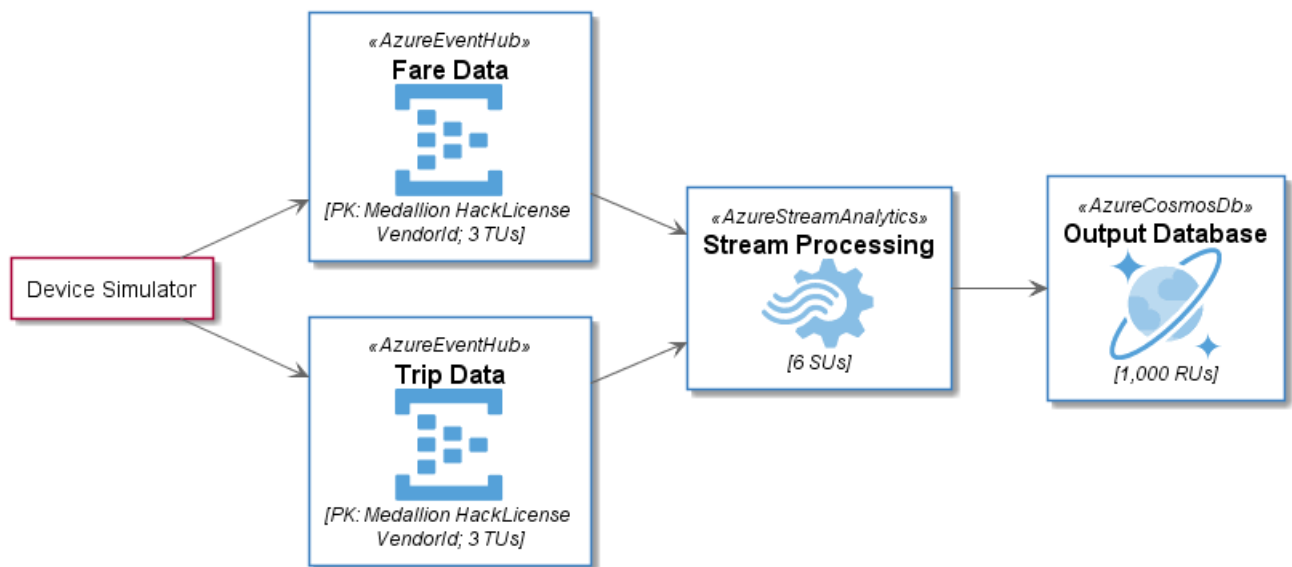
left to right direction



```
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
```



## 21.3 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

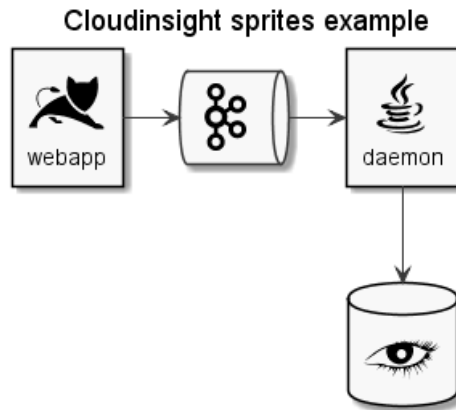
rectangle "<$tomcat>\nwebapp" as webapp
queue "<$kafka>" as kafka
rectangle "<$java>\ndaemon" as daemon
database "<$cassandra>" as cassandra

webapp -> kafka
kafka -> daemon
daemon --> cassandra
```





```
@enduml
```



## 21.4 Devicons and Font Awesome library

<https://github.com/tupadr3/plantuml-icon-font-sprites>

These two library consists respectively of Devicons and Font Awesome libraries of icons.

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>

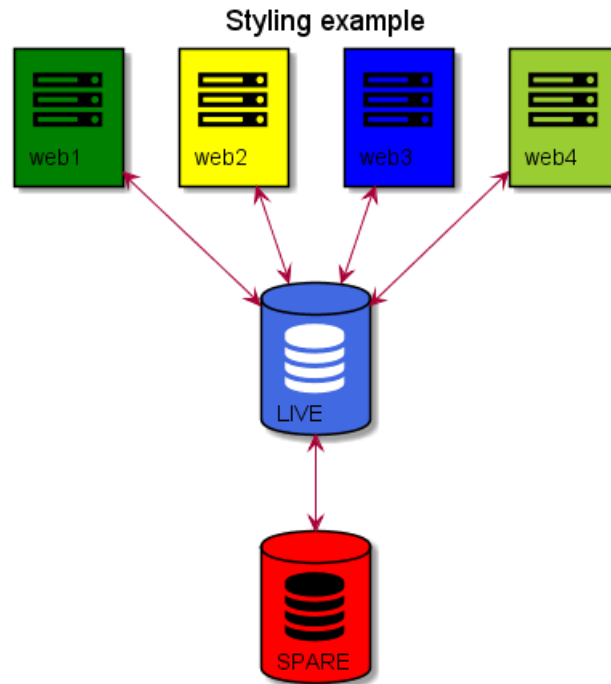
title Styling example

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
  
```

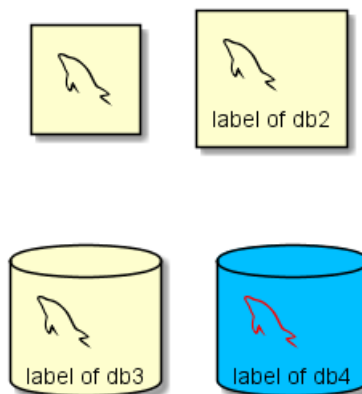


```

@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

```



## 21.5 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

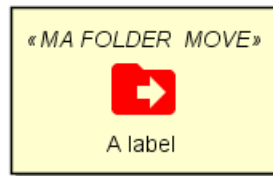


again the use of the prefix MA\_.

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
```



## 21.6 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>
```



```

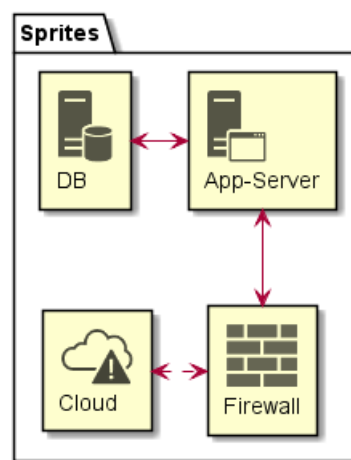
title Office Icons Example

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

```

Office Icons Example



```

@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]
}

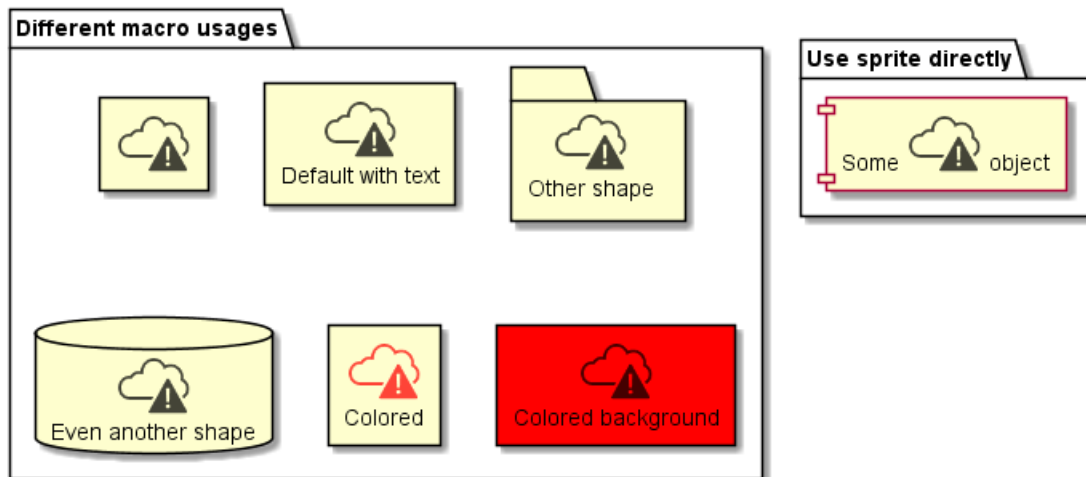
package "Different macro usages" {
  OFF_CLOUD_DISASTER_RED(cloud1)
  OFF_CLOUD_DISASTER_RED(cloud2,Default with text)
  OFF_CLOUD_DISASTER_RED(cloud3,Other shape,Folder)
  OFF_CLOUD_DISASTER_RED(cloud4,Even another shape,Database)
  OFF_CLOUD_DISASTER_RED(cloud5,Colored,Rectangle, red)
  OFF_CLOUD_DISASTER_RED(cloud6,Colored background) #red
}

@enduml

```



## Extended Office Icons Example



## 21.7 ArchiMate

<https://github.com/ebbypeter/ArchiMate-PlantUML>

This repository contains ArchiMate PlantUML macros and other includes for creating ArchiMate Diagrams easily and consistently.

```
@startuml
!includeurl https://raw.githubusercontent.com/ebbypeter/ArchiMate-PlantUML/master/ArchiMate.puml

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)")

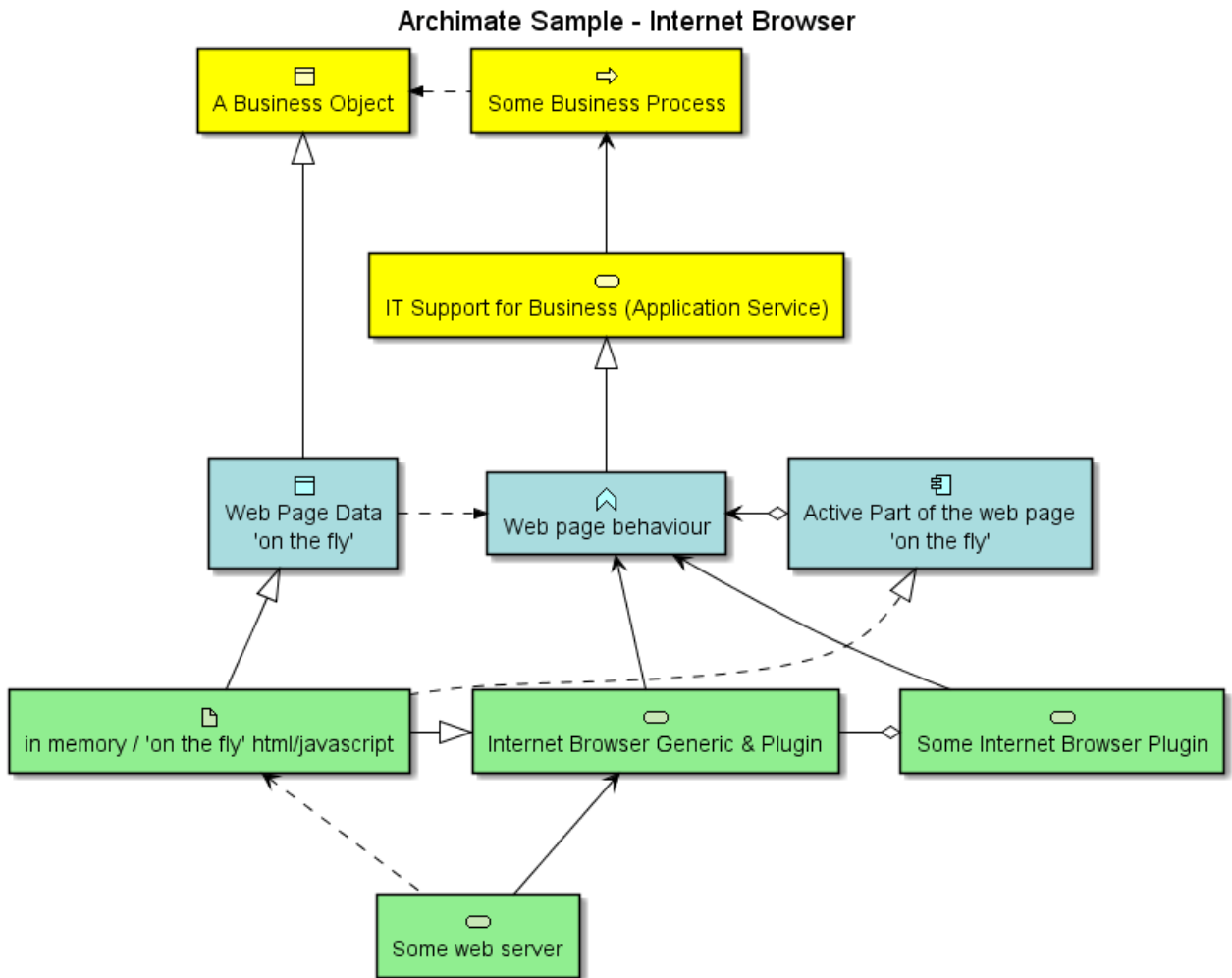
Application_DataObject(dataObject, "Web Page Data \n 'on the fly'")
Application_Function(webpageBehaviour, "Web page behaviour")
Application_Component(ActivePartWebPage, "Active Part of the web page \n 'on the fly'")

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
```



## 21.8 Miscellaneous

You can list standard library folders using the special diagram:

```
@startuml
stdlib
@enduml
```

**aws**

Version 18.02.22

Delivered by <https://github.com/milo-minderbinder/AWS-PlantUML>**awslib**

Version 3.0.0

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 0.0.1

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

Version 0.0.1

Delivered by <https://github.com/cloudogu/plantuml-cloudogu-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 2.0.0

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>시퀀스 다이어그램</b>	<b>1</b>
1.1	기본예제 . . . . .	1
1.2	참여자 (participant) 선언 . . . . .	1
1.3	참여자에서특수문자사용하기 . . . . .	3
1.4	자신에게메시지보내기 . . . . .	3
1.5	화살표스타일변경 . . . . .	3
1.6	화살표색상변경 . . . . .	4
1.7	메시지순서번호매기기 . . . . .	4
1.8	페이지제목, 머리말과꼬리말 . . . . .	7
1.9	다이어그램분리 . . . . .	7
1.10	메세지그룹화 . . . . .	8
1.11	메시지에노트추가하기 . . . . .	9
1.12	다른형태의노트들 . . . . .	10
1.13	노트모양바꾸기 . . . . .	10
1.14	Creole 과 HTML . . . . .	11
1.15	구분자 . . . . .	12
1.16	참조 . . . . .	13
1.17	지연 . . . . .	13
1.18	공백 . . . . .	14
1.19	생명선활성화및비활성화 . . . . .	15
1.20	리턴 . . . . .	16
1.21	참여자생성 . . . . .	16
1.22	Incoming and outgoing messages . . . . .	17
1.23	Stereotypes and Spots . . . . .	18
1.24	More information on titles . . . . .	19
1.25	Participants encompass . . . . .	20
1.26	Removing Footer . . . . .	21
1.27	Skinparam . . . . .	21
1.28	Changing padding . . . . .	23
<b>2</b>	<b>유즈케이스 다이어그램</b>	<b>25</b>
2.1	유즈케이스 . . . . .	25
2.2	Actors . . . . .	25
2.3	유즈케이스종류 . . . . .	26
2.4	기본예제 . . . . .	26
2.5	Extension . . . . .	27
2.6	Using notes . . . . .	27
2.7	Stereotypes . . . . .	28
2.8	화살표방향변경 . . . . .	29
2.9	Splitting diagrams . . . . .	30
2.10	Left to right direction . . . . .	30
2.11	Skinparam . . . . .	31
2.12	Complete example . . . . .	32
<b>3</b>	<b>클래스 다이어그램</b>	<b>33</b>
3.1	클래스관계 . . . . .	33
3.2	관계를나타내기위한레이블 . . . . .	34
3.3	Adding methods . . . . .	35
3.4	Defining visibility . . . . .	36
3.5	Abstract and Static . . . . .	36
3.6	Advanced class body . . . . .	37
3.7	Notes and stereotypes . . . . .	37
3.8	More on notes . . . . .	38
3.9	Note on links . . . . .	39
3.10	Abstract class and interface . . . . .	40
3.11	Using non-letters . . . . .	41
3.12	Hide attributes, methods... . . . .	41



3.13	Hide classes	42
3.14	Use generics	43
3.15	Specific Spot	43
3.16	Packages	43
3.17	Packages style	44
3.18	Namespaces	45
3.19	Automatic namespace creation	46
3.20	Lollipop interface	47
3.21	Changing arrows direction	47
3.22	Association classes	48
3.23	Skinparam	49
3.24	Skinned Stereotypes	49
3.25	Color gradient	50
3.26	Help on layout	51
3.27	대용량파일분할하기	51
<b>4</b>	<b>Activity Diagram</b>	<b>53</b>
4.1	Simple Activity	53
4.2	화살표라벨	53
4.3	Changing arrow direction	53
4.4	Branches	54
4.5	브랜치에덧붙임	55
4.6	Synchronization	56
4.7	Long activity description	57
4.8	Notes	57
4.9	Partition	58
4.10	Skinparam	59
4.11	Octagon	60
4.12	Complete example	60
<b>5</b>	<b>Activity Diagram (beta)</b>	<b>63</b>
5.1	Simple Activity	63
5.2	Start/Stop	63
5.3	Conditional	64
5.4	Repeat loop	65
5.5	While loop	65
5.6	Parallel processing	66
5.7	Notes	67
5.8	Colors	67
5.9	Arrows	68
5.10	Connector	69
5.11	Grouping	69
5.12	Swimlanes	70
5.13	Detach	71
5.14	SDL	72
5.15	Complete example	73
<b>6</b>	<b>컴포넌트 다이어그램</b>	<b>75</b>
6.1	컴포넌트	75
6.2	인터페이스	75
6.3	기본예제	76
6.4	메모사용하기	76
6.5	컴포넌트를그룹으로나누기	76
6.6	화살표방향바꾸기	78
6.7	Use UML2 notation	79
6.8	Long description	80
6.9	Individual colors	80
6.10	Using Sprite in Stereotype	80
6.11	Skinparam	81

<b>7</b>	<b>상태 다이어그램</b>	<b>83</b>
7.1	간단한 상태	83
7.2	Change state rendering	83
7.3	상태 수정	84
7.4	긴 이름	85
7.5	Concurrent state	86
7.6	Arrow direction	87
7.7	Note	88
7.8	More in notes	89
7.9	Skinparam	89
<b>8</b>	<b>Object Diagram</b>	<b>91</b>
8.1	Definition of objects	91
8.2	Relations between objects	91
8.3	Adding fields	91
8.4	Common features with class diagrams	92
<b>9</b>	<b>Timing Diagram</b>	<b>93</b>
9.1	Declaring participant	93
9.2	Adding message	93
9.3	Relative time	94
9.4	Participant oriented	95
9.5	Setting scale	95
9.6	Initial state	95
9.7	Intricated state	96
9.8	Hidden state	97
9.9	Adding constraint	97
9.10	Adding texts	98
<b>10</b>	<b>Gantt Diagram</b>	<b>99</b>
10.1	Declaring tasks	99
10.2	Adding constraints	99
10.3	Short names	99
10.4	Customize colors	100
10.5	Milestone	100
10.6	Calendar	100
10.7	Close day	100
10.8	Simplified task succession	101
10.9	Separator	101
10.10	Working with resources	101
10.11	Complex example	102
<b>11</b>	<b>MindMap</b>	<b>103</b>
11.1	OrgMode syntax	103
11.2	Removing box	103
11.3	Arithmetic notation	104
11.4	Markdown syntax	104
11.5	Changing diagram direction	105
11.6	Complete example	105
<b>12</b>	<b>Work Breakdown Structure</b>	<b>107</b>
12.1	OrgMode syntax	107
12.2	Change direction	107
12.3	Arithmetic notation	108
<b>13</b>	<b>Maths</b>	<b>110</b>
13.1	Standalone diagram	110
13.2	How is this working ?	111



<b>14 Common commands</b>	<b>112</b>
14.1 Comments . . . . .	112
14.2 Footer and header . . . . .	112
14.3 Zoom . . . . .	112
14.4 Title . . . . .	113
14.5 Caption . . . . .	114
14.6 Legend the diagram . . . . .	114
<b>15 Salt (wireframe)</b>	<b>116</b>
15.1 기본위젯 . . . . .	116
15.2 그리드사용하기 . . . . .	116
15.3 Group box . . . . .	117
15.4 Using separator . . . . .	117
15.5 Tree widget . . . . .	118
15.6 Enclosing brackets . . . . .	118
15.7 Adding tabs . . . . .	119
15.8 Using menu . . . . .	119
15.9 Advanced table . . . . .	120
15.10OpenIconic . . . . .	121
15.11Include Salt . . . . .	121
15.12Scroll Bars . . . . .	124
<b>16 Creole</b>	<b>126</b>
16.1 Emphasized text . . . . .	126
16.2 List . . . . .	126
16.3 Escape character . . . . .	127
16.4 Horizontal lines . . . . .	127
16.5 Headings . . . . .	128
16.6 Legacy HTML . . . . .	128
16.7 Table . . . . .	129
16.8 Tree . . . . .	130
16.9 Special characters . . . . .	130
16.10OpenIconic . . . . .	130
<b>17 Defining and using sprites</b>	<b>132</b>
17.1 Encoding Sprite . . . . .	133
17.2 Importing Sprite . . . . .	133
17.3 Examples . . . . .	133
<b>18 Skinparam command</b>	<b>135</b>
18.1 Usage . . . . .	135
18.2 Nested . . . . .	135
18.3 Black and White . . . . .	135
18.4 Shadowing . . . . .	136
18.5 Reverse colors . . . . .	137
18.6 Colors . . . . .	137
18.7 Font color, name and size . . . . .	138
18.8 Text Alignment . . . . .	138
18.9 Examples . . . . .	139
18.10List of all skinparam parameters . . . . .	142
<b>19 Preprocessing</b>	<b>145</b>
19.1 Migration notes . . . . .	145
19.2 Variable definition . . . . .	145
19.3 Conditions . . . . .	146
19.4 Void function . . . . .	146
19.5 Return function . . . . .	147
19.6 Default argument value . . . . .	148
19.7 Unquoted function . . . . .	149

19.8 Including files or URL . . . . .	149
19.9 Including Subpart . . . . .	150
19.10 Builtin functions . . . . .	151
19.11 Logging . . . . .	151
19.12 Memory dump . . . . .	152
19.13 Assertion . . . . .	152
19.14 Building custom library . . . . .	153
19.15 Search path . . . . .	153
19.16 Argument concatenation . . . . .	153
19.17 Dynamic function invocation . . . . .	154
<b>20 Unicode</b>	<b>155</b>
20.1 Examples . . . . .	155
20.2 Charset . . . . .	157
<b>21 Standard Library</b>	<b>158</b>
21.1 AWS library . . . . .	158
21.2 Azure library . . . . .	158
21.3 Cloud Insight . . . . .	159
21.4 Devicons and Font Awesome library . . . . .	160
21.5 Google Material Icons . . . . .	161
21.6 Office . . . . .	162
21.7 ArchiMate . . . . .	164
21.8 Miscellaneous . . . . .	165