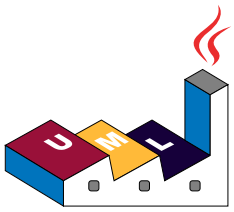
Drawing UML with PlantUML



PlantUML Language Reference Guide

(Version 1.2019.1)

PlantUML is a component that allows to quickly write :

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

The following non-UML diagrams are also supported:

- Wireframe graphical interface
- Archimate diagram
- Specification and Description Language (SDL)
- Ditaa diagram
- Gantt diagram
- Mathematic with AsciiMath or JLaTeXMath notation

Diagrams are defined using a simple and intuitive language.

1 Sequence Diagram

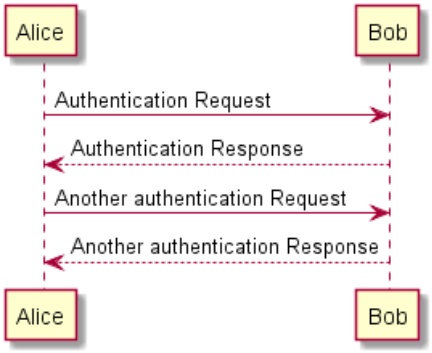
1.1 Basic examples

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

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

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

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

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



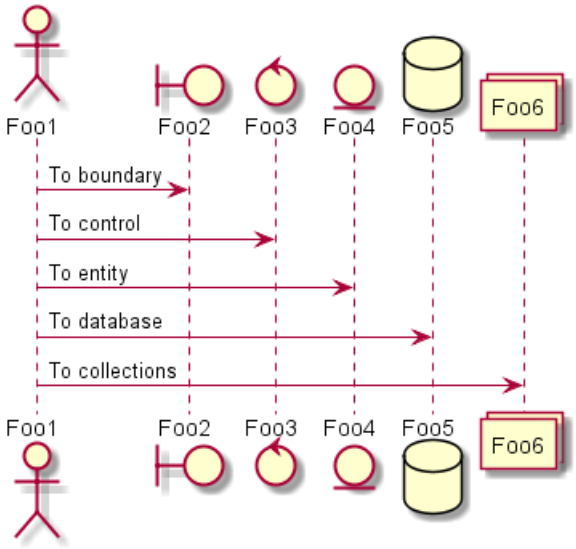
1.2 Declaring participant

It is possible to change participant order using the `participant` keyword.

It is also possible to use other keywords to declare a participant:

- `actor`
- `boundary`
- `control`
- `entity`
- `database`

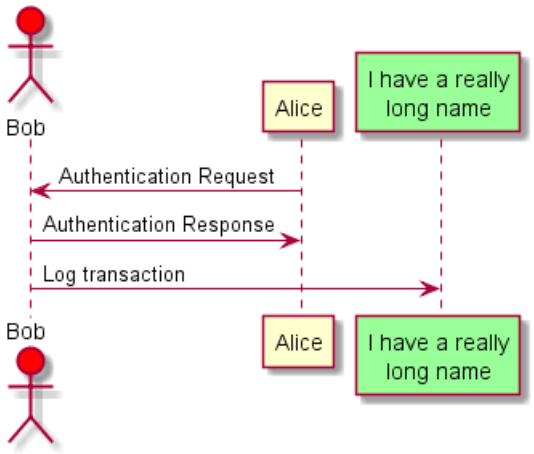
```
@startuml
actor Foo1
boundary Foo2
control Foo3
entity Foo4
database Foo5
collections Foo6
Foo1 -> Foo2 : To boundary
Foo1 -> Foo3 : To control
Foo1 -> Foo4 : To entity
Foo1 -> Foo5 : To database
Foo1 -> Foo6 : To collections

@enduml
```



You can rename a participant using the `as` keyword.

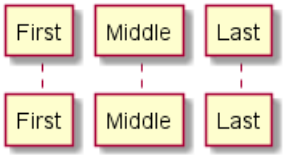
You can also change the background color of actor or participant.

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

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



You can use the `order` keyword to custom the print order of participant.

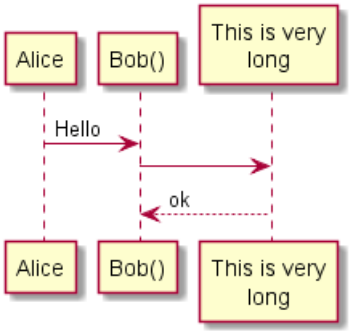
```
@startuml
participant Last order 30
participant Middle order 20
participant First order 10
@enduml
```



1.3 Use non-letters in participants

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

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

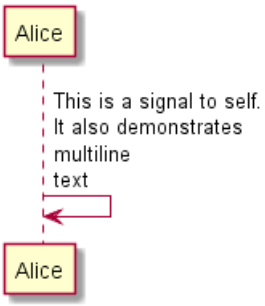


1.4 Message to Self

A participant can send a message to itself.

It is also possible to have multi-line using `\n`.

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



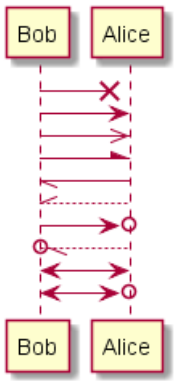
1.5 Change arrow style

You can change arrow style by several ways:

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

- use -- instead of - to have a dotted arrow
- add a final "o" at arrow head
- use bidirectional arrow <->

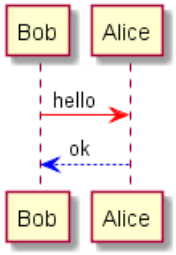
```
@startuml
Bob ->x Alice
Bob -> Alice
Bob ->> Alice
Bob -\ Alice
Bob \\- Alice
Bob //-- Alice

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

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



1.6 Change arrow color

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

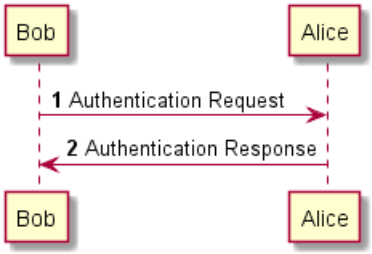
```
@startuml
Bob -[#red]> Alice : hello
Alice -[#0000FF]->Bob : ok
@enduml
```



1.7 Message sequence numbering

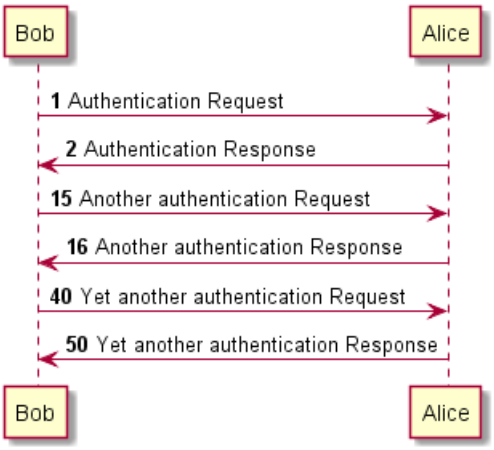
The keyword `autonumber` is used to automatically add number to messages.

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



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

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

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

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

@enduml
```

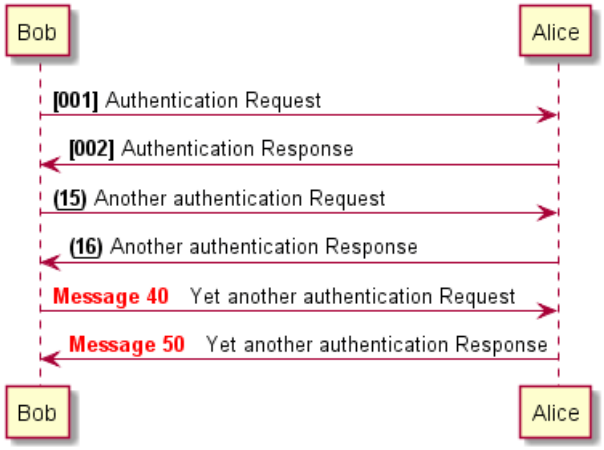


You can specify a format for your number by using between double-quote.

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

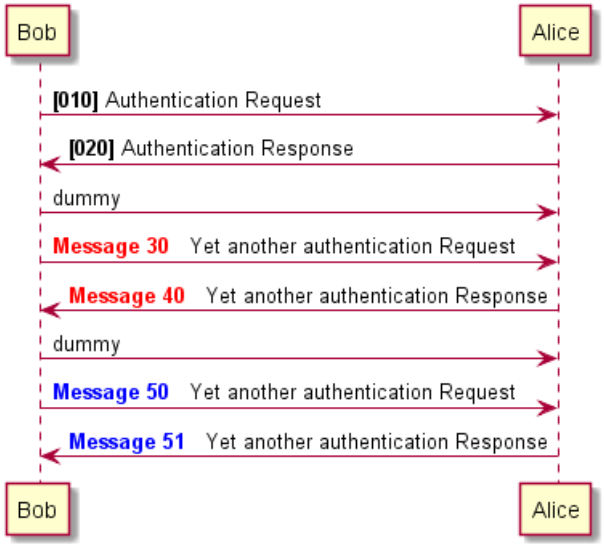
You can use some html tag in the format.

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

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

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

@enduml
```



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

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

autonumber stop
Bob -> Alice : dummy

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

autonumber stop
Bob -> Alice : dummy

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



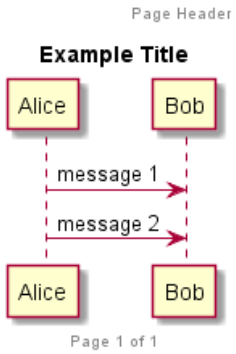
1.8 Page Title, Header and Footer

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

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

```
@startuml
```

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

title Example Title

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

@enduml
```



1.9 Splitting diagrams

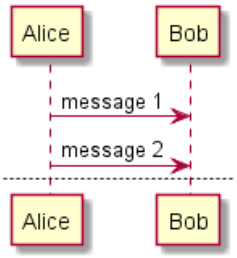
The `newpage` keyword is used to split a diagram into several images.

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

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

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

```
@startuml

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

newpage

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

newpage A title for the\nlast page

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



1.10 Grouping message

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

- `alt/else`

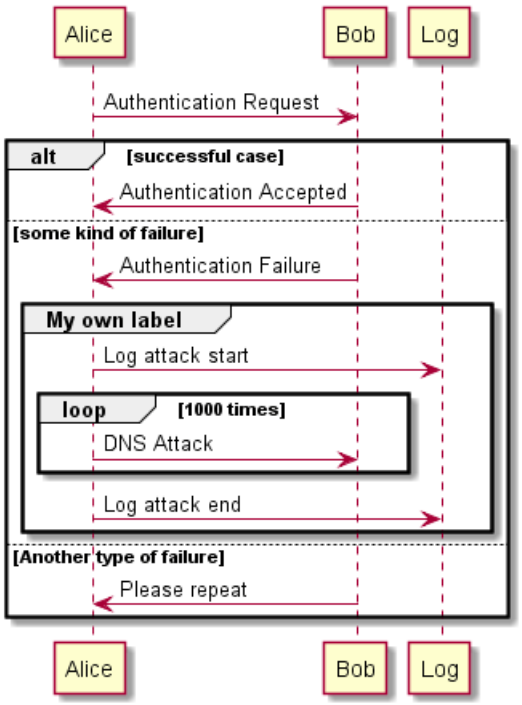
- opt
- loop
- par
- break
- critical
- group, followed by a text to be displayed

It is possible a add a text that will be displayed into the header (except for group).

The end keyword is used to close the group.

Note that it is possible to nest groups.

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

alt successful case

        Bob -> Alice: Authentication Accepted

else some kind of failure

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

else Another type of failure

   Bob -> Alice: Please repeat

end
@enduml
```

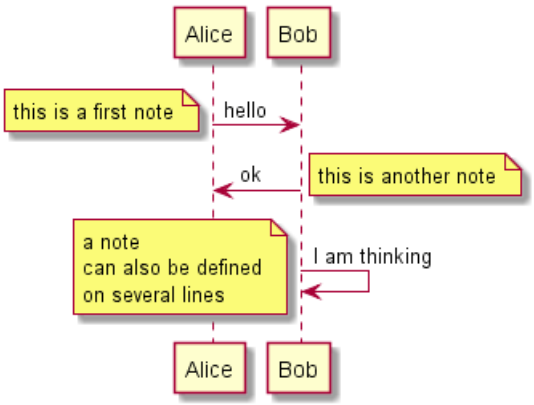


1.11 Notes on messages

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

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

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

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

Bob->Bob : I am thinking
note left
        a note
        can also be defined
        on several lines
end note
@enduml
```



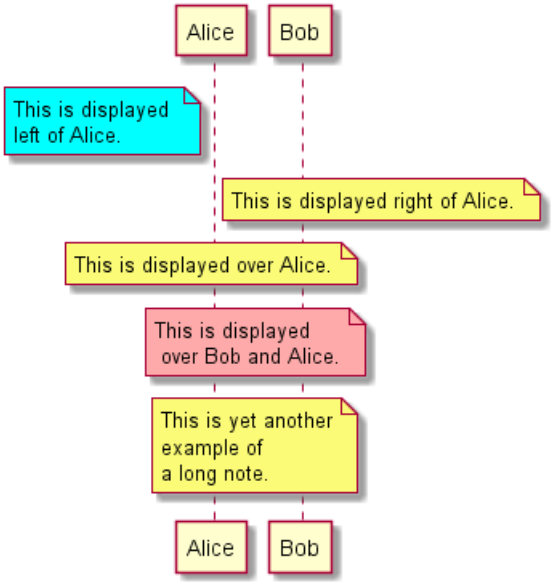
1.12 Some other notes

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

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

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

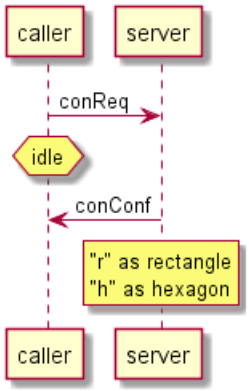
```
@startuml
participant Alice
participant Bob
note left of Alice #aqua
        This is displayed
        left of Alice.
end note

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

note over Alice: This is displayed over Alice.

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

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



1.13 Changing notes shape

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

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

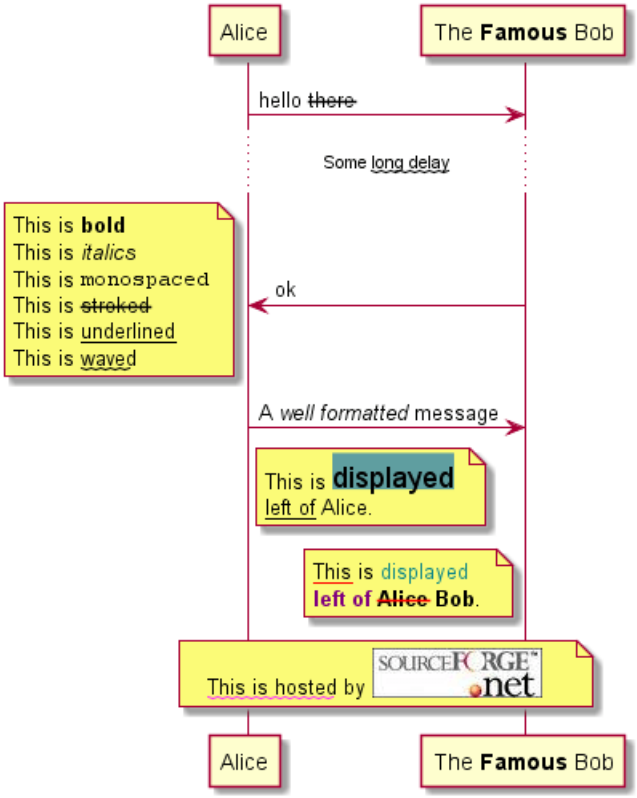


1.14 Creole and HTML

It is also possible to use creole formatting:

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

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

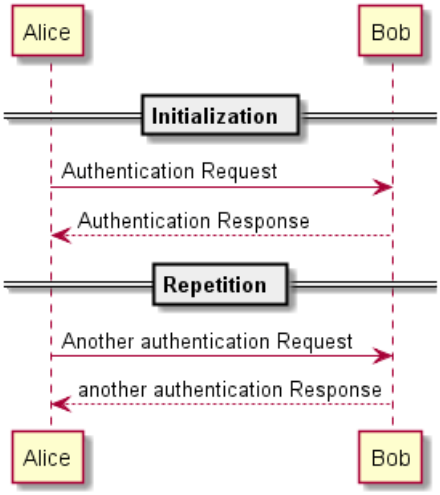
```
  This is --stroked--
  This is __underlined__
  This is ~~waved~~
end note

Alice -> Bob : A //well formatted// message
note right of Alice
 This is <back:cadetblue><size:18>displayed</size></back>
 __left of__ Alice.
end note
note left of Bob
 <u:red>This</u> is <color #118888>displayed</color>
 **<color purple>left of</color> <s:red>Alice</strike> Bob**.
end note
note over Alice, Bob
 <w:#FF33FF>This is hosted</w> by <img sourceforge.jpg>
end note
@enduml
```



1.15 Divider

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

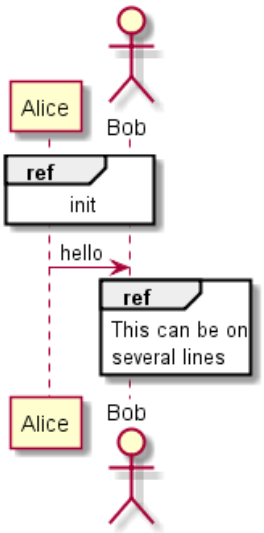
```
@startuml

== Initialization ==

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

== Repetition ==

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

@enduml
```



1.16 Reference

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

```
@startuml
participant Alice
actor Bob

ref over Alice, Bob : init

Alice -> Bob : hello

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

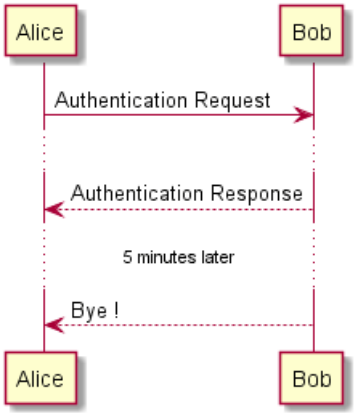


1.17 Delay

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

```
@startuml

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

```
...5 minutes later...
Bob --> Alice: Bye !

@enduml
```

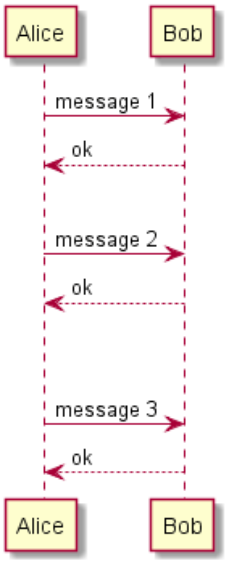


1.18 Space

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

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

```
@startuml

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

@enduml
```



1.19 Lifeline Activation and Destruction

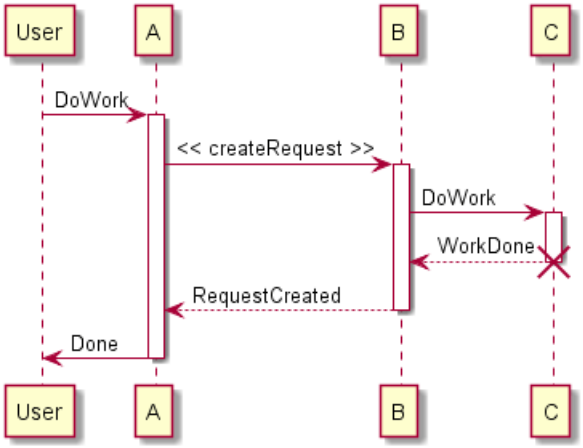
The `activate` and `deactivate` are used to denote participant activation.

Once a participant is activated, its lifeline appears.

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

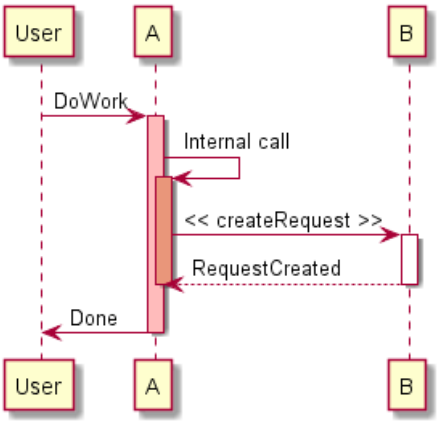
The `destroy` denote the end of the lifeline of a participant.

```
@startuml
participant User

User -> A: DoWork
activate A

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

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

B --> A: RequestCreated
deactivate B

A -> User: Done
deactivate A

@enduml
```



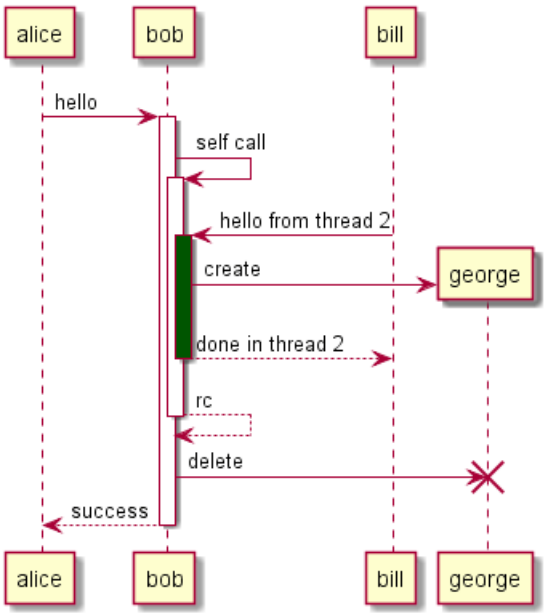
Nested lifeline can be used, and it is possible to add a color on the lifeline.

```
@startuml
participant User

User -> A: DoWork
activate A #FFBBBB

A -> A: Internal call
activate A #DarkSalmon

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

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

@enduml
```



Autoactivation is possible and works with the return keywords:

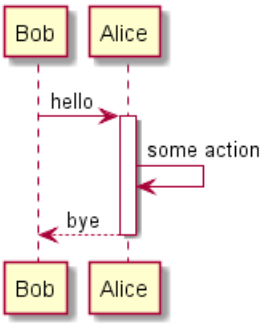
```
@startuml
autoactivate on
alice -> bob : hello
bob -> bob : self call
bill -> bob #005500 : hello from thread 2
bob -> george ** : create
return done in thread 2
return rc
bob -> george !! : delete
return success
@enduml
```



1.20 Return

A new command `return` for generating a return message with optional text label. The point returned to is the point that cause the most recently activated life-line. The syntax is simply return label where label, if provided, can be any string acceptable on conventional messages.

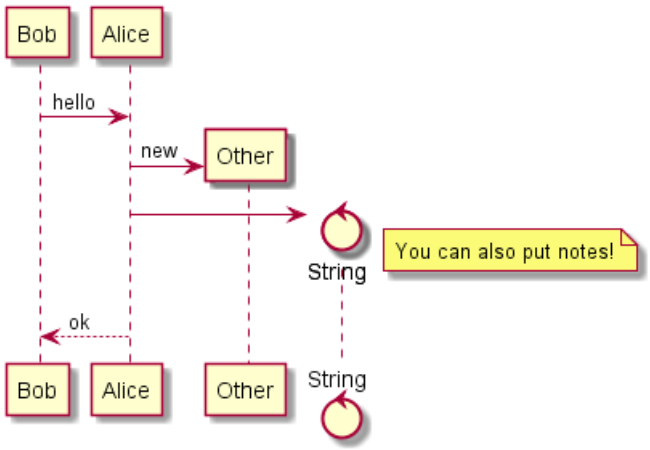
```
@startuml
Bob -> Alice : hello
activate Alice
Alice -> Alice : some action
return bye
@enduml
```



1.21 Participant creation

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

```
@startuml
Bob -> Alice : hello

create Other
Alice -> Other : new

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

Alice --> Bob : ok

@enduml
```



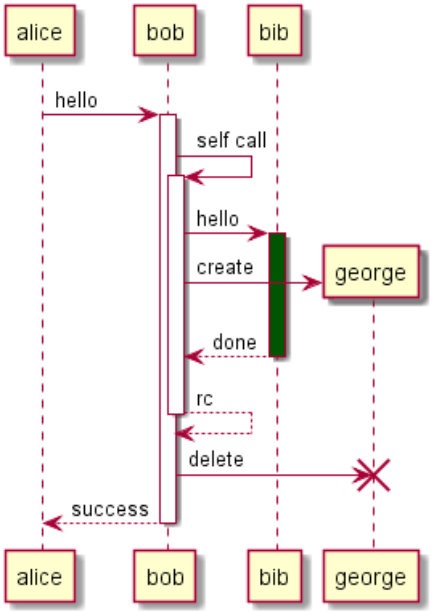
1.22 Shortcut syntax for activation, deactivation, creation

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

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

```
@startuml
alice -> bob ++ : hello
bob -> bob ++ : self call
bob -> bib ++  #005500 : hello
bob -> george ** : create
return done
```

```
return rc
bob -> george !! : delete
return success
@enduml
```

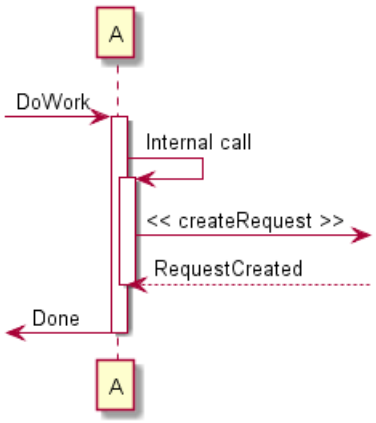


1.23 Incoming and outgoing messages

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:

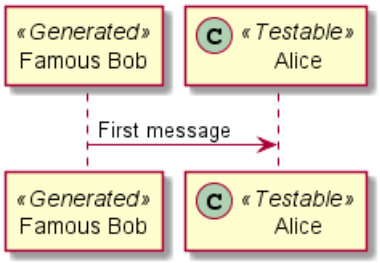
```
@startuml
[-> Bob
[o-> Bob
[o->o Bob
[x-> Bob

[<- Bob
[x<- Bob

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

Bob <-]
Bob x<-]
@enduml
```

1.24 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)`.

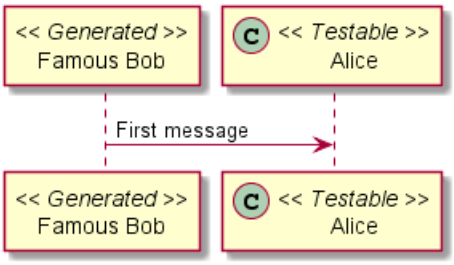
```
@startuml

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

Bob->Alice: First message

@enduml
```



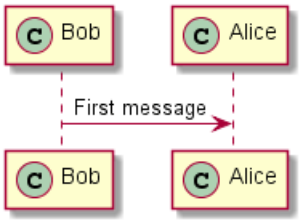
By default, the *guillemet* character is used to display the stereotype. You can change this behavious using the skinparam `guillemet`:

```
@startuml

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

```
Bob->Alice: First message

@enduml
```



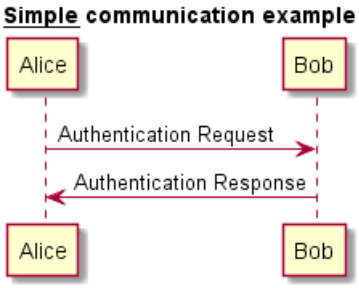

```
@startuml

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

Bob->Alice: First message

@enduml
```



1.25 More information on titles

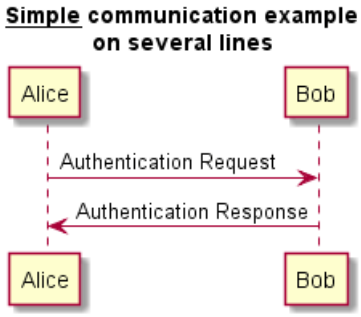
You can use creole formatting in the title.

```
@startuml

title __Simple__ **communication** example

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

@enduml
```



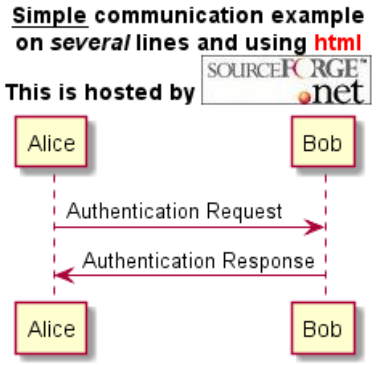
You can add newline using \n in the title description.

```
@startuml

title __Simple__ communication example\non 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
```

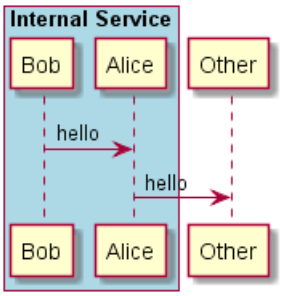


1.26 Participants encompass

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.

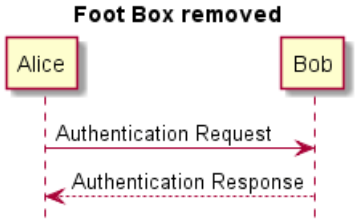
```
@startuml

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

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

@enduml
```



1.27 Removing Foot Boxes

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

```
@startuml

hide footbox
title Foot Box removed

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

@enduml
```



1.28 Skinparam

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

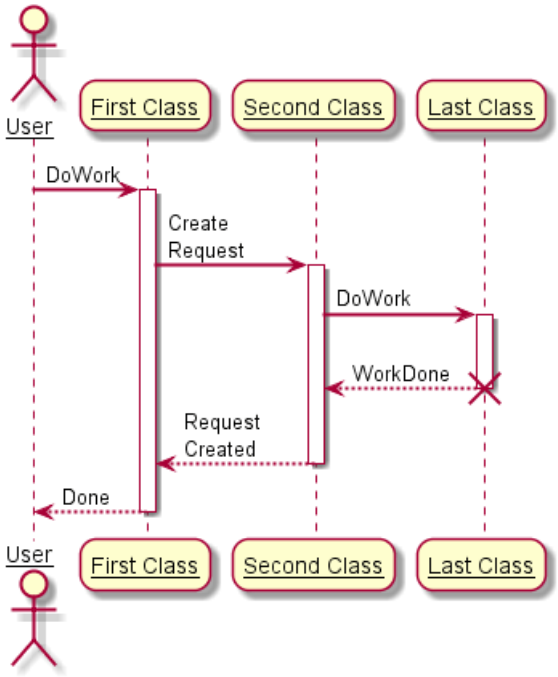
You can use this command:

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

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

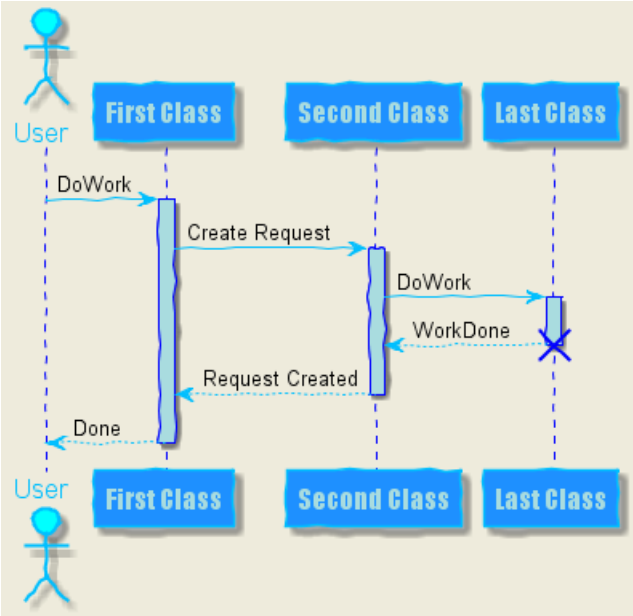
```
@startuml
skinparam sequenceArrowThickness 2
skinparam roundcorner 20
skinparam maxmessagesize 60
skinparam sequenceParticipant underline

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

User -> A: DoWork
activate A

A -> B: Create Request
activate B

B -> C: DoWork
activate C
```

```
C --> B: WorkDone
destroy C

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml
```




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

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

        ParticipantBorderColor DeepSkyBlue
        ParticipantBackgroundColor DodgerBlue
        ParticipantFontName Impact
        ParticipantFontSize 17
        ParticipantFontColor #A9DCDF

        ActorBackgroundColor aqua
        ActorFontColor DeepSkyBlue
        ActorFontSize 17
        ActorFontName Aapex
}

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

User -> A: DoWork
activate A

A -> B: Create Request
activate B

B -> C: DoWork
activate C
```

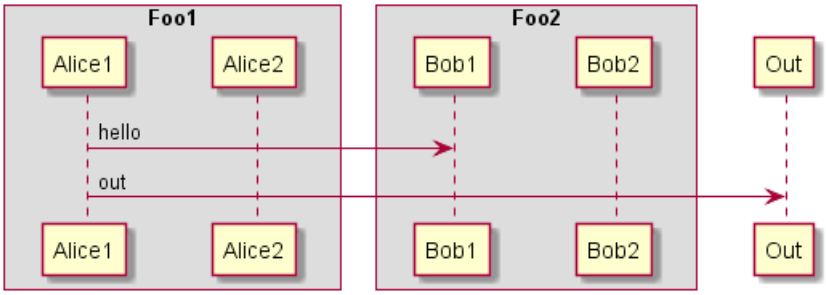
```
C --> B: WorkDone
destroy C

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml
```



1.29 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 Use Case Diagram

Let's have few examples :

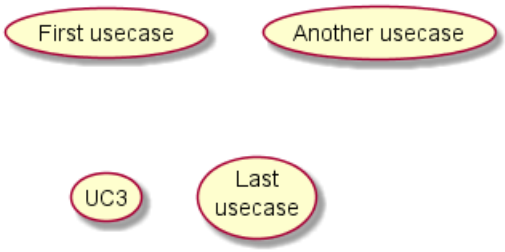
Note that you can disable the shadowing using the `skinparam shadowing false` command.

2.1 Usecases

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

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

```
@startuml

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

@enduml
```



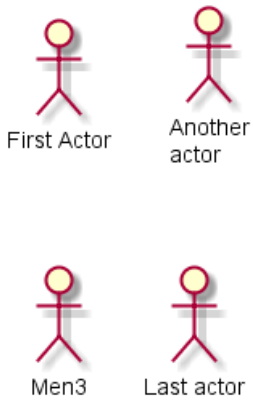
2.2 Actors

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
```

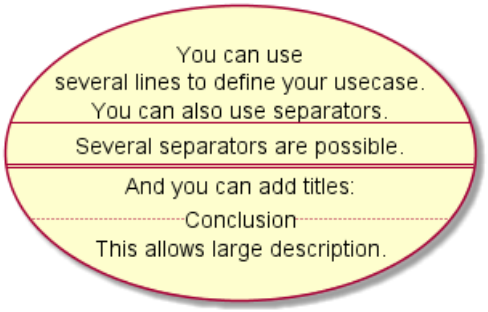


2.3 Usecases description

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
```



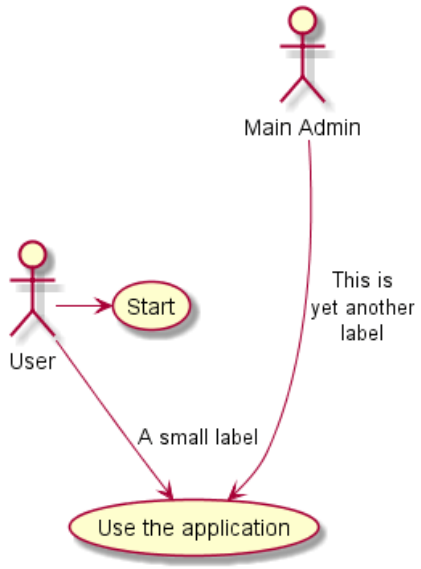
2.4 Basic example

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

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

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

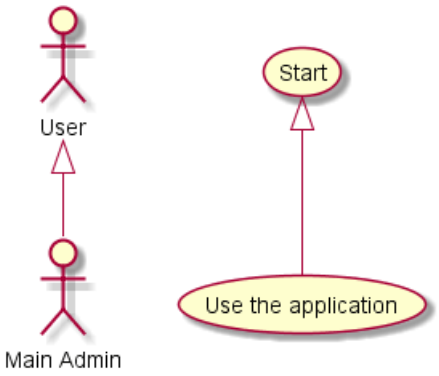
```
@startuml

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

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

@enduml
```



2.5 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
```



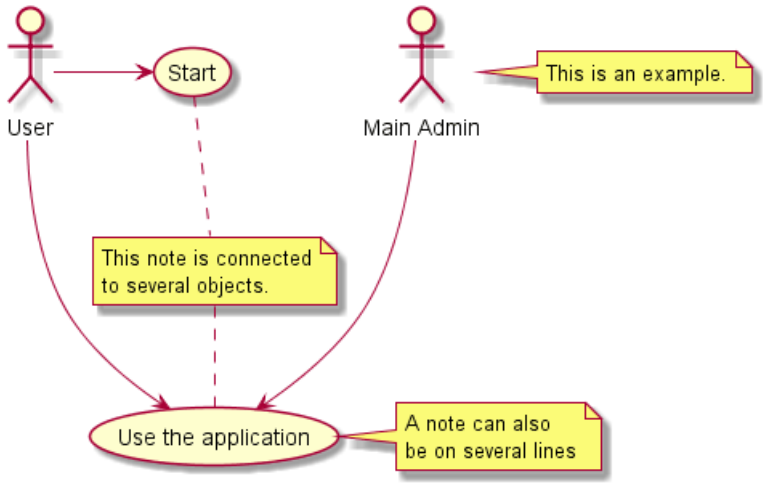
2.6 Using notes

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)
```

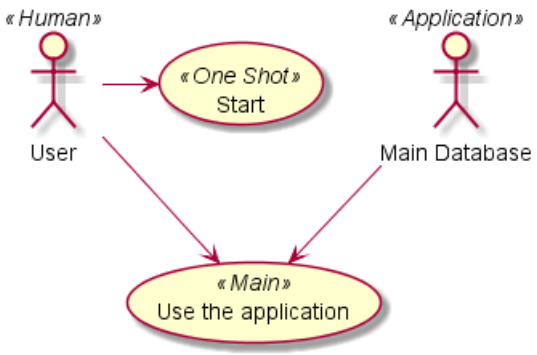
```
note right of Admin : This is an example.

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 >>.

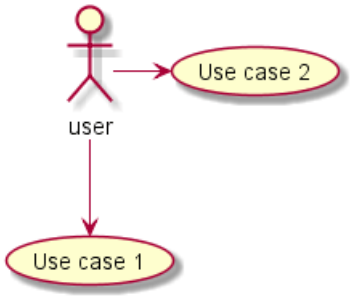
```
@startuml
User << Human >>
:Main Database: as MySql << Application >>
(Start) << One Shot >>
(Use the application) as (Use) << Main >>

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

MySql --> (Use)

@enduml
```



2.8 Changing arrows direction

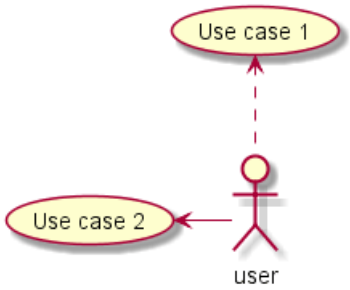
By default, links between classes have two dashes -- and are vertically oriented. It is possible to use horizontal link by putting a single dash (or dot) like this:

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



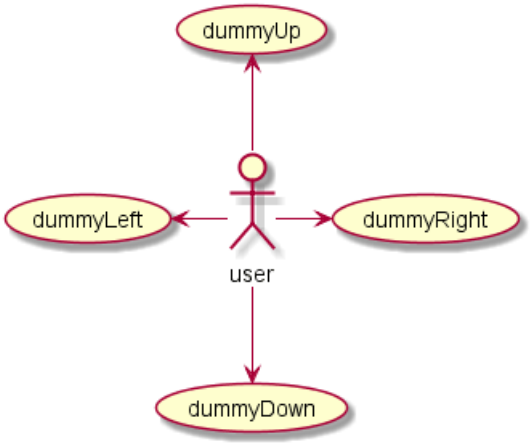
You can also change directions by reversing the link:

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



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

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



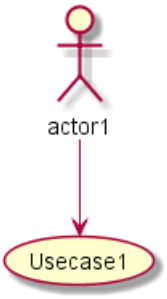
You can shorten the arrow by using only the first character of the direction (for example, `-d-` instead of `-down-`) or the two first characters (`-do-`).

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

2.9 Splitting diagrams

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

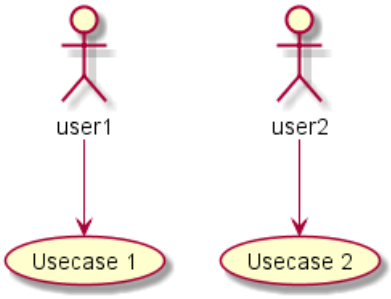
```
@startuml
:actor1: --> (Usecase1)
newpage
:actor2: --> (Usecase2)
@enduml
```



2.10 Left to right direction

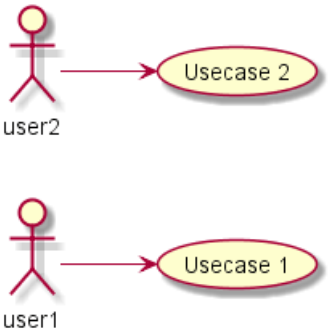
The general default behavior when building diagram is **top to bottom**.

```
@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

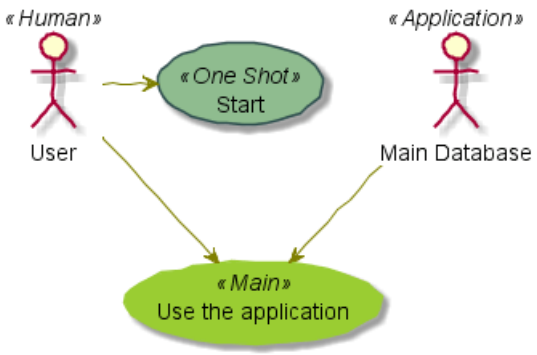
You can use the `skinparam` command to change colors and fonts for the drawing.

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.

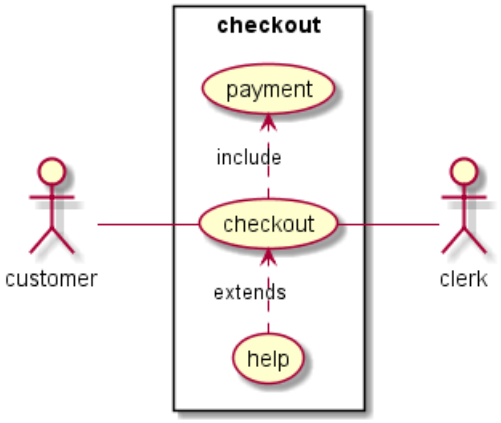
You can define specific color and fonts for stereotyped actors and usecases.

```
@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 Class Diagram

3.1 Relations between classes

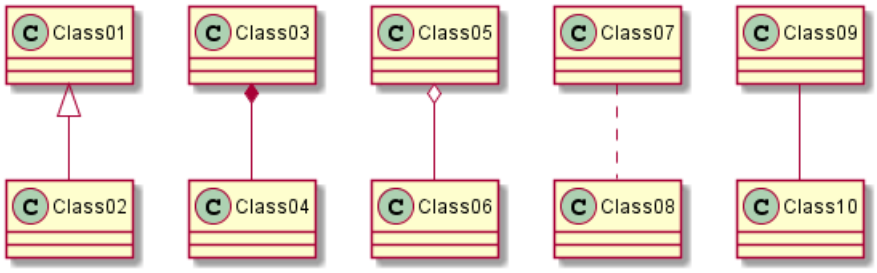
Relations between classes are defined using the following symbols :

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

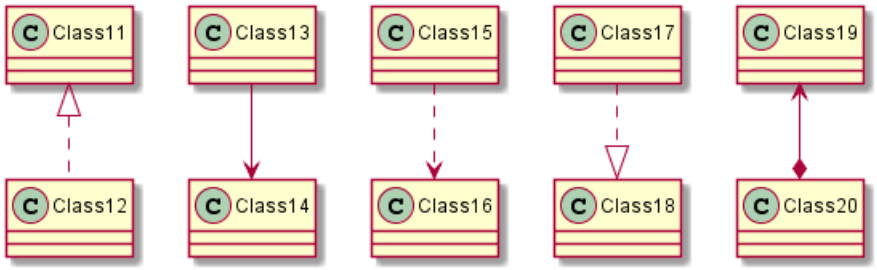
It is possible to replace `--` by `..` to have a dotted line.

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

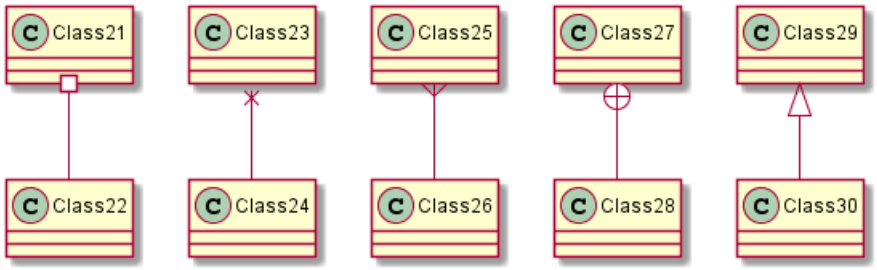
```
@startuml
Class01 <|-- Class02
Class03 *-- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10
@enduml
```




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




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

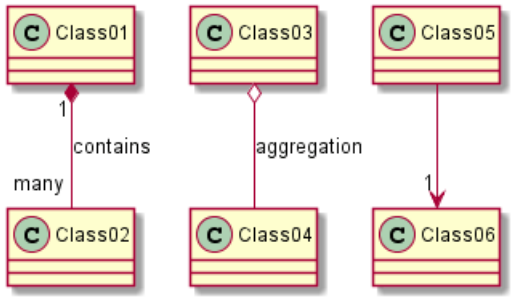


3.2 Label on relations

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

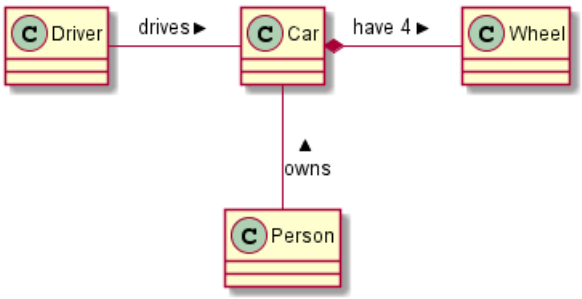
For cardinality, you can use double-quotes "" on each side of the relation.

```
@startuml

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

Class03 o-- Class04 : aggregation

Class05 --> "1" Class06

@enduml
```



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

```
@startuml
class Car

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

@enduml
```

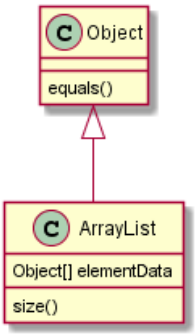


3.3 Adding methods

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.

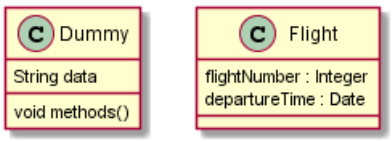
```
@startuml
Object <|-- ArrayList

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

@enduml
```



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

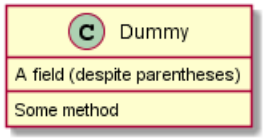
Note that the syntax is highly flexible about type/name order.

```
@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
```

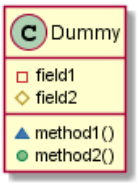


3.4 Defining visibility

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

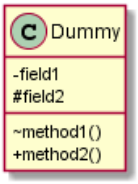
Character	Icon for field	Icon for method	Visibility
-	□	■	private
#	◇	◆	protected
~	△	▲	package private
+	○	●	public

```
@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
```

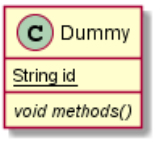


3.5 Abstract and Static

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
```

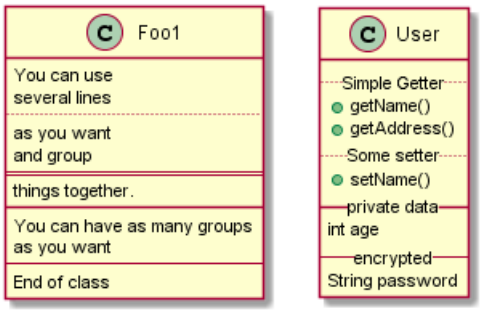


3.6 Advanced class body

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 >>.

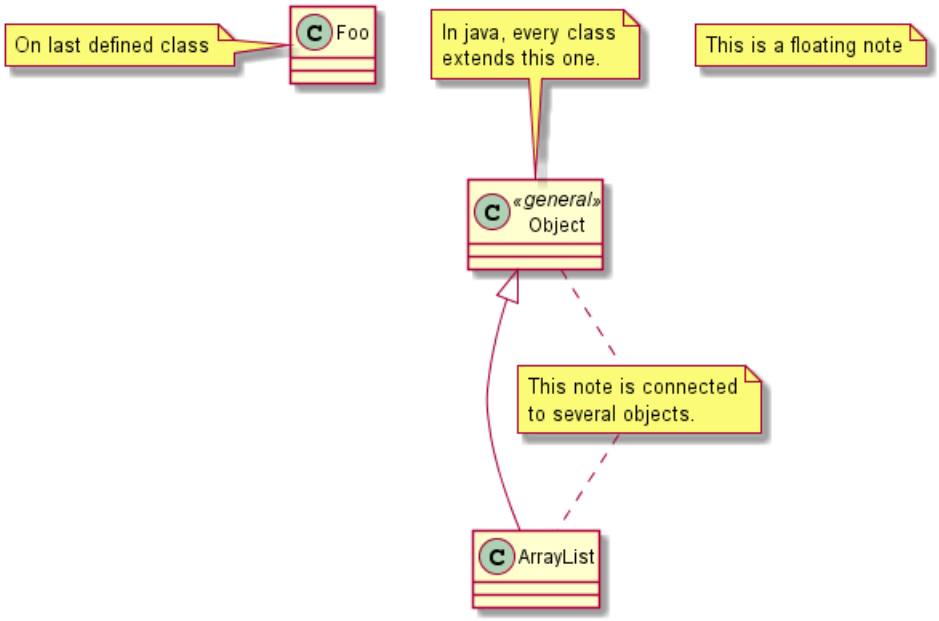
You can also define notes using `note left of` , `note right of` , `note top of` , `note bottom of` keywords.

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

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

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

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

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

```
class Foo
note left: On last defined class

@enduml
```



3.8 More on notes

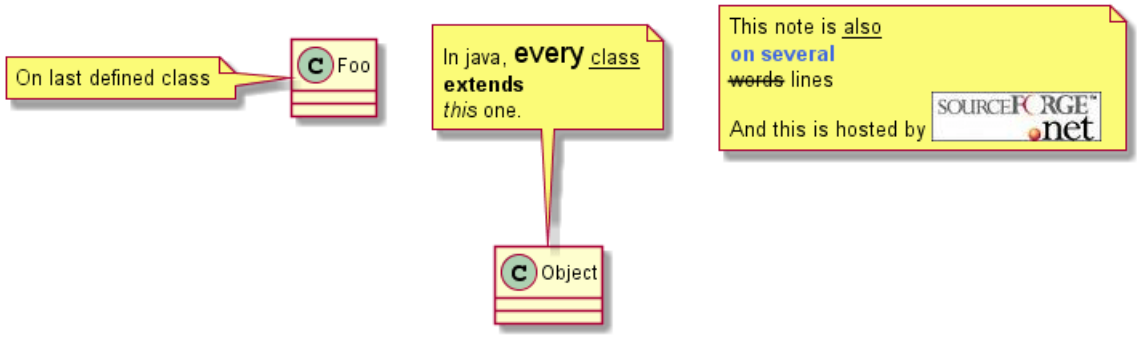
It is also possible to use few html tags like :

- `<b>`
- `<u>`
- `<i>`
- `<s>`, `<del>`, `<strike>`
- `<font color="#AAAAAA">` or `<font color="colorName">`
- `<color:#AAAAAA>` or `<color:colorName>`
- `<size:nn>` to change font size
- `<img src="file">` 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
```

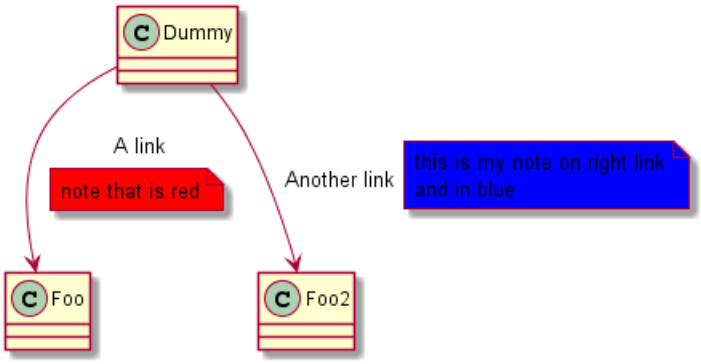


3.9 Note on links

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

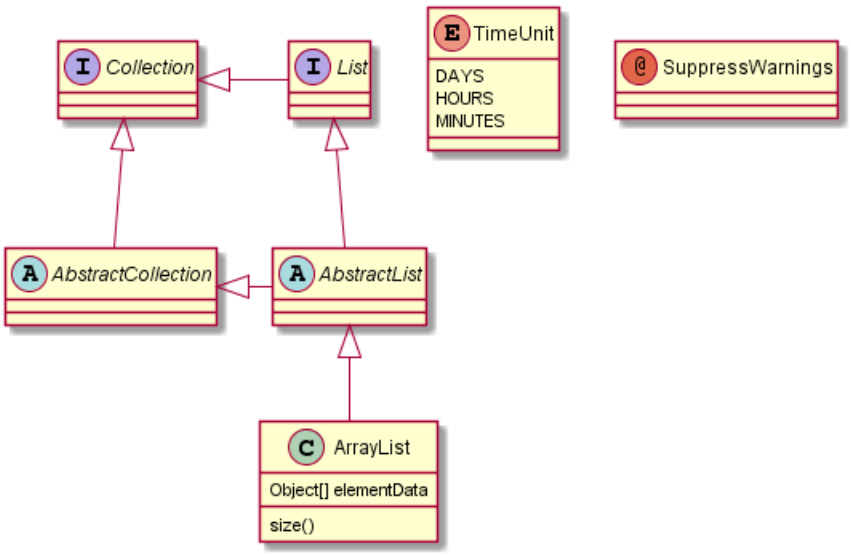
You can declare a class as abstract using `abstract"` or `abstract class` keywords.

The class will be printed in *italic*.

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

```
@startuml

abstract class AbstractList
abstract AbstractCollection
interface List
interface Collection

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

Collection <|- List
```

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

class ArrayList {
  Object[] elementData
  size()
}

enum TimeUnit {
  DAYS
  HOURS
  MINUTES
}

annotation SuppressWarnings

@enduml
```

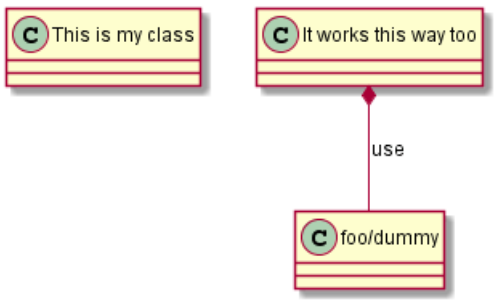


3.11 Using non-letters

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:

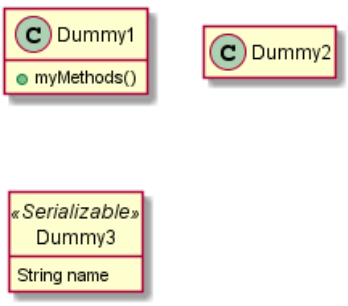
- `empty fields` or `empty attributes` for empty fields,
- `empty methods` for empty methods,
- `fields` or `attributes` which will hide fields, even if they are described,
- `methods` which will hide methods, even if they are described,
- `members` which will hide fields and methods, even if they are 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.

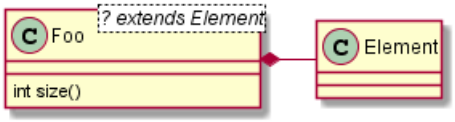
This may be useful if you define a large !included file, and if you want to hide come classes after file inclusion.

```
@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
```



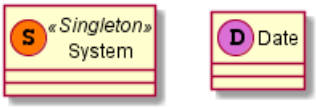
It is possible to disable this drawing using `skinparam genericDisplay old` command.

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
```

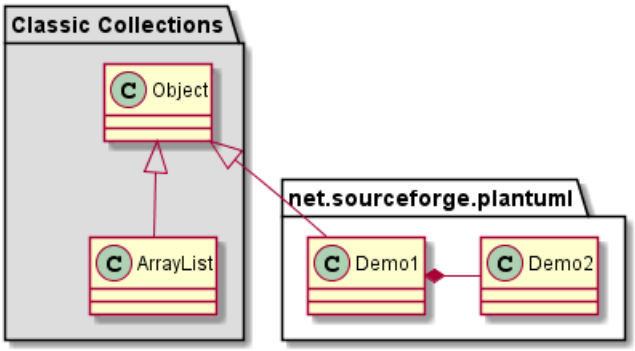


3.16 Packages

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
}

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

@enduml
```

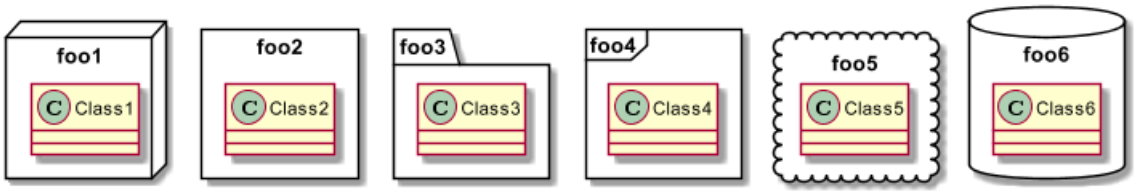


3.17 Packages style

There are different styles available for packages.

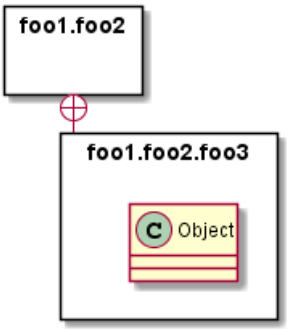
You can specify them either by setting a default style with the command : `skinparam packageStyle`, or by using a stereotype on the package:

```
@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.

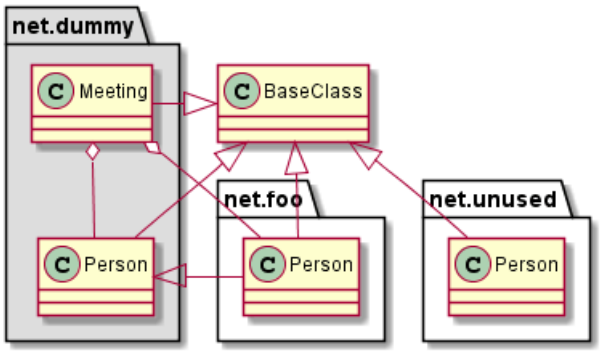
In that case, you should use namespaces instead of packages.

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
}
```

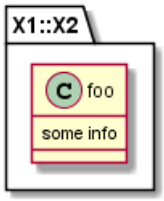
```
BaseClass <|-- net.unused.Person

@enduml
```



3.19 Automatic namespace creation

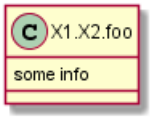
You can define another separator (other than the dot) using the command : `set namespaceSeparator ???`.

```
@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
```

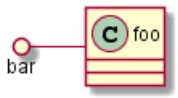


3.20 Lollipop interface

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

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

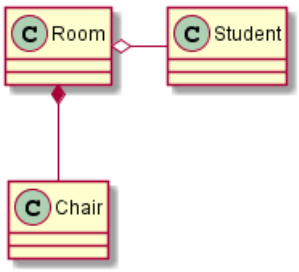
```
@startuml
class foo
bar ()- foo
@enduml
```



3.21 Changing arrows direction

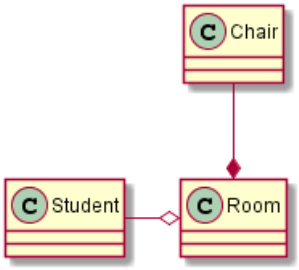
By default, links between classes have two dashes `--` and are vertically oriented. It is possible to use horizontal link by putting a single dash (or dot) like this:

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



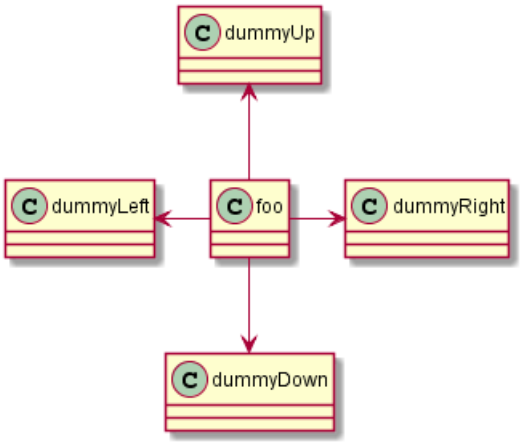
You can also change directions by reversing the link:

```
@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
```



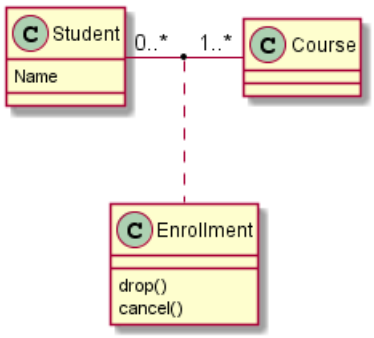
You can shorten the arrow by using only the first character of the direction (for example, `-d-` instead of `-down-`) or the two first characters (`-do-`).

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

3.22 Association classes

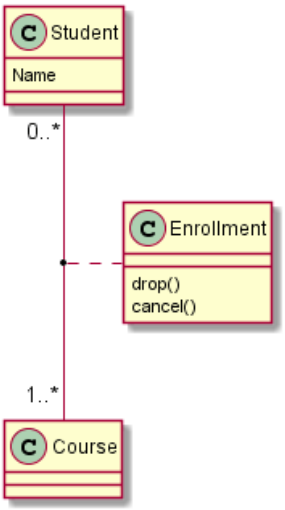
You can define *association class* after that a relation has been defined between two classes, like in this example:

```
@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
```



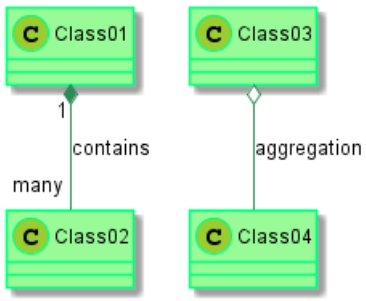
3.23 Skinparam

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
```

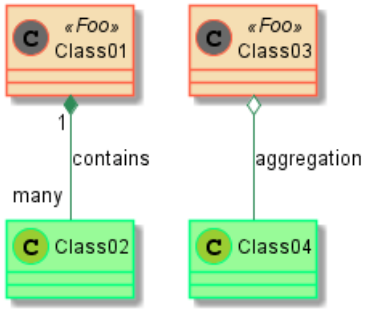


3.24 Skinned Stereotypes

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.

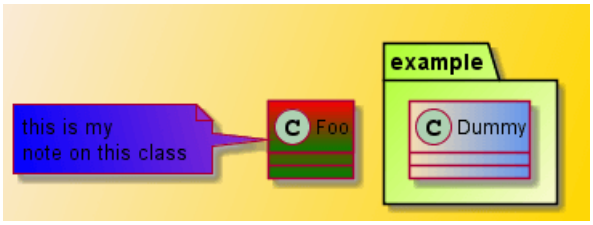
You can also use color gradient in background, with the following syntax: two colors names separated either by:

- |,
- /,
- \,
- 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
```



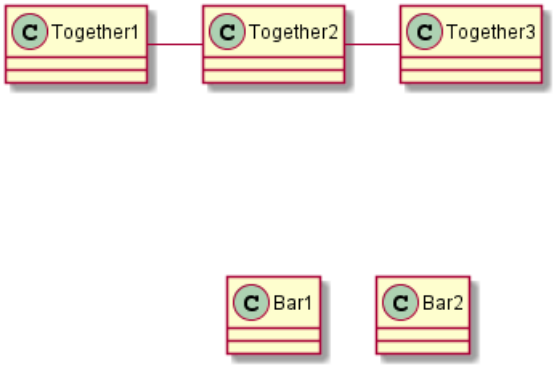
3.26 Help on layout

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 Splitting large files

Sometimes, you will get some very large image files.

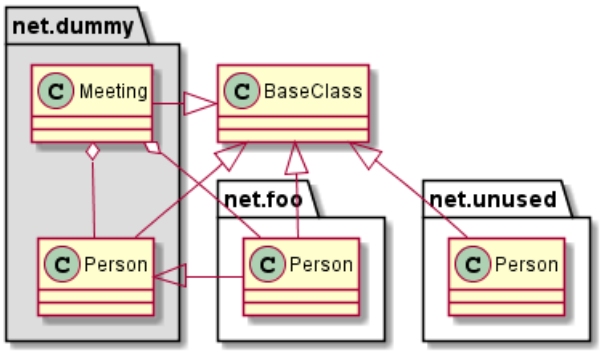
You can use the `page (hpages)x(vpages)` command to split the generated image into several files :

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

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

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

```
class BaseClass

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

        .BaseClass <|- Meeting

}

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

  net.dummy.Meeting o-- Person
}

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



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.

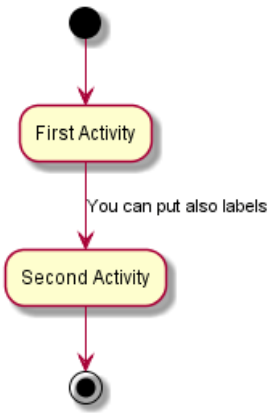
```
@startuml

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

@enduml
```

4.2 Label on arrows

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

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

```
@startuml

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

@enduml
```



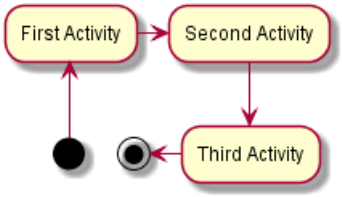
4.3 Changing arrow direction

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->

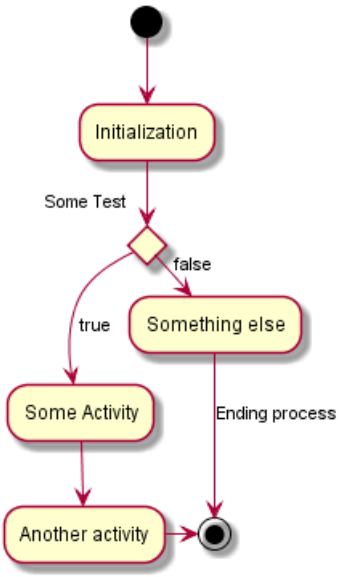
```
@startuml

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

@enduml
```



4.4 Branches

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

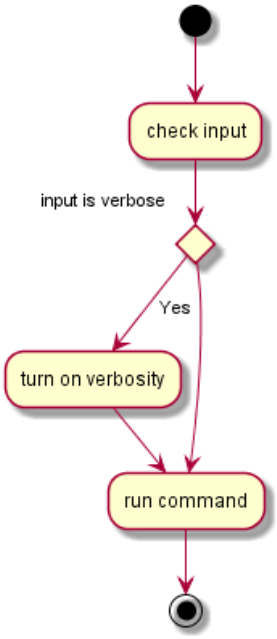
```
@startuml
(*) --> "Initialization"

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
```

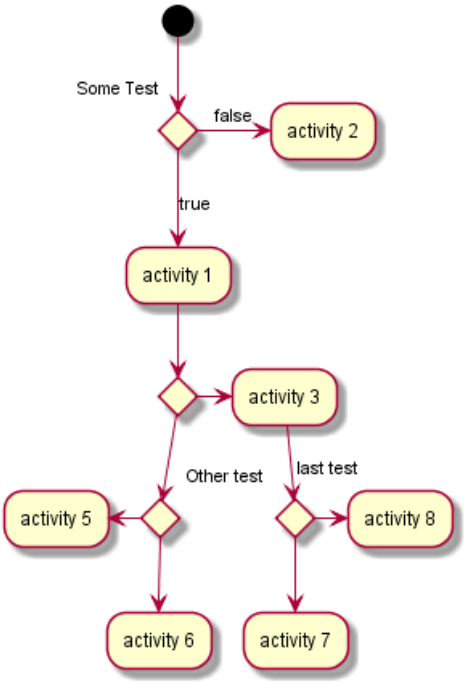


4.5 More on Branches

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

It is also possible to nest branches.

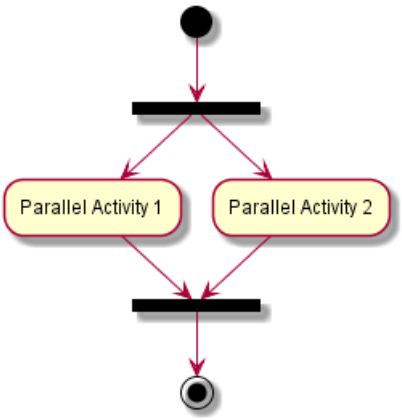
```
@startuml

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

  -->[true] "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
```

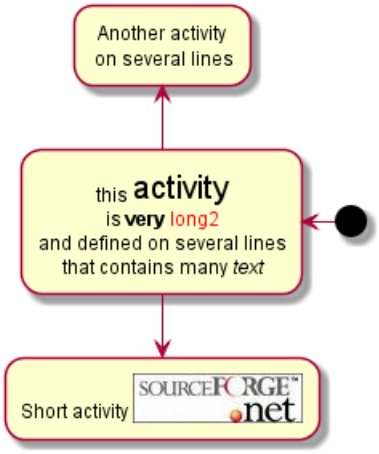


4.7 Long activity description

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
```



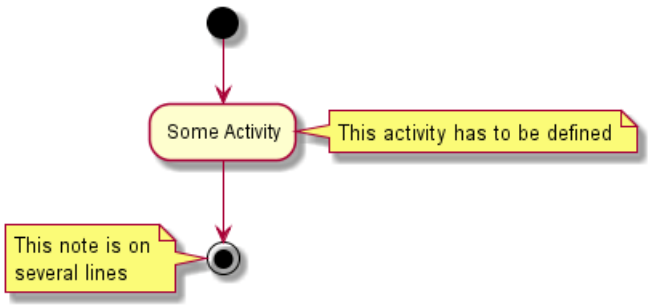
4.8 Notes

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
```



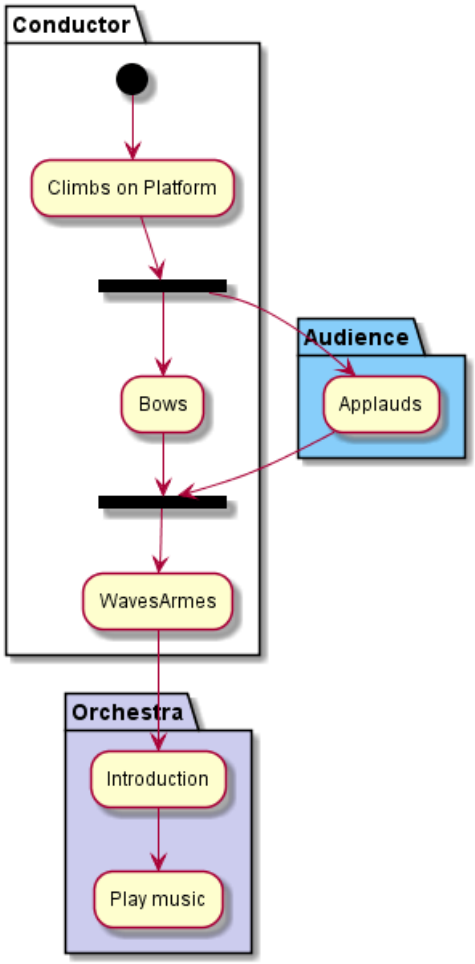
4.9 Partition

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

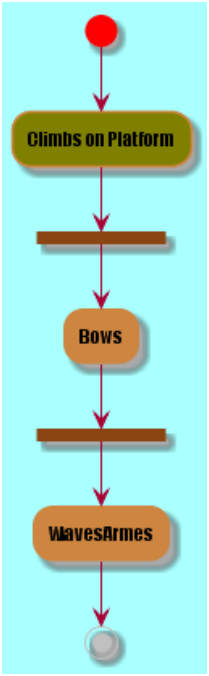
You can use the `skinparam` command to change colors and fonts for the drawing.

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

(*) --> "Climbs on Platform" << Begin >>
--> === S1 ===
--> Bows
--> === S2 ===
--> WavesArmes
--> (*)

@enduml
```



4.11 Octagon

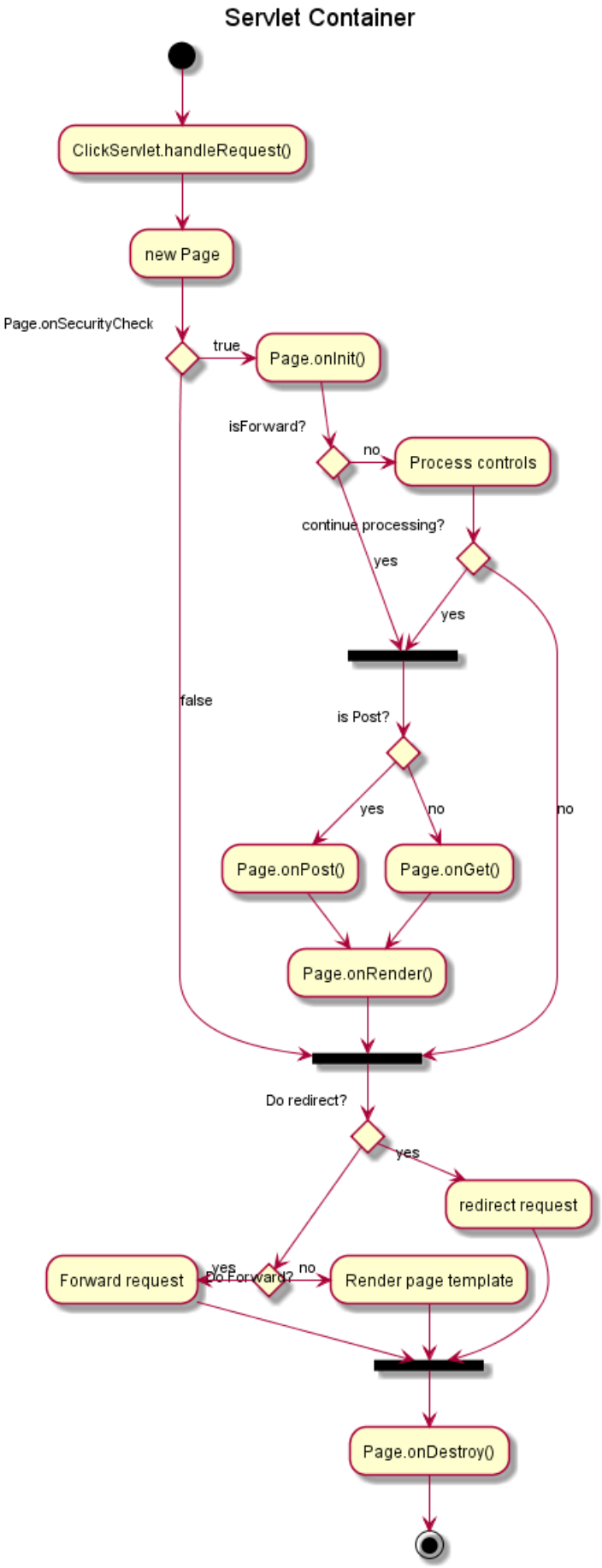
You can change the shape of activities to octagon using the `skinparam activityShape octagon` command.

```
@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
```


Servlet Container
ClickServlet.handleRequest()
new Page
Page.onSecurityCheck
true
Page.onInit()
isForward?
no
Process controls
continue processing?
yes
yes
false
is Post?
yes
no
no
Page.onPost()
Page.onGet()
Page.onRender()
Do redirect?
yes
redirect request
Forward request
yes
Do Forward?
no
Render page template
Page.onDestroy()

5 Activity Diagram (beta)

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

So a completely new syntax and implementation is proposed as **beta version** to users (starting with V7947), so that we could define a better format and syntax.

Another advantage of this new implementation is that it's done without the need of having Graphviz installed (as for sequence diagrams).

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

Users are simply encouraged to migrate to the new syntax.

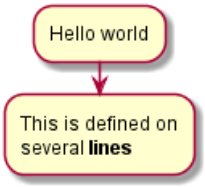
5.1 Simple Activity

Activities label starts with `:` and ends with `;`.

Text formatting can be done using creole wiki syntax.

They are implicitly linked in their definition order.

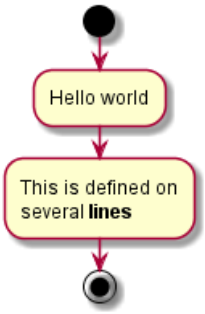
```
@startuml
:Hello world;
:This is defined on
several **lines**;
@enduml
```



5.2 Start/Stop

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

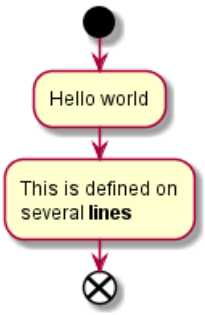
```
@startuml
start
:Hello world;
:This is defined on
several **lines**;
stop
@enduml
```



You can also use the `end` keyword.

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

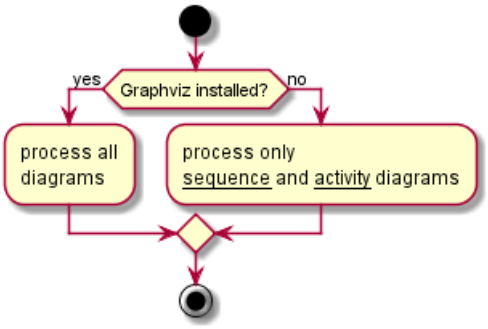
```
several **lines**;
end
@enduml
```



5.3 Conditional

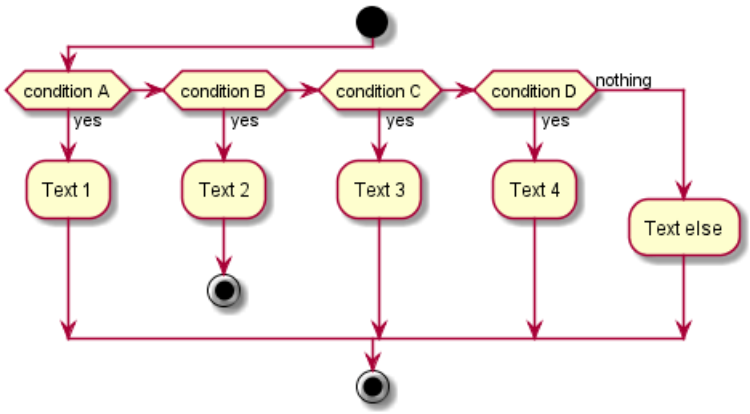
You can use `if`, `then` and `else` keywords to put tests if your diagram. Labels can be provided using parentheses.

```
@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 :

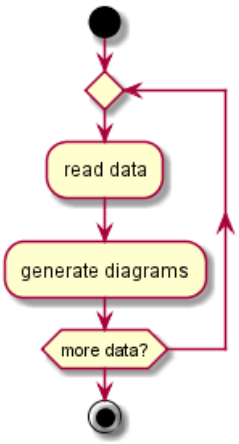
```
@startuml
start
if (condition A) then (yes)
  :Text 1;
elseif (condition B) then (yes)
  :Text 2;
  stop
elseif (condition C) then (yes)
  :Text 3;
elseif (condition D) then (yes)
  :Text 4;
else (nothing)
  :Text else;
endif
stop
@enduml
```



5.4 Repeat loop

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

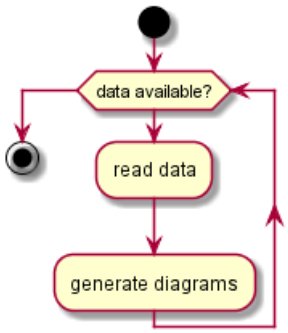
```
@startuml

start

repeat
  :read data;
  :generate diagrams;
repeat while (more data?)

stop

@enduml
```



5.5 While loop

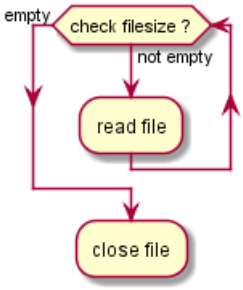
You can use `while` and `end while` keywords to have repeat loops.

```
@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.

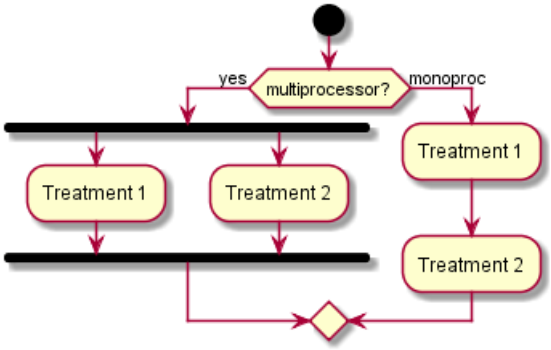
```
@startuml
while (check filesize ?) is (not empty)
  :read file;
endwhile (empty)
:close file;
@enduml
```



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

@enduml
```

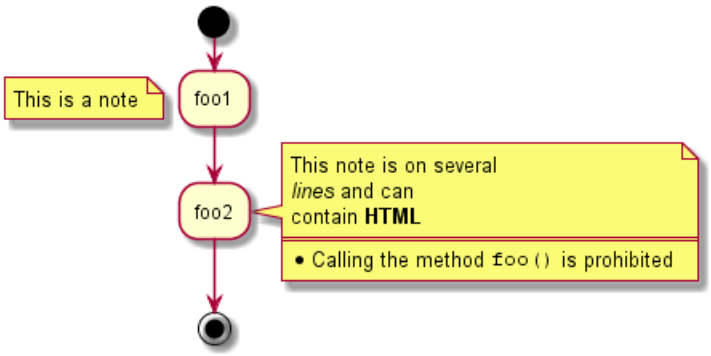


5.7 Notes

Text formatting can be done using creole wiki syntax.

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

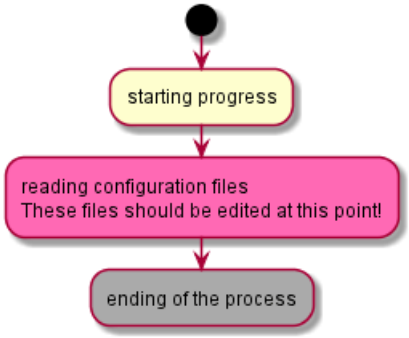
```
@startuml

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
```

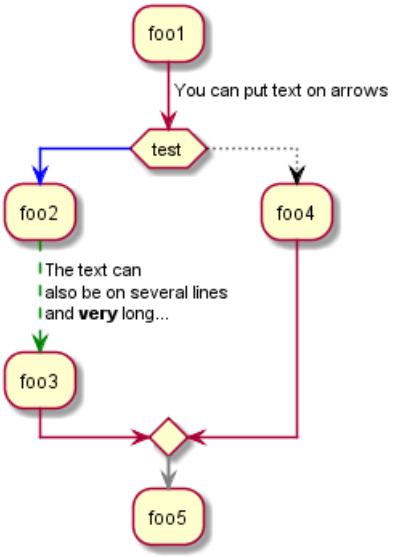


5.9 Arrows

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
```

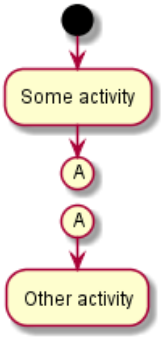


5.10 Connector

You can use parentheses to denote connector.

```
@startuml
start
:Some activity;
(A)
```

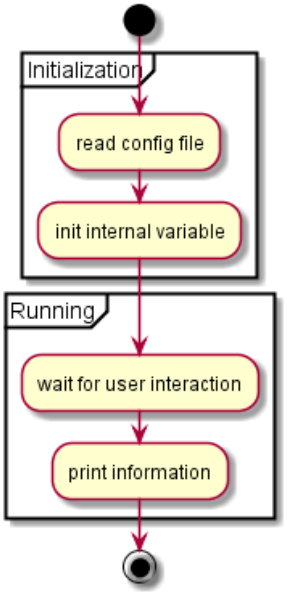
```
detach
(A)
:Other activity;
@enduml
```



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
```

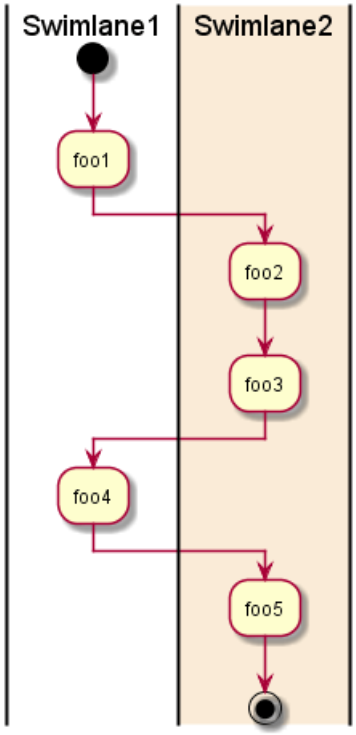


5.12 Swimlanes

Using pipe |, you can define swimlanes.

It's also possible to change swimlanes color.

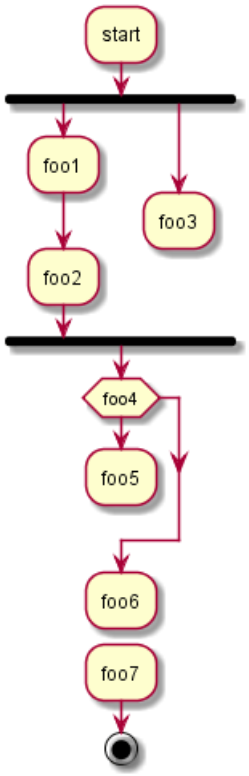
```
@startuml
|Swimlane1|
start
:foo1;
|#AntiqueWhite|Swimlane2|
:foo2;
:foo3;
|Swimlane1|
:foo4;
|Swimlane2|
:foo5;
stop
@enduml
```



5.13 Detach

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

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

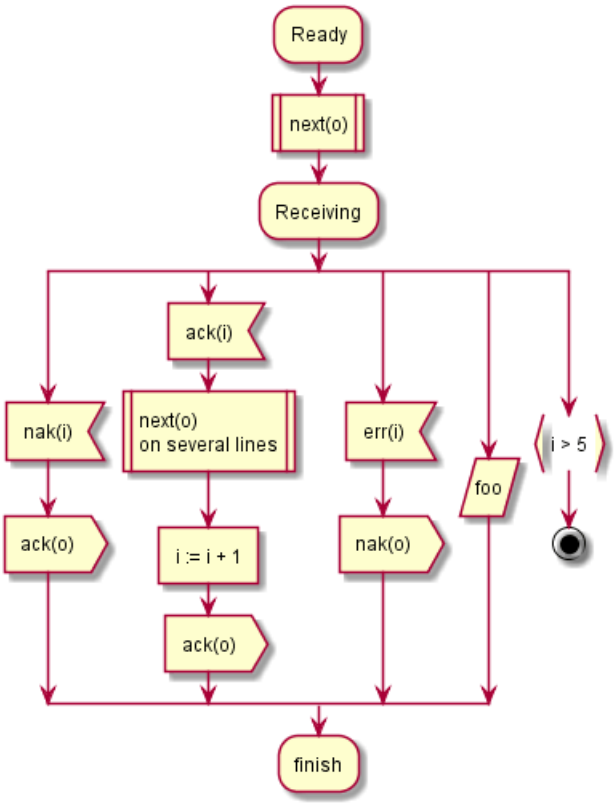


5.14 SDL

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

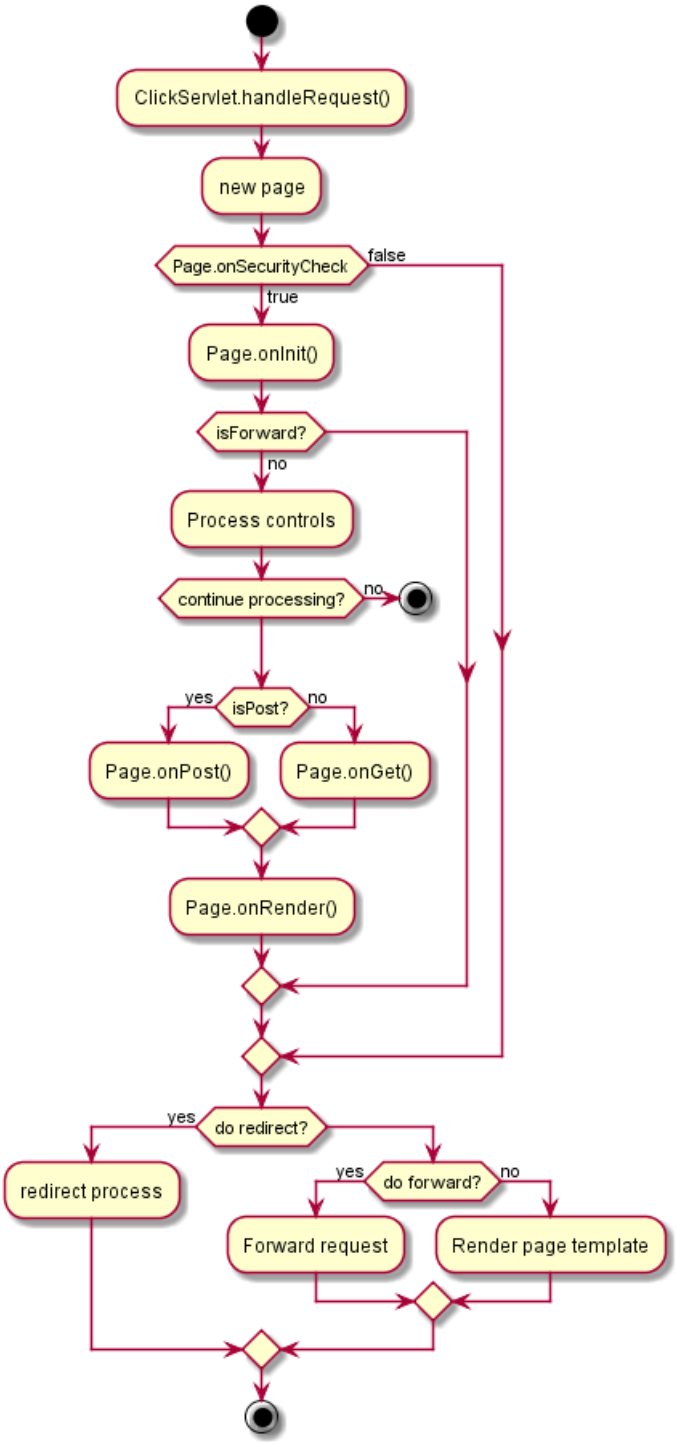
- |
- <
- >
- /
-]
- }

```
@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

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

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

stop

@enduml
```


ClickServlet.handleRequest()
new page
Page.onSecurityCheck
false
true
Page.onInit()
isForward?
no
Process controls
continue processing?
no
yes
isPost?
no
Page.onPost()
Page.onGet()
Page.onRender()
yes
do redirect?
redirect process
yes
do forward?
no
Forward request
Render page template

6 Component Diagram

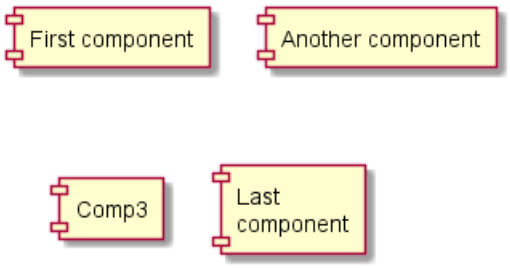
Let's have few examples :

6.1 Components

Components must be bracketed.

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

```
@startuml

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

@enduml
```



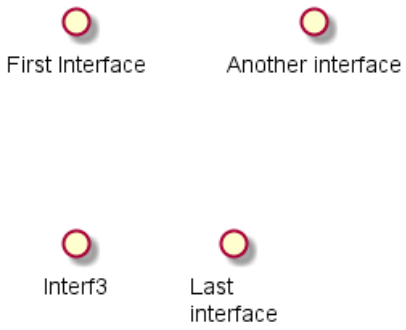
6.2 Interfaces

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

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

We will see latter that interface definition is optional.

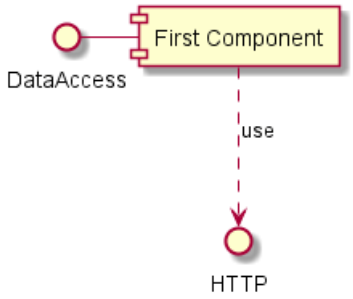
```
@startuml

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

@enduml
```



6.3 Basic example

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

```
@startuml

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

@enduml
```

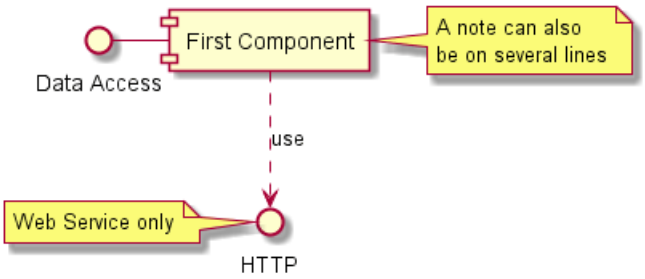


6.4 Using notes

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

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

```
@startuml

interface "Data Access" as DA

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

note left of HTTP : Web Service only

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

@enduml
```



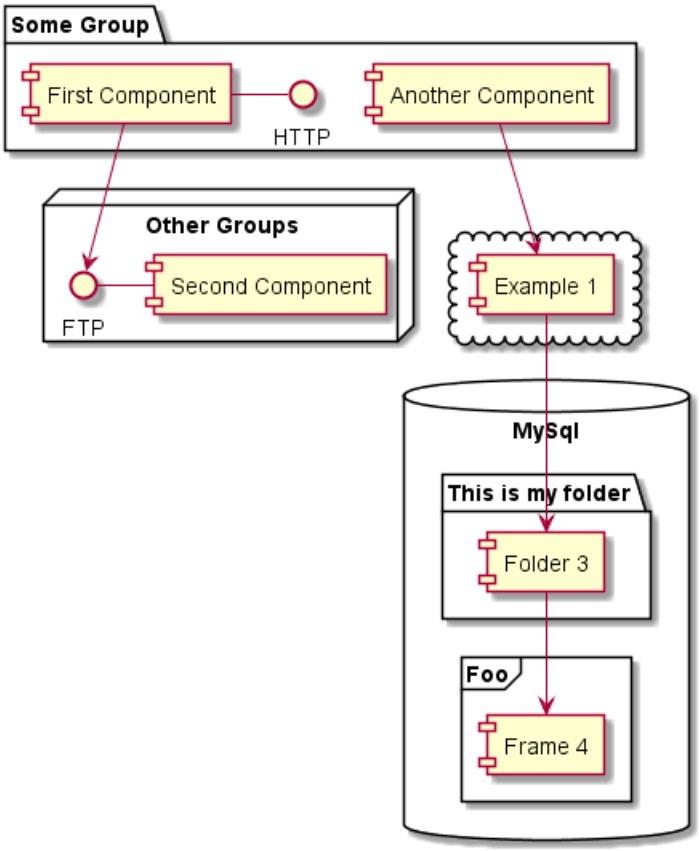
6.5 Grouping Components

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

- `package`
- `node`

- folder
- frame
- cloud
- database

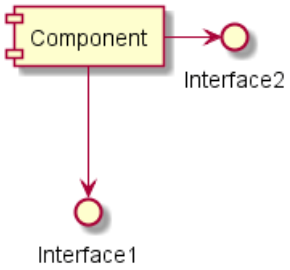
```
@startuml

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

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

cloud {
  [Example 1]
}


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


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

@enduml
```



6.6 Changing arrows direction

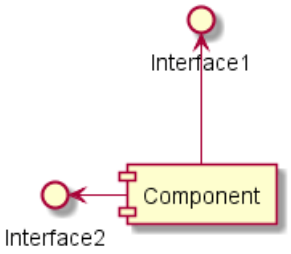
By default, links between classes have two dashes -- and are vertically oriented. It is possible to use horizontal link by putting a single dash (or dot) like this:

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



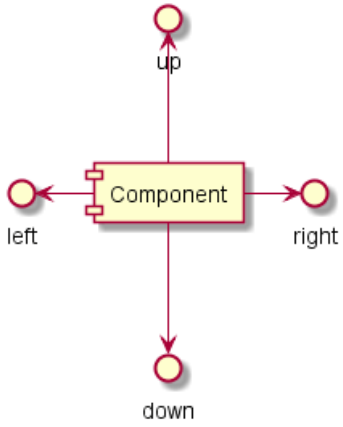
You can also change directions by reversing the link:

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



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

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



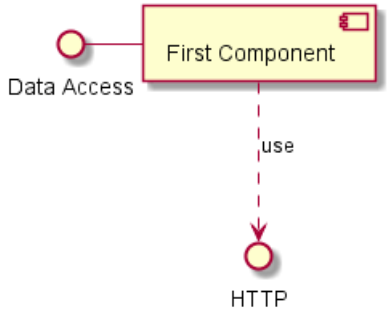
You can shorten the arrow by using only the first character of the direction (for example, `-d-` instead of `-down-`) or the two first characters (`-do-`).

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

6.7 Use UML2 notation

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

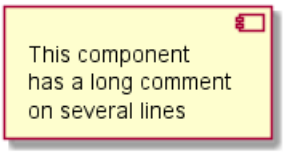
```
@startuml
skinparam componentStyle uml2

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.

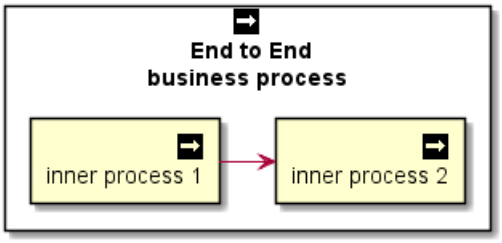
```
@startuml
component  [Web Server] #Yellow
@enduml
```

6.10 Using Sprite in Stereotype

You can use sprites within stereotype components.

```
@startuml
sprite $businessProcess [16x16/16] {
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFF0FFFFF
FFFFFFFFFF00FFFF
FF000000000000FFF
FF0000000000000FF
FF000000000000FFF
FFFFFFFFFF00FFFF
FFFFFFFFFF0FFFFF
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

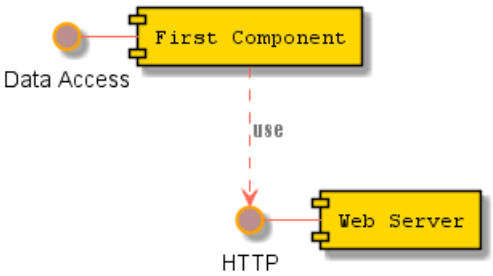
You can use the `skinparam` command to change colors and fonts for the drawing.

You can use this command :

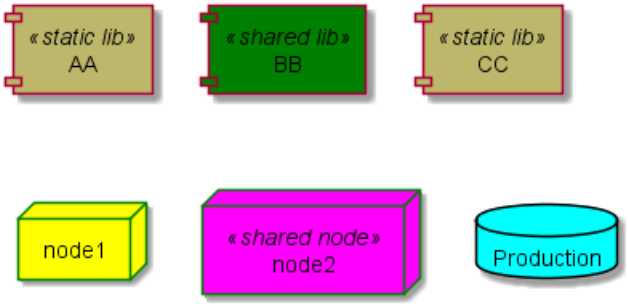
- In the diagram definition, like any other commands,
- In an included file,
- In a configuration file, provided in the command line or the ANT task.

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
```



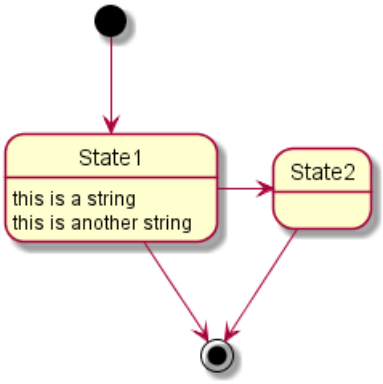
7 State Diagram

7.1 Simple State

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

Use `-->` for arrows.

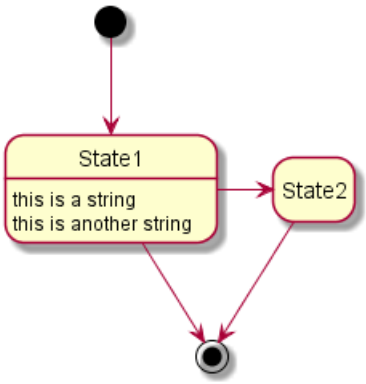
```
@startuml

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

State1 -> State2
State2 --> [*]

@enduml
```



7.2 Change state rendering

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

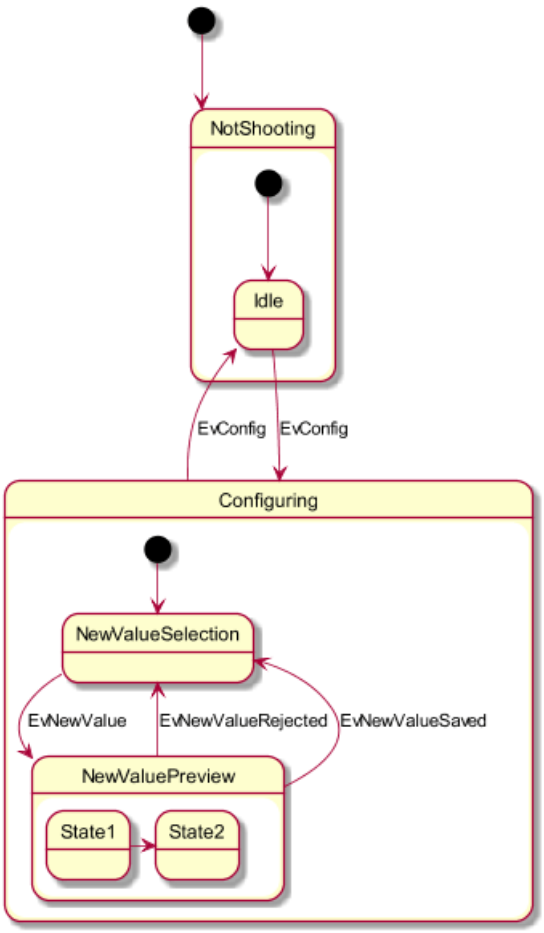
```
@startuml
hide empty description
[*] --> State1
State1 --> [*]
State1 : this is a string
State1 : this is another string

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



7.3 Composite state

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

```
@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
```

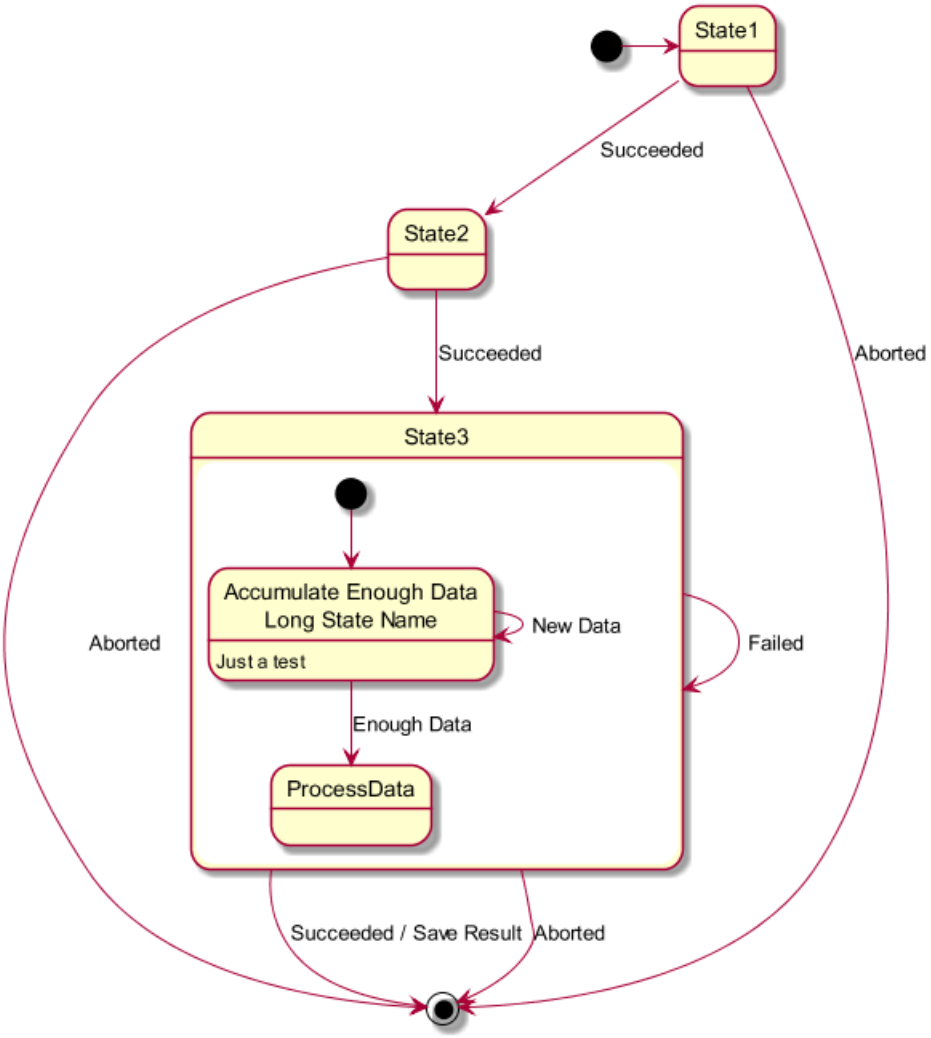


7.4 Long name

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

```
@startuml
scale 600 width
```

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

@enduml
```

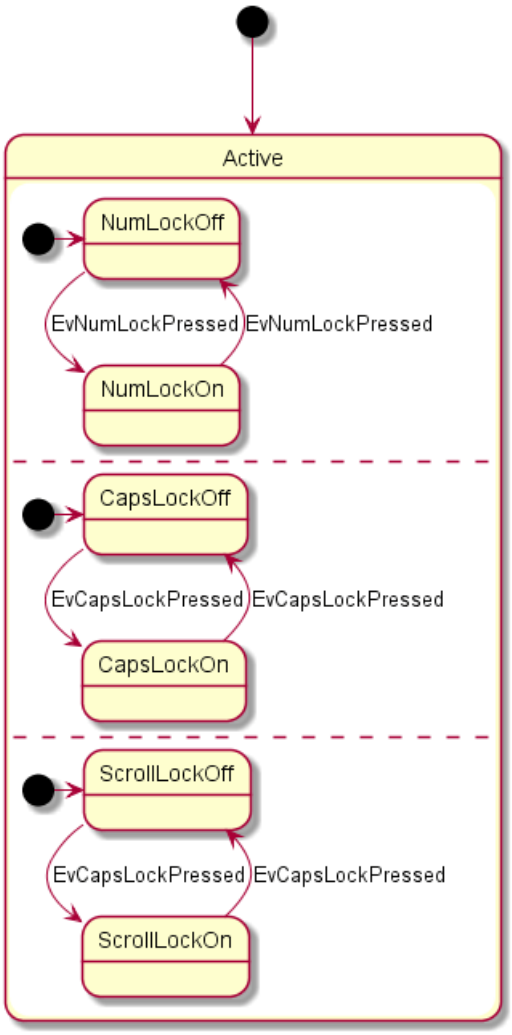


7.5 Concurrent state

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
```



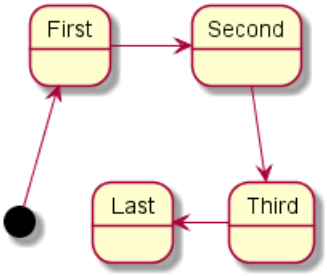
7.6 Arrow direction

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-`).

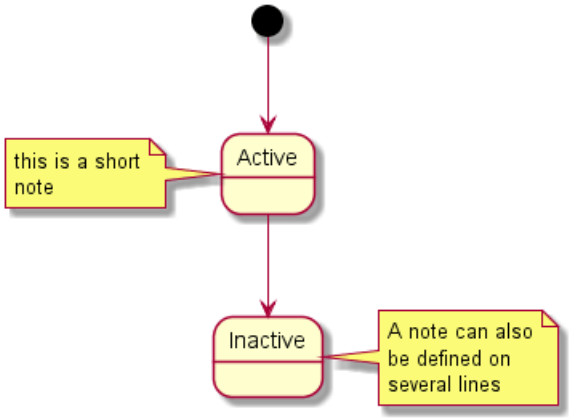
Please note that you should not abuse this functionality : *Graphviz* gives usually good results without tweaking.

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
```



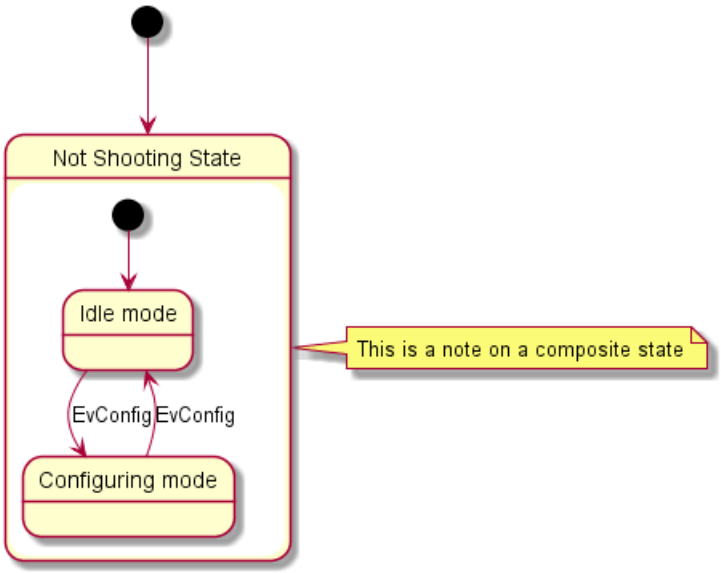
You can also have floating notes.

```
@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.

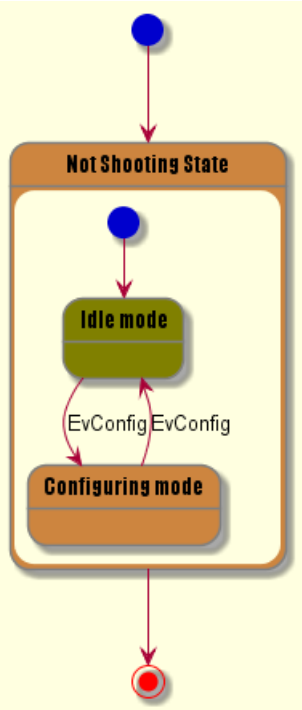
You can use this command :

- 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--`	

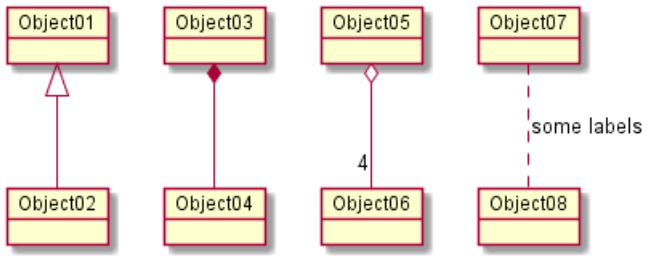
It is possible to replace `--` by `..` to have a dotted line.

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

It is possible a 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
```

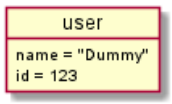


8.3 Adding fields

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

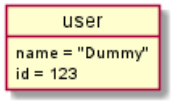
```
@startuml

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.

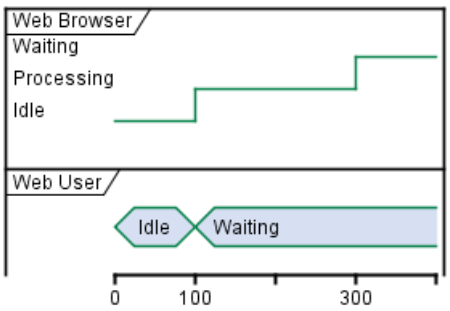
You are very welcome to create a new discussion on this future syntax. Your feedbacks, ideas and suggestions help us to find the right solution.

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.

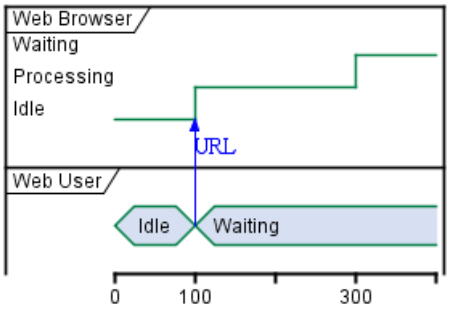
```
@startuml
robust "Web Browser" as WB
concise "Web User" as WU

@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.

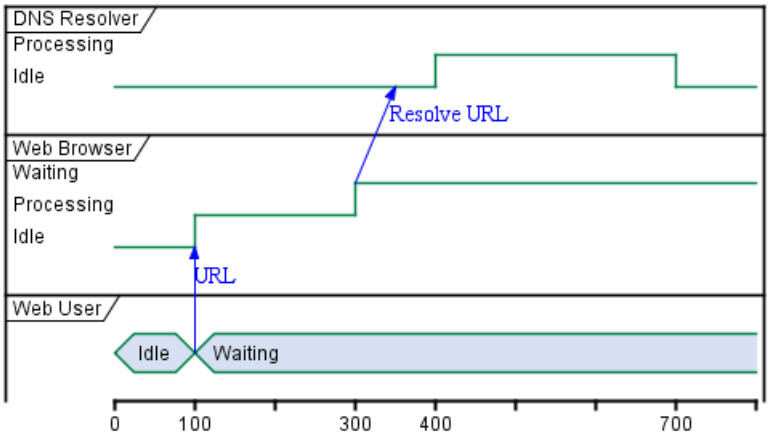
```
@startuml
robust "Web Browser" as WB
concise "Web User" as WU

@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
```

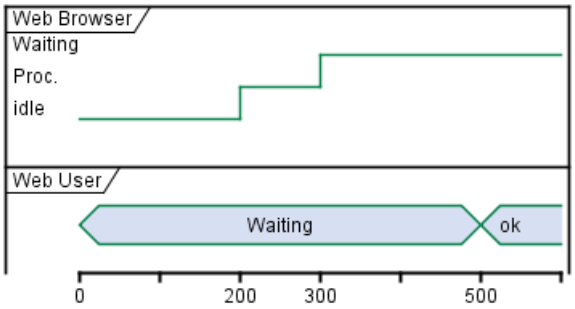


9.4 Participant oriented

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
```

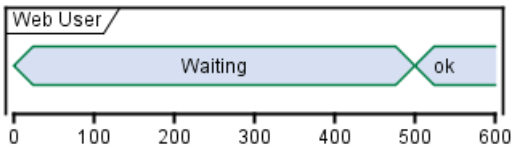
```
@WB
0 is idle
+200 is Proc.
+100 is Waiting

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



9.5 Setting scale

You can also set a specific scale.

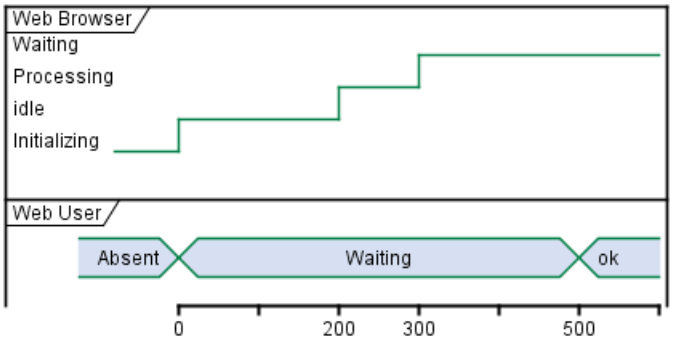
```
@startuml
concise "Web User" as WU
scale 100 as 50 pixels

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



9.6 Initial state

You can also define an inital state.

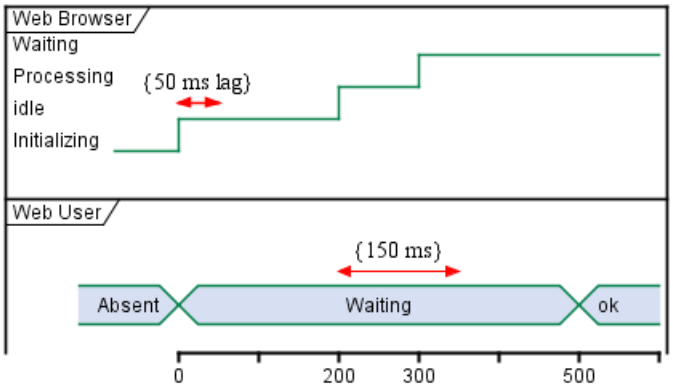
```
@startuml
robust "Web Browser" as WB
concise "Web User" as WU

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 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
```

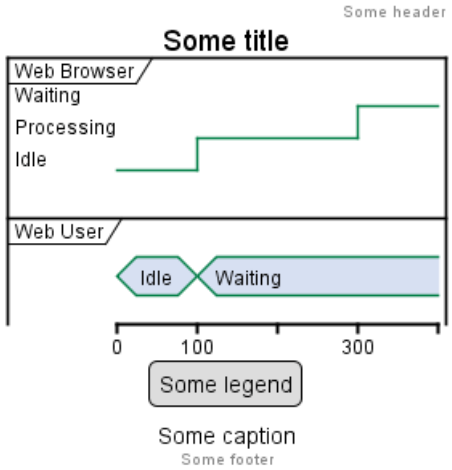


9.8 Adding texts

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.

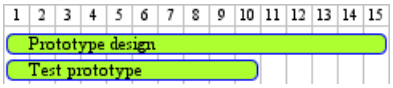
You are very welcome to create a new discussion on this future syntax. Your feedbacks, ideas and suggestions help us to find the right solution.

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:

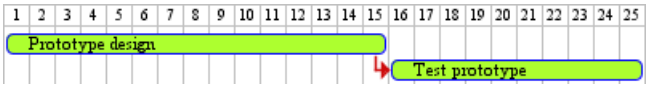
```
@startgantt
[Prototype design] lasts 15 days
[Test prototype] lasts 10 days
@endgantt
```



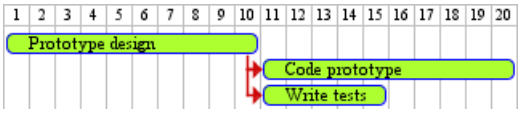
10.2 Adding constraints

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
```



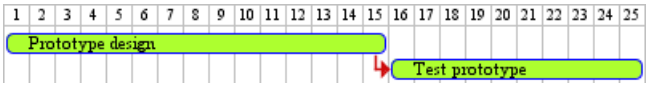

```
@startgantt
[Prototype design] lasts 10 days
[Code prototype] lasts 10 days
[Write tests] lasts 5 days
[Code prototype] starts at [Prototype design]'s end
[Write tests] starts at [Code prototype]'s start
@endgantt
```



10.3 Short names

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

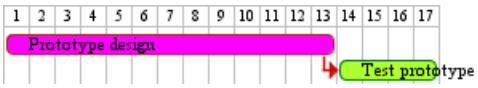
```
@startgantt
[Prototype design] as [D] lasts 15 days
[Test prototype] as [T] lasts 10 days
[T] starts at [D]'s end
@endgantt
```



10.4 Customize colors

It also possible to customize colors.

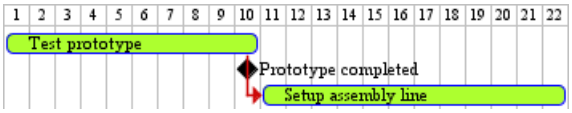
```
@startgantt
[Prototype design] lasts 13 days
[Test prototype] lasts 4 days
[Test prototype] starts at [Prototype design]'s end
[Prototype design] is colored in Fuchsia/FireBrick
[Test prototype] is colored in GreenYellow/Green
@endgantt
```



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

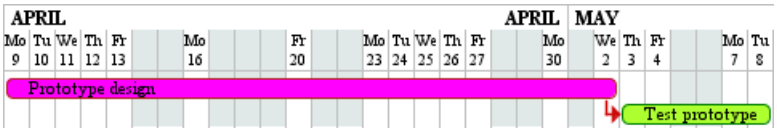
You can specify a starting date for the whole project. By default, the first task starts at this date.

```
@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.

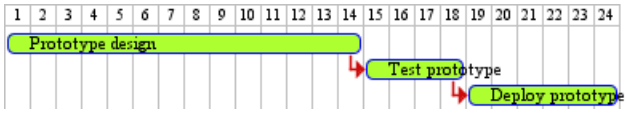
```
@startgantt
project starts the 2018/04/09
saturday are closed
sunday are closed
2018/05/01 is closed
2018/04/17 to 2018/04/19 is closed
[Prototype design] lasts 14 days
[Test prototype] lasts 4 days
[Test prototype] starts at [Prototype design]'s end
[Prototype design] is colored in Fuchsia/FireBrick
[Test prototype] is colored in GreenYellow/Green
@endgantt
```



10.8 Simplified task succession

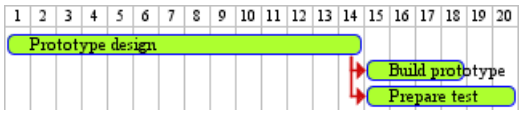
It's possible to use the `then` keyword to denote consecutive tasks.

```
@startgantt
[Prototype design] lasts 14 days
then [Test prototype] lasts 4 days
then [Deploy prototype] lasts 6 days
@endgantt
```



You can also use arrow `->`

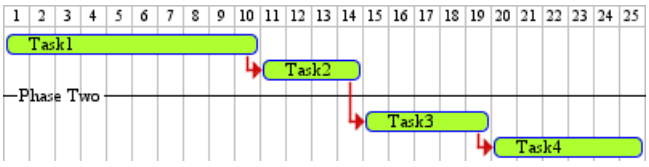
```
@startgantt
[Prototype design] lasts 14 days
[Build prototype] lasts 4 days
[Prepare test] lasts 6 days
[Prototype design] -> [Build prototype]
[Prototype design] -> [Prepare test]
@endgantt
```



10.9 Separator

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

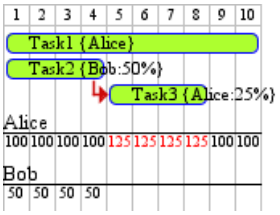
```
@startgantt
[Task1] lasts 10 days
then [Task2] lasts 4 days
-- Phase Two --
then [Task3] lasts 5 days
then [Task4] lasts 6 days
@endgantt
```



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
@endgantt
```

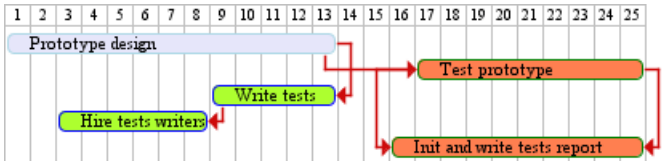


10.11 Complex example

It also possible to use the and conjunction.

You can also add delays in constraints.

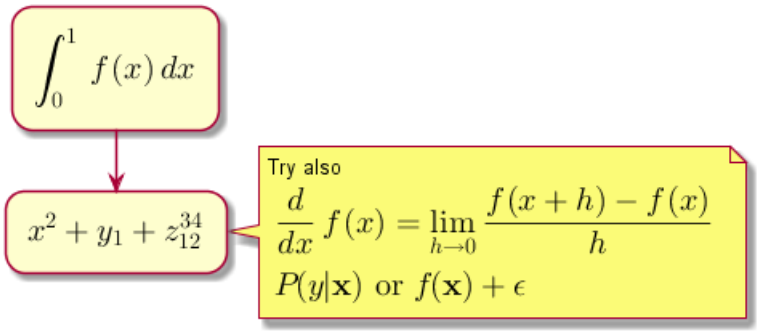
```
@startgantt
[Prototype design] lasts 13 days and is colored in Lavender/LightBlue
[Test prototype] lasts 9 days and is colored in Coral/Green and starts 3 days after [Prototype design]'s end
[Write tests] lasts 5 days and ends at [Prototype design]'s end
[Hire tests writers] lasts 6 days and ends at [Write tests]'s start
[Init and write tests report] is colored in Coral/Green
[Init and write tests report] starts 1 day before [Test prototype]'s start and ends at [Test prototype]'s end
@endgantt
```



11 Maths

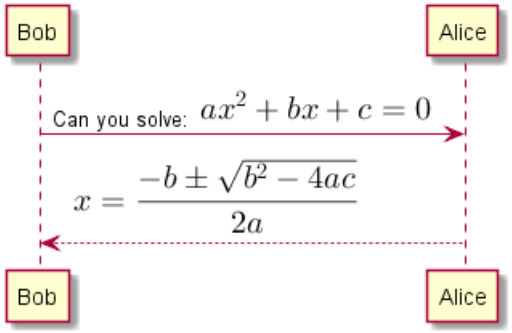
You can use AsciiMath or JLaTeXMath notation within PlantUML:

```
@startuml
:<math>int_0^1f(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
```



11.1 Standalone diagram

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

```
@startmath
f(t)=(a_0)/2 + sum_(n=1)^ooa_ncos((npit)/L)+sum_(n=1)^oo b_n\ sin((npit)/L)
@endmath
```

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(\frac{n\pi t}{L}\right) + \sum_{n=1}^{\infty} b_n \sin\left(\frac{n\pi t}{L}\right)$$

Or use `@startlatex`/`@endlatex` to create standalone JLaTeXMath formula.

```
@startlatex
\sum_{i=0}^{n-1} (a_i + b_i^2)
@endlatex
```

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

11.2 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.

12 Common commands

12.1 Comments

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

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

12.2 Footer and header

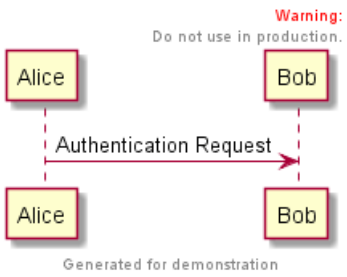
You can use the commands `header` or `footer` to add a footer or a header on any generated diagram.

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
```



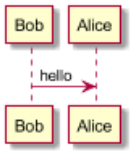
12.3 Zoom

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
```

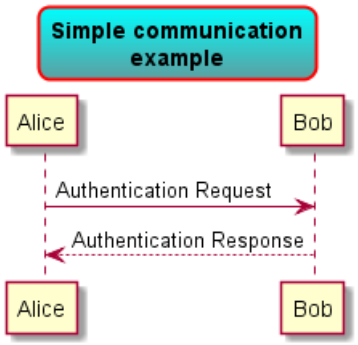


12.4 Title

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.

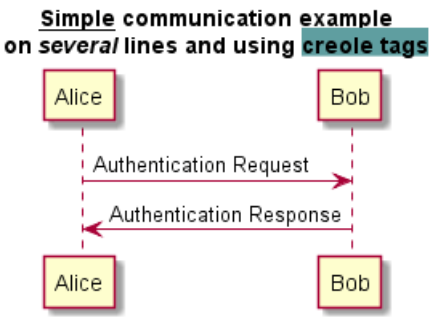
```
@startuml
skinparam titleBorderRoundCorner 15
skinparam titleBorderThickness 2
skinparam titleBorderColor red
skinparam titleBackgroundColor Aqua-CadetBlue

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.

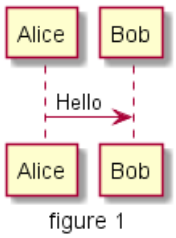
```
@startuml

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
```



12.5 Caption

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

```
@startuml

caption figure 1
Alice -> Bob: Hello

@enduml
```

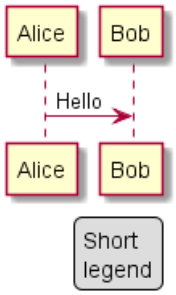


12.6 Legend the diagram

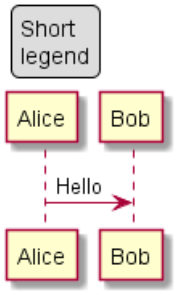
The `legend` and `end legend` are keywords is used to put a legend.

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
```



13 Salt (wireframe)

Salt is a subproject included in PlantUML that may help you to design graphical interface.

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

13.1 Basic widgets

A window must start and end with brackets. You can then define:

- Button using `[` and `]`.
- Radio button using `(` and `)`.
- Checkbox using `[` and `]`.
- User text area using `"`.

```
@startuml
salt
{
  Just plain text
  [This is my button]
  ()  Unchecked radio
  (X) Checked radio
  []  Unchecked box
  [X] Checked box
  "Enter text here   "
  ^This is a droplist^
}
@enduml
```

The goal of this tool is to discuss about simple and sample windows.

13.2 Using grid

A table is automatically created when you use an opening bracket `{`. And you have to use `|` to separate columns.

For example:

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

Just after the opening bracket, you can use a character to define if you want to draw lines or columns of the grid :

Symbol	Result
#	To display all vertical and horizontal lines
!	To display all vertical lines
-	To display all horizontal lines
+	To display external lines

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

13.3 Group box

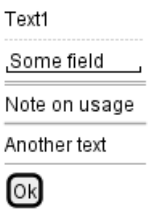
more info

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

13.4 Using separator

You can use several horizontal lines as separator.

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



13.5 Tree widget

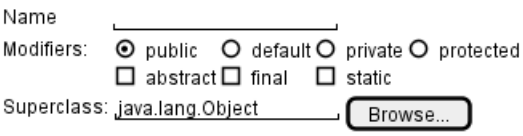
To have a Tree, you have to start with {T and to use + to denote hierarchy.

```
@startsalt
{
{T
 + World
 ++ America
 +++ Canada
 +++ USA
 ++++ New York
 ++++ Boston
 +++ Mexico
 ++ Europe
 +++ Italy
 +++ Germany
 ++++ Berlin
 ++ Africa
}
}
@endsalt
```

13.6 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
```

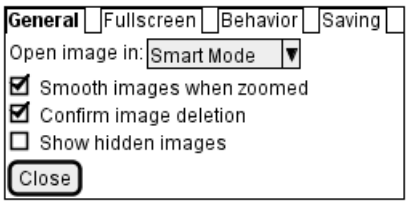


13.7 Adding tabs

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

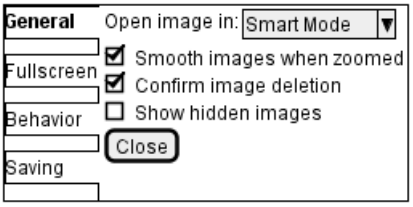
```
@startsalt
{+
{/ <b>General | Fullscreen | Behavior | Saving }
{
        { Open image in: | ^Smart Mode^ }
        [X] Smooth images when zoomed
        [X] Confirm image deletion
        [ ] Show hidden images
}
[Close]
```

```
}
@endsalt
```



Tab could also be vertically oriented:

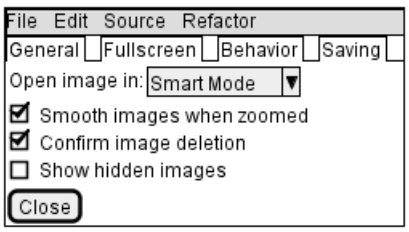
```
@startsalt
{+
{/ <b>General
Fullscreen
Behavior
Saving } |
{
        { Open image in: | ^Smart Mode^ }
        [X] Smooth images when zoomed
        [X] Confirm image deletion
        [ ] Show hidden images
        [Close]
}
}
@endsalt
```



13.8 Using menu

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

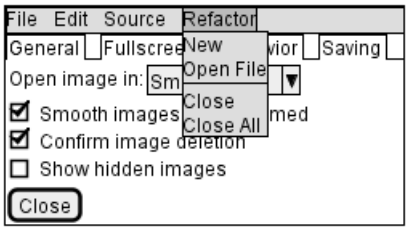
```
@startsalt
{+
{* File | Edit | Source | Refactor }
{/ General | Fullscreen | Behavior | Saving }
{
        { Open image in: | ^Smart Mode^ }
        [X] Smooth images when zoomed
        [X] Confirm image deletion
        [ ] Show hidden images
}
[Close]
}
@endsalt
```



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
```



13.9 Advanced table

You can use two special notations for table :

- `*` to indicate that a cell with span with left
- `.` to denotate an empty cell

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

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

13.10 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:

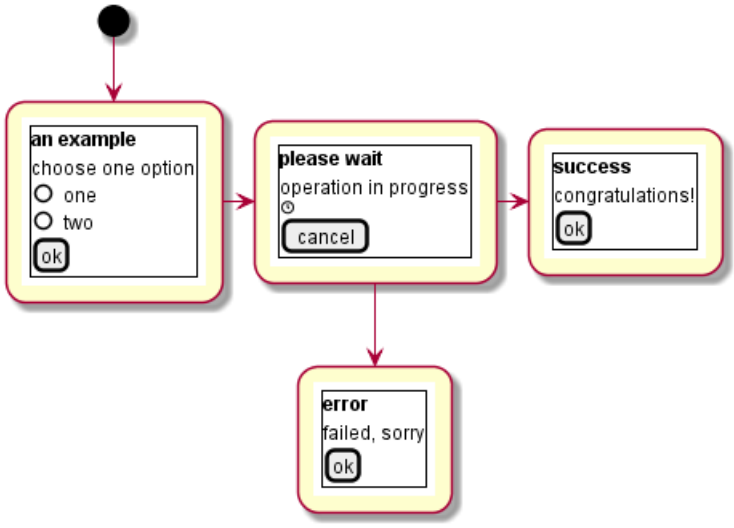
```
@startuml
listopeniconic
@enduml
```

13.11 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
```

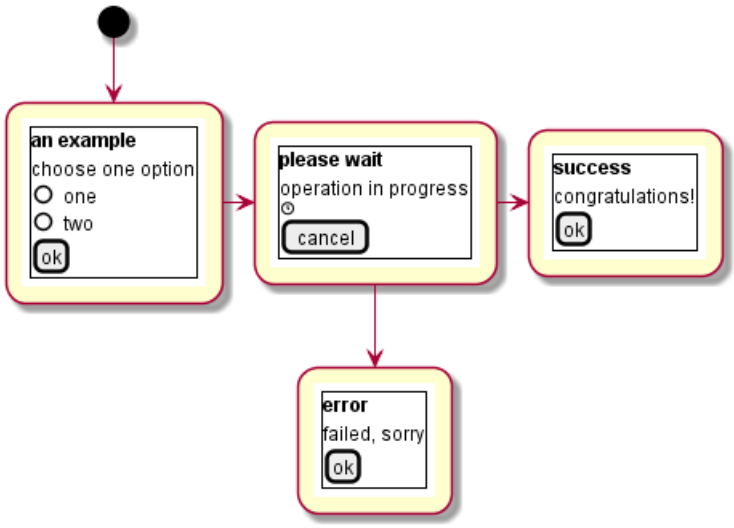
```
wait -down-> "
{{
salt
{+
<b>error
failed, sorry
[ok]
}
}}
"
@enduml
```



It can also be combined with define macro.

```
@startuml
!definelong SALT(x)
"{{
salt
_##x
}}
" as x
!enddefinelong

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

!definelong _wait
{+
<b>please wait
operation in progress
<&clock>
[cancel]
}
!enddefinelong

!definelong _success
{+
<b>success
congratulations!
[ok]
}
!enddefinelong

!definelong _error
{+
```

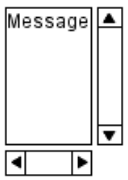
```
<b>error
failed, sorry
[ok]
}
!enddefinelong

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

@enduml
```



13.12 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
```

14 Creole

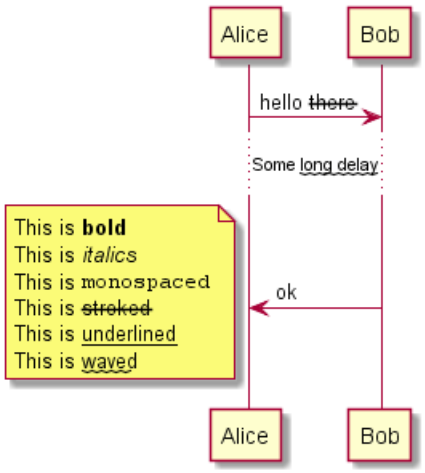
A light Creole engine has been integrated into PlantUML to have a standardized way of defining text style.

All diagrams are now supporting this syntax.

Note that ascending compatibility with HTML syntax is preserved.

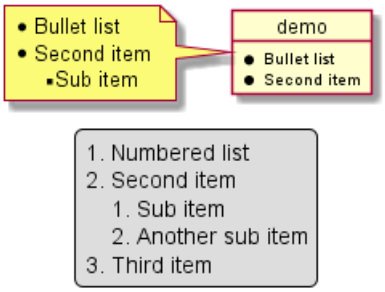
14.1 Emphasized text

```
@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
```



14.2 List

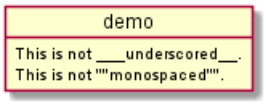
```
@startuml
object demo {
  * Bullet list
  * Second item
}
note left
  * Bullet list
  * Second item
  ** Sub item
end note

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



14.3 Escape character

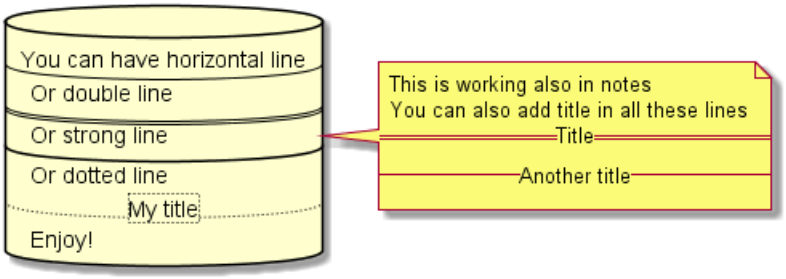
You can use the tilde ~ to escape special creole characters.

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



14.4 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
```



14.5 Headings

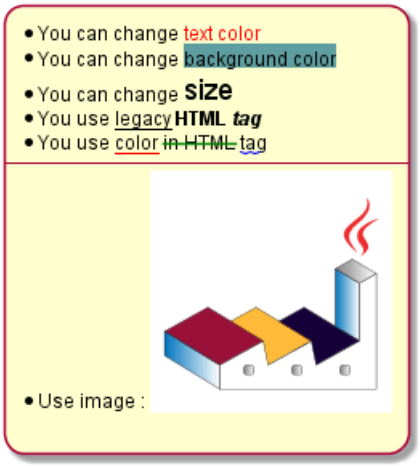
```
@startuml
usecase UC1 as "
= Extra-large heading
Some text
== Large heading
Other text
=== Medium heading
Information
....
==== Small heading"
@enduml
```

14.6 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

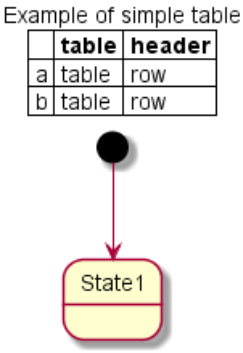
```
@startuml
:* You can change <color:red>text color</color>
* You can change <back:cadetblue>background color</back>
* You can change <size:18>size</size>
* You use <u>legacy</u> <b>HTML <i>tag</i></b>
* You use <u:red>color</u> <s:green>in HTML</s> <w:#0000FF>tag</w>
----
* Use image : <img:http://plantuml.com/logo3.png>
;
@enduml
```



14.7 Table

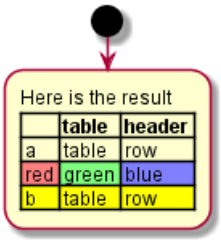
It is possible to build table.

```
@startuml
skinparam titleFontSize 14
title
  Example of simple table
  |= |= table |= header |
  | a | table | row |
  | b | table | row |
end title
[*] --> State1
@enduml
```



You can specify background colors for cells and lines.

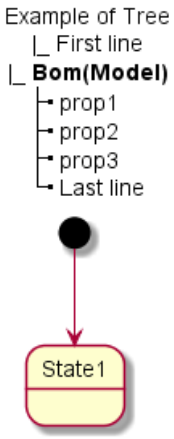
```
@startuml
start
:Here is the result
|= |= table |= header |
| a | table | row |
|<#FF8080> red |<#80FF80> green |<#8080FF> blue |
<#yellow>| b | table | row |;
@enduml
```



14.8 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
```



14.9 Special characters

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

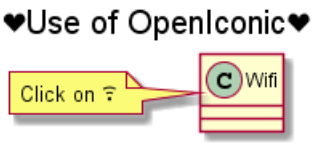
```
@startuml
usecase foo as "this is &#8734; long"
usecase bar as "this is also <U+221E> long"
@enduml
```

14.10 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
```



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
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
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

15 Defining and using sprites

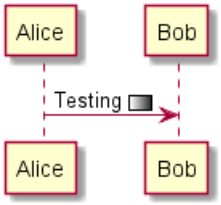
A *Sprite* is a small graphic element that can be used in diagrams.

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.

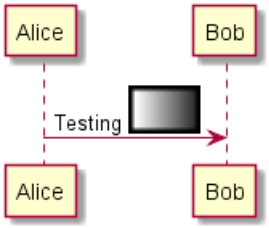
Then you can use the sprite using `<$XXX>` where XXX is the name of the sprite.

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



You can scale the sprite.

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



15.1 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.

15.2 Importing Sprite

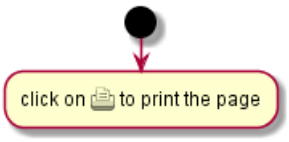
You can also launch the GUI to generate a sprite from an existing image.

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

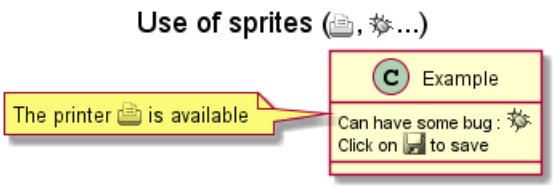
After copying an image into you clipboard, several possible definitions of the corresponding sprite will be displayed : you will just have to pickup the one you want.

15.3 Examples

```
@startuml
sprite $printer [15x15/8z] NOtH3W0W208HxFz_kMAhj7lHWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvFfPEqm0
start
:click on <$printer> to print the page;
@enduml
```




```
@startuml
 sprite $bug [15x15/16z] PKzR2i0m2BFMi15p__FEjQEqB1z27aeqCqixa8S4OT7C53cKpsHpaYPDJY_12MHM-BLRyywPhrrlw3qumqNTh
 sprite $printer [15x15/8z] NOtH3W0W208HxFz_kMAhj7lHWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvFfPEqm0
 sprite $disk {
   444445566677881
   436000000009991
   43600000000ACA1
   53700000001A7A1
   53700000012B8A1
   53800000123B8A1
   63800001233C9A1
   634999AABBC99B1
   744566778899AB1
   7456AAAAA99AAB1
   8566AFC228AABB1
   8567AC8118BBBB1
   867BD4433BBBBB1
   39AAAAABBBBBBC1
}

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

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

 note left : The printer <$printer> is available

@enduml
```



16 Skinparam command

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

Example:

```
skinparam backgroundColor transparent
```

16.1 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.

16.2 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
}
```

16.3 List

Since the documentation is not always up to date, you can have the complete list of parameters using this command:

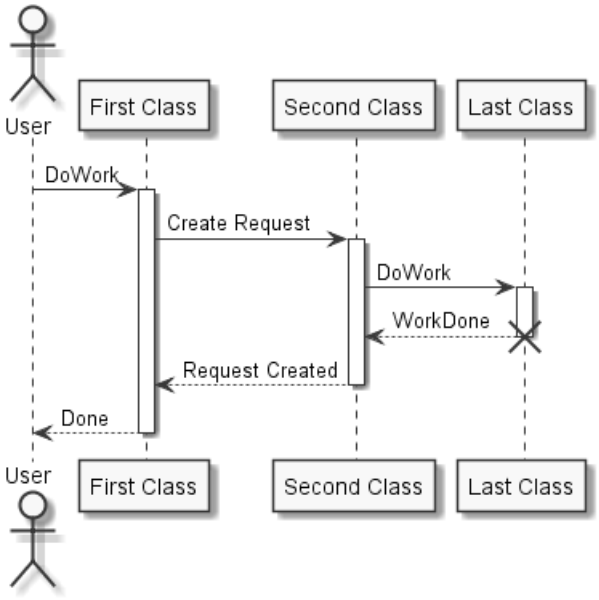
```
java -jar plantuml.jar -language
```

16.4 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
```

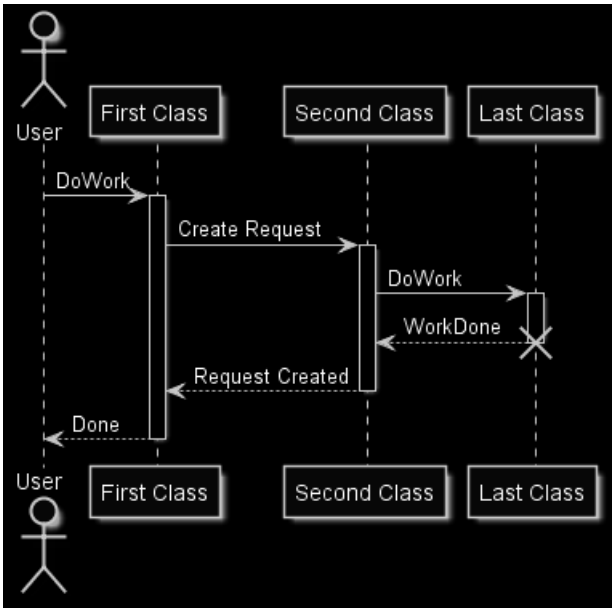
```
B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml
```



16.5 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
```



16.6 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.

16.7 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:

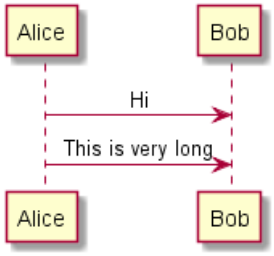
```
java -jar plantuml.jar -language
```

16.8 Text Alignment

Text alignment can be set up to `left`, `right` or `center`. You can also use `direction` or `reverseDirection` values for `sequenceMessageAlign` which align text depending on arrow direction.

Param name	Default value	Comment
sequenceMessageAlign	left	Used for messages in sequence diagrams
sequenceReferenceAlign	center	Used for `ref over` in sequence diagrams

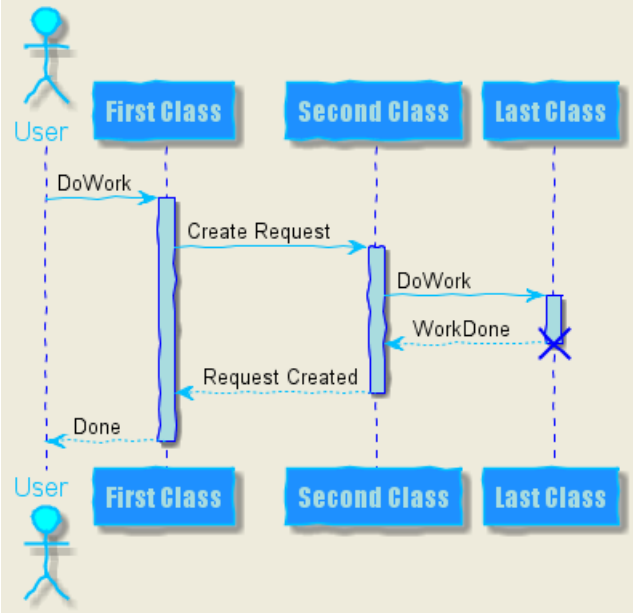
```
@startuml
skinparam sequenceMessageAlign center
Alice -> Bob : Hi
Alice -> Bob : This is very long
@enduml
```



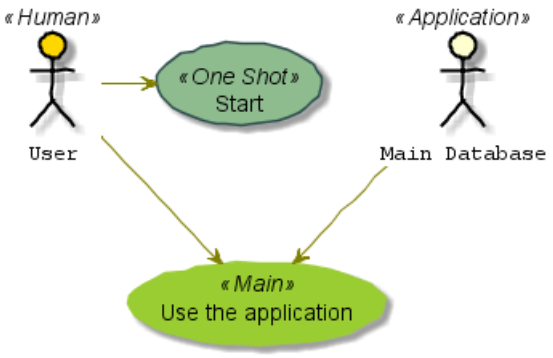
16.9 Examples

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

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

        ParticipantBorderColor DeepSkyBlue
        ParticipantBackgroundColor DodgerBlue
        ParticipantFontName Impact
        ParticipantFontSize 17
        ParticipantFontColor #A9DCDF

        ActorBackgroundColor aqua
        ActorFontColor DeepSkyBlue
        ActorFontSize 17
        ActorFontName Aapex
}

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

User -> A: DoWork
activate A

A -> B: Create Request
activate B
```

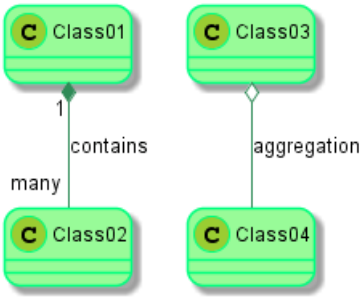
```
B -> C: DoWork
activate C
C --> B: WorkDone
destroy C

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml
```



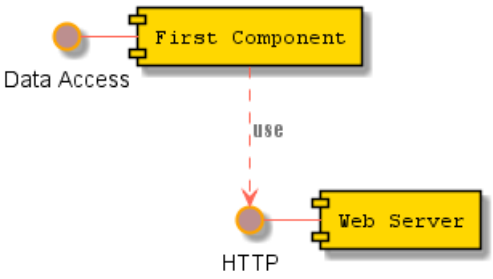

```
@startuml
skinparam handwritten true

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

skinparam usecase {
        BackgroundColor DarkSeaGreen
        BorderColor DarkSlateGray

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

        ArrowColor Olive
}

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

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

MySql --> (Use)
@enduml
```



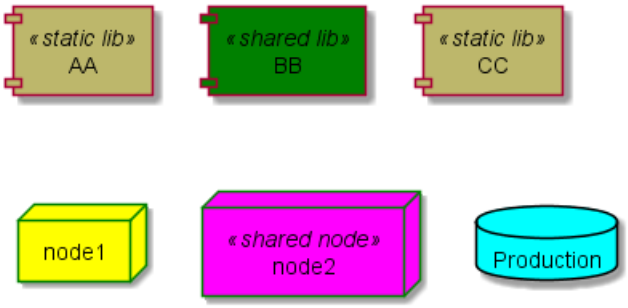

```
@startuml
skinparam roundcorner 20
skinparam class {
        BackgroundColor PaleGreen
        ArrowColor SeaGreen
        BorderColor SpringGreen
}
skinparam stereotypeCBackgroundColor YellowGreen

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

Class03 o-- Class04 : aggregation
@enduml
```




```
@startuml

skinparam interface {
  backgroundColor RosyBrown
  borderColor orange
}

skinparam component {
  FontSize 13
  BackgroundColor<<Apache>> 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
```



17 Preprocessing

Some minor preprocessing capabilities are included in **PlantUML**, and available for *all* diagrams.

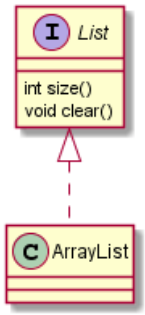
Those functionnalities are very similar to the C language preprocessor, except that the special character # has been changed to the exclamation mark `!`.

17.1 Including files

Use the `!include` directive to include file in your diagram.

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.

A file can be only be included once. If you want to include several times the very same file, you have to use the directive `!include_many` instead of `!include`.

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.

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`.

17.2 Including URL

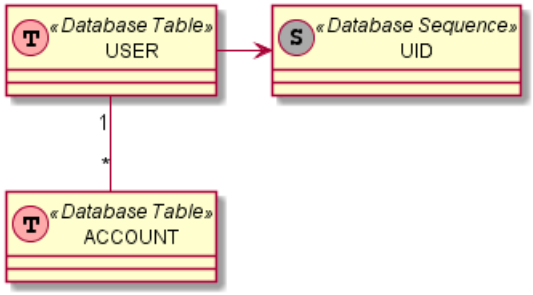
Use the `!includeurl` directive to include file from Internet/Intranet in your diagram.

You can also use `!includeurl http://someurl.com/mypath!0` to specify which `@startuml/@enduml` block from `http://someurl.com/mypath` you want to include. The `!0` notation denotes the first diagram.

17.3 Constant definition

You can define constant using the `!define` directive. As in C language, a constant name can only use alphanumeric and underscore characters, and cannot start with a digit.

```
@startuml

!define SEQUENCE (S,#AAAAAA) Database Sequence
!define TABLE (T,#FFAAAA) Database Table

class USER << TABLE >>
class ACCOUNT << TABLE >>
class UID << SEQUENCE >>
USER "1" -- "*" ACCOUNT
USER -> UID
@enduml
```



Of course, you can use the `!include` directive to define all your constants in a single file that you include in your diagram.

Constant can be undefined with the `!undef XXX` directive.

You can also specify constants within the command line, with the `-D` flags.

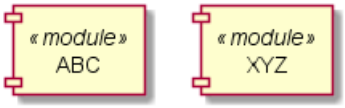
```
java -jar plantuml.jar -DTITLE="My title" atest1.txt
```

Note that the `-D` flag must be put after the "`-jar plantuml.jar`" section.

17.4 Macro definition

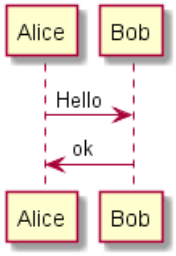
You can also define macro with arguments.

```
@startuml

!define module(x) component x <<module>>
module(ABC)
module(XYZ)
@enduml
```



Macro can have several arguments.

```
@startuml
!define send(a,b,c) a->b : c
send(Alice, Bob, Hello)
send(Bob, Alice, ok)
@enduml
```

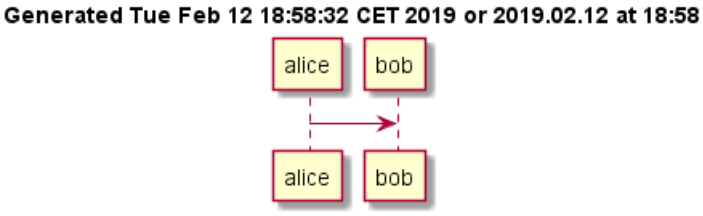


17.5 Adding date and time

You can also expand current date and time using the special variable `%date%`.

Date format can be specified using format specified in SimpleDataFormat documentation.

```
@startuml
!define ANOTHER_DATE %date[yyyy.MM.dd 'at' HH:mm]%
Title Generated %date% or ANOTHER_DATE
alice -> bob
@enduml
```



17.6 Other special variables

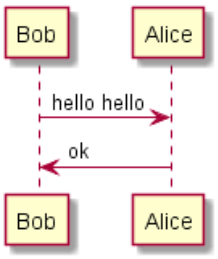
You can also use the following special variables:

Variable	Content
%dirpath%	Path of the current file
%filename%	Name of the current file

17.7 Macro on several lines

You can also define macro on several lines using `!definelong` and `!enddefinelong`.

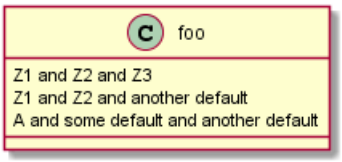
```
@startuml
!define DOUBLE(x) x x
!definelong AUTHEN(x,y)
x -> y : DOUBLE(hello)
y -> x : ok
!enddefinelong

AUTHEN(Bob,Alice)
@enduml
```



17.8 Default values for macro parameters

It is possible to assign default values to macro parameters.

```
@startuml
!define some_macro(x, y = "some default" , z = 'another default' ) x and y and z
class foo {
  some_macro(Z1, Z2, Z3)
  some_macro(Z1, Z2)
  some_macro(A)
}
@enduml
```



17.9 Conditions

You can use `!ifdef XXX` and `!endif` directives to have conditionnal drawings.

The lines between those two directives will be included only if the constant after the `!ifdef` directive has been defined before.

You can also provide a `!else` part which will be included if the constant has **not** been defined.

```
@startuml
!include ArrayList.iuml
@enduml
```

File ArrayList.iuml:

```
class ArrayList
!ifdef SHOW_METHODS
class ArrayList {
  int size()
  void clear()
}
!endif
```

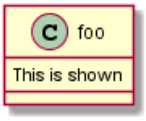
You can then use the `!define` directive to activate the conditionnal part of the diagram.

```
@startuml
!define SHOW_METHODS
!include ArrayList.iuml
@enduml
```

You can also use the `!ifndef` directive that includes lines if the provided constant has NOT been defined.

You can use boolean expression with parenthesis, operators `&&` and `||` in the test.

```
@startuml
!define SHOW_FIELDS
!undef SHOW_METHODS
class foo {
!ifdef SHOW_FIELDS || SHOW_METHODS
This is shown
!endif
!ifdef SHOW_FIELDS && SHOW_METHODS
This is NOT shown
!endif
}
@enduml
```



17.10 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

...
```

17.11 Search path

You can specify the java property `plantuml.include.path` in the command line.

For example:

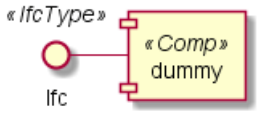
```
java -Dplantuml.include.path="c:/mydir" -jar plantuml.jar atest1.txt
```

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.

17.12 Advanced features

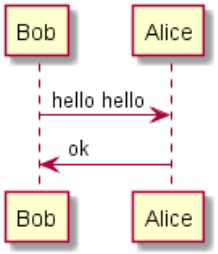
It is possible to append text to a macro argument using the ## syntax.

```
@startuml
!definelong COMP_TEXTGENCOMP(name)
[name] << Comp >>
interface Ifc << IfcType >> AS name##Ifc
name##Ifc - [name]
!enddefinelong
COMP_TEXTGENCOMP(dummy)
@enduml
```



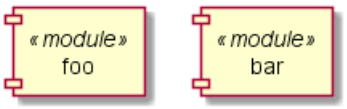
A macro can be defined by another macro.

```
@startuml
!define DOUBLE(x) x x
!definelong AUTHEN(x,y)
x -> y : DOUBLE(hello)
y -> x : ok
!enddefinelong
AUTHEN(Bob,Alice)
@enduml
```



A macro can be polymorphic with argument count.

```
@startuml
!define module(x) component x <<module>>
!define module(x,y) component x as y <<module>>
module(foo)
module(bar, barcode)
@enduml
```



You can use system environment variable or constant definition when using include:

```
!include %windir%/test1.txt
!define PLANTUML_HOME /home/foo
!include PLANTUML_HOME/test1.txt
```

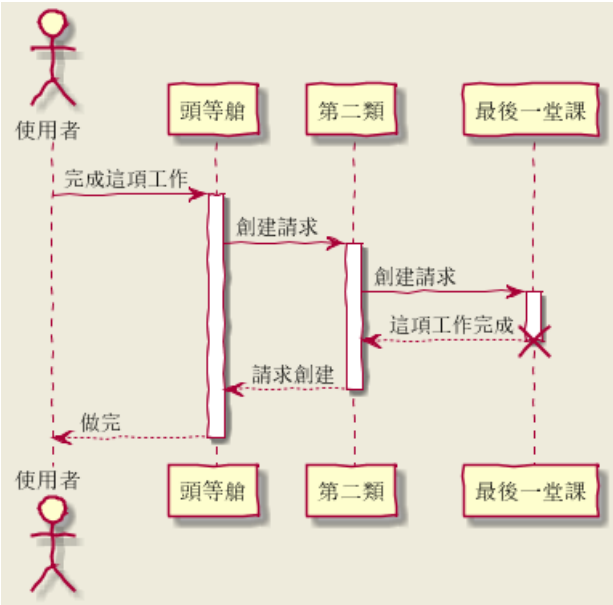
18 Unicode

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*.

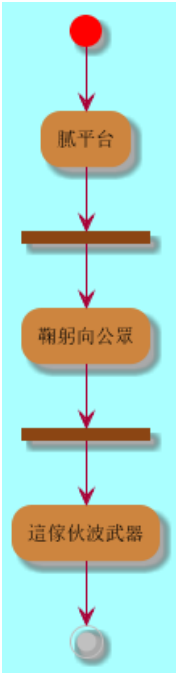
18.1 Examples

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

actor 使用者
participant "頭等艙" as A
participant "第二類" as B
participant "最後一堂課" as 別的東西

使用者 -> A: 完成這項工作
activate A

A -> B: 創建請求
activate B

B -> 別的東西: 創建請求
activate 別的東西
別的東西 --> B: 這項工作完成
destroy 別的東西

B --> A: 請求創建
deactivate B

A --> 使用者: 做完
deactivate A
@enduml
```

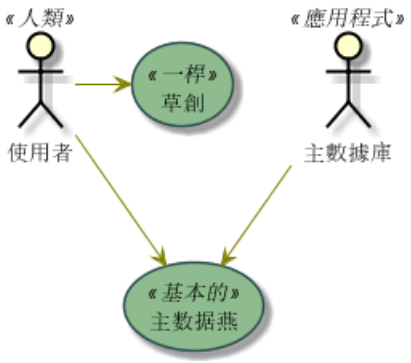



```
@startuml

(*) --> "膩平台"
--> === S1 ===
--> 鞠躬向公眾
--> === S2 ===
--> 這傢伙波武器
--> (*)

skinparam backgroundColor #AAFFFF
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
```



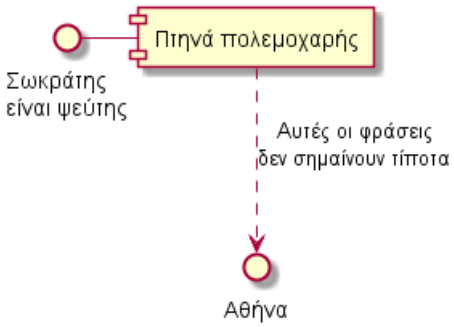
@startuml

() "Σωκράτηςψεύτης" as Σωκράτης

Σωκράτης - [Πτηνά πολεμοχαρής]

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

@enduml



18.2 Charset

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

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

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

Or, with the ant task:

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

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

19 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_li
)

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

19.1 AWS library

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

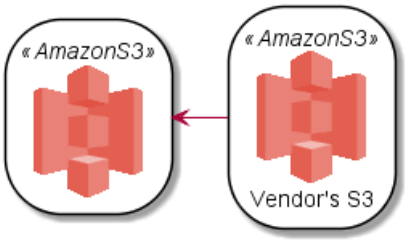
The AWS library consists of Amazon AWS icons, it provides icons of two different sizes.

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
```

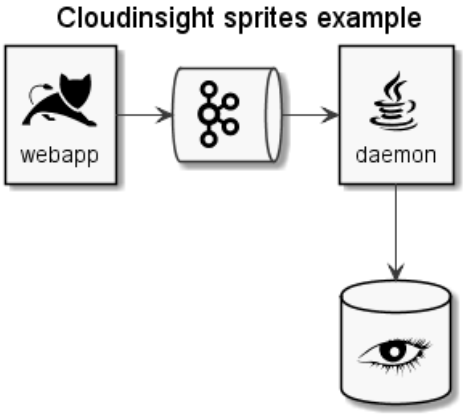


19.2 Cloud Insight

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
```



19.3 Devicons and Font Awesome library

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

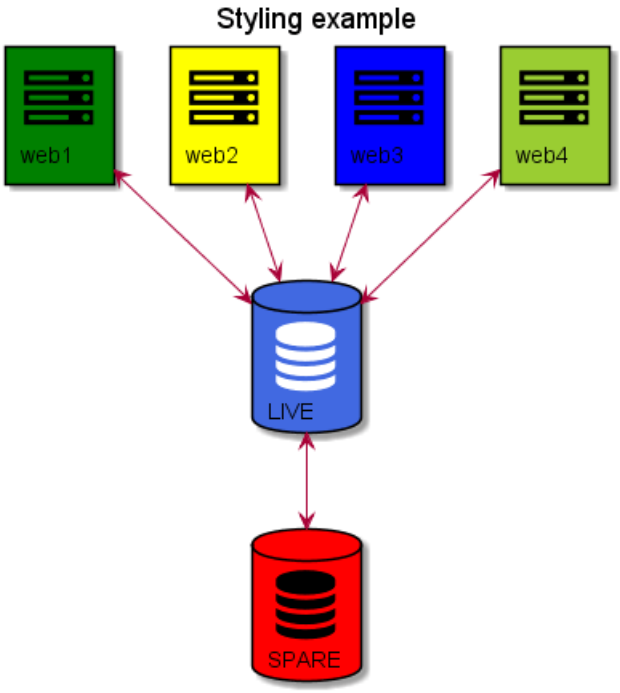
These two library consists respectively of Devicons and Font Awesome libraries of icons.

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>`.

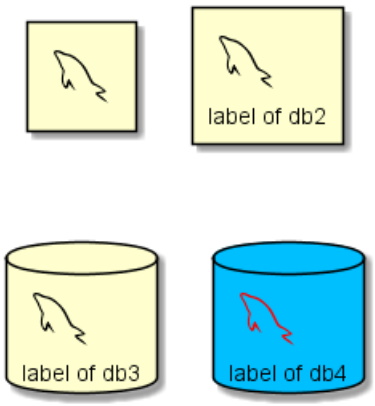
You may also include the `common.puml` file, eg: `!include <font-awesome/common>`, which contains helper macros defined. With the `common.puml` imported, you can use the `NAME_OF_SPRITE(parameters...)` macro.

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
```



19.4 Google Material Icons

https://github.com/Templarian/MaterialDesign

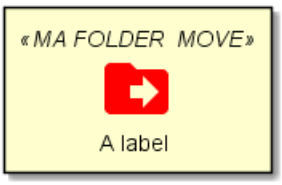
This library consists of a free Material style icons from Google and other artists.

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_` preffix 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
```

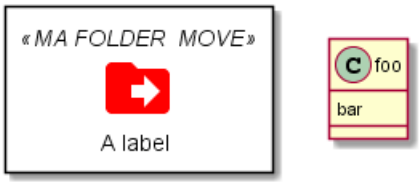


Notes

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
```



19.5 Office

https://github.com/Roemer/plantuml-office

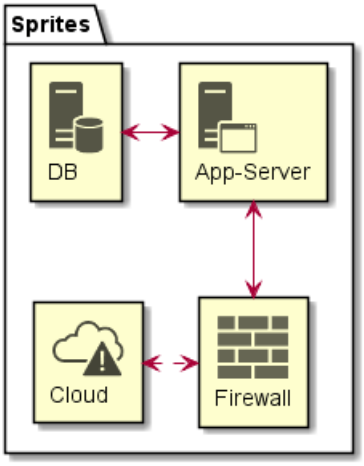
There are sprites (*.puml) and colored png icons available. Be aware that the sprites are all only monchrome even if they have a color in their name (due to automatically generating the files). You can either color the sprites with the macro (see examples below) or directly use the fully colored pngs. See the following examples on how to use the sprites, the pngs and the macros.

Example of usage:

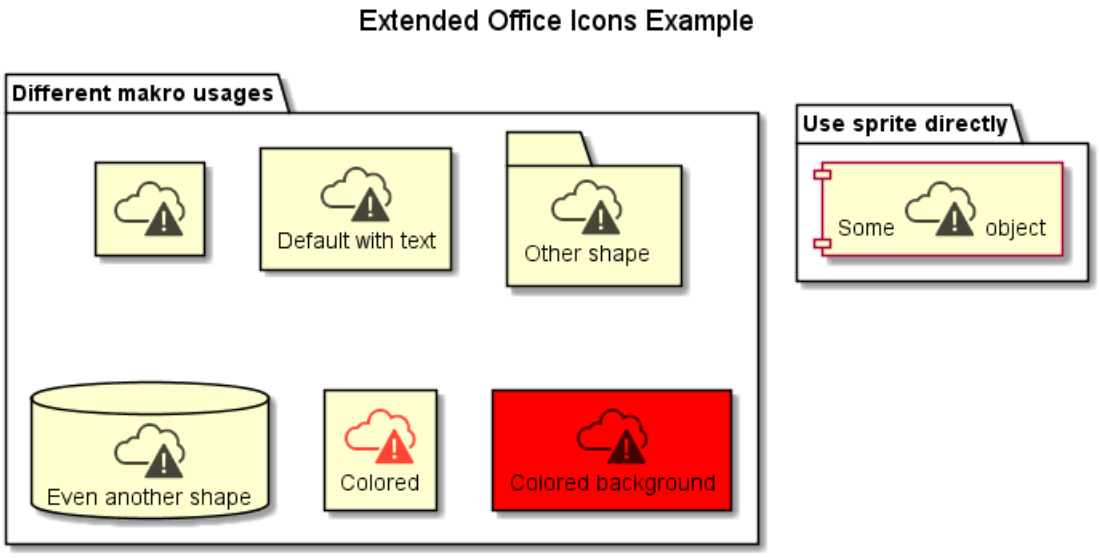
```
@startuml
!include <tupadr3/common>

!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 makro 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
```

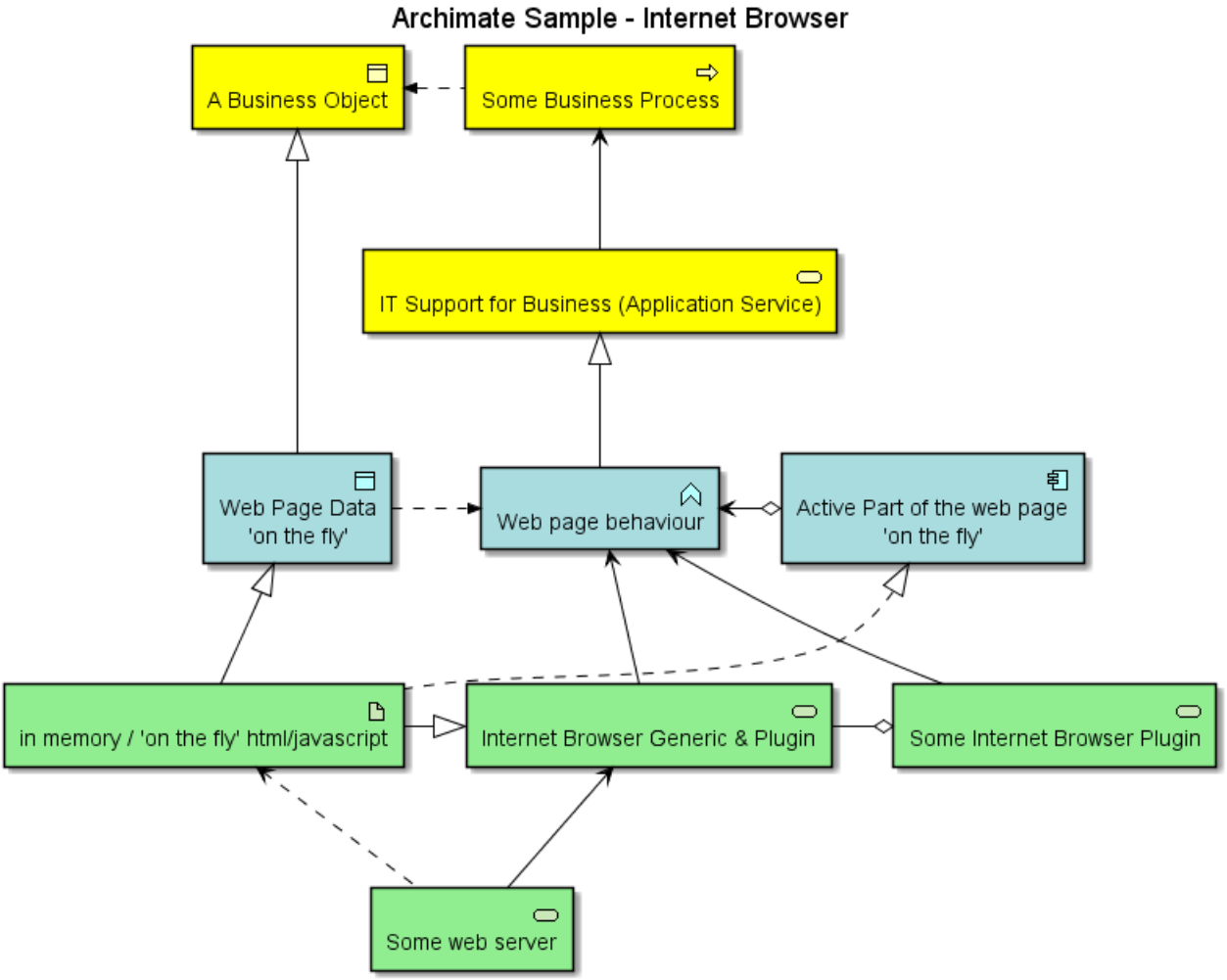


19.6 ArchiMate

https://github.com/ebbypeter/Archimate-PlantUML

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

```
@startuml Internet Browser Example
!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_Specilization_Up(webpageBehaviour, itSupportService, "")
Rel_Flow_Right(dataObject, webpageBehaviour, "")
Rel_Specilization_Up(dataObject, businessObject, "")
Rel_Assignment_Left(ActivePartWebPage, webpageBehaviour, "")
Rel_Specilization_Up(inMemoryItem, dataObject, "")
Rel_Realization_Up(inMemoryItem, ActivePartWebPage, "")
Rel_Specilization_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
```



19.7 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

azure
Version 0.0.1
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

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

1 Sequence Diagram **1**
1.1 Basic examples . . . 1
1.2 Declaring participant . . . 1
1.3 Use non-letters in participants . . . 3
1.4 Message to Self . . . 3
1.5 Change arrow style . . . 3
1.6 Change arrow color . . . 4
1.7 Message sequence numbering . . . 4
1.8 Page Title, Header and Footer . . . 6
1.9 Splitting diagrams . . . 7
1.10 Grouping message . . . 7
1.11 Notes on messages . . . 9
1.12 Some other notes . . . 9
1.13 Changing notes shape . . . 10
1.14 Creole and HTML . . . 10
1.15 Divider . . . 11
1.16 Reference . . . 12
1.17 Delay . . . 12
1.18 Space . . . 13
1.19 Lifeline Activation and Destruction . . . 13
1.20 Return . . . 15
1.21 Participant creation . . . 16
1.22 Shortcut syntax for activation, deactivation, creation . . . 16
1.23 Incoming and outgoing messages . . . 17
1.24 Stereotypes and Spots . . . 18
1.25 More information on titles . . . 19
1.26 Participants encompass . . . 20
1.27 Removing Foot Boxes . . . 21
1.28 Skinparam . . . 21
1.29 Changing padding . . . 23

2 Use Case Diagram **24**
2.1 Usecases . . . 24
2.2 Actors . . . 24
2.3 Usecases description . . . 25
2.4 Basic example . . . 25
2.5 Extension . . . 26
2.6 Using notes . . . 26
2.7 Stereotypes . . . 27
2.8 Changing arrows direction . . . 27
2.9 Splitting diagrams . . . 29
2.10 Left to right direction . . . 29
2.11 Skinparam . . . 30
2.12 Complete example . . . 31

3 Class Diagram **32**
3.1 Relations between classes . . . 32
3.2 Label on relations . . . 33
3.3 Adding methods . . . 33
3.4 Defining visibility . . . 34
3.5 Abstract and Static . . . 35
3.6 Advanced class body . . . 36
3.7 Notes and stereotypes . . . 36
3.8 More on notes . . . 37
3.9 Note on links . . . 38
3.10 Abstract class and interface . . . 38
3.11 Using non-letters . . . 39

3.12 Hide attributes, methods... . . . 40
3.13 Hide classes . . . 40
3.14 Use generics . . . 41
3.15 Specific Spot . . . 41
3.16 Packages . . . 42
3.17 Packages style . . . 42
3.18 Namespaces . . . 43
3.19 Automatic namespace creation . . . 44
3.20 Lollipop interface . . . 44
3.21 Changing arrows direction . . . 45
3.22 Association classes . . . 46
3.23 Skinparam . . . 47
3.24 Skinned Stereotypes . . . 47
3.25 Color gradient . . . 48
3.26 Help on layout . . . 49
3.27 Splitting large files . . . 49

4 Activity Diagram **51**
4.1 Simple Activity . . . 51
4.2 Label on arrows . . . 51
4.3 Changing arrow direction . . . 51
4.4 Branches . . . 52
4.5 More on Branches . . . 53
4.6 Synchronization . . . 54
4.7 Long activity description . . . 54
4.8 Notes . . . 55
4.9 Partition . . . 56
4.10 Skinparam . . . 57
4.11 Octagon . . . 57
4.12 Complete example . . . 58

5 Activity Diagram (beta) **60**
5.1 Simple Activity . . . 60
5.2 Start/Stop . . . 60
5.3 Conditional . . . 61
5.4 Repeat loop . . . 62
5.5 While loop . . . 62
5.6 Parallel processing . . . 63
5.7 Notes . . . 64
5.8 Colors . . . 64
5.9 Arrows . . . 65
5.10 Connector . . . 65
5.11 Grouping . . . 66
5.12 Swimlanes . . . 66
5.13 Detach . . . 67
5.14 SDL . . . 68
5.15 Complete example . . . 69

6 Component Diagram **71**
6.1 Components . . . 71
6.2 Interfaces . . . 71
6.3 Basic example . . . 72
6.4 Using notes . . . 72
6.5 Grouping Components . . . 72
6.6 Changing arrows direction . . . 74
6.7 Use UML2 notation . . . 75
6.8 Long description . . . 76
6.9 Individual colors . . . 76
6.10 Using Sprite in Stereotype . . . 76

6.11 Skinparam 77

7 State Diagram **79**
7.1 Simple State 79
7.2 Change state rendering 79
7.3 Composite state 80
7.4 Long name 80
7.5 Concurrent state 81
7.6 Arrow direction 82
7.7 Note 83
7.8 More in notes 84
7.9 Skinparam 84

8 Object Diagram **86**
8.1 Definition of objects 86
8.2 Relations between objects 86
8.3 Adding fields 86
8.4 Common features with class diagrams 87

9 Timing Diagram **88**
9.1 Declaring participant 88
9.2 Adding message 88
9.3 Relative time 89
9.4 Participant oriented 89
9.5 Setting scale 90
9.6 Initial state 90
9.7 Adding constraint 91
9.8 Adding texts 91

10 Gantt Diagram **93**
10.1 Declaring tasks 93
10.2 Adding constraints 93
10.3 Short names 93
10.4 Customize colors 94
10.5 Milestone 94
10.6 Calendar 94
10.7 Close day 94
10.8 Simplified task succession 95
10.9 Separator 95
10.10 Working with resources 95
10.11 Complex example 96

11 Maths **97**
11.1 Standalone diagram 97
11.2 How is this working ? 98

12 Common commands **99**
12.1 Comments 99
12.2 Footer and header 99
12.3 Zoom 99
12.4 Title 100
12.5 Caption 101
12.6 Legend the diagram 101

13 Salt (wireframe) **102**
13.1 Basic widgets 102
13.2 Using grid 102
13.3 Group box 103
13.4 Using separator 103

13.5 Tree widget . . . 104
13.6 Enclosing brackets . . . 104
13.7 Adding tabs . . . 104
13.8 Using menu . . . 105
13.9 Advanced table . . . 106
13.10 OpenIconic . . . 106
13.11 Include Salt . . . 107
13.12 Scroll Bars . . . 109

14 Creole **111**
14.1 Emphasized text . . . 111
14.2 List . . . 111
14.3 Escape character . . . 112
14.4 Horizontal lines . . . 112
14.5 Headings . . . 112
14.6 Legacy HTML . . . 113
14.7 Table . . . 114
14.8 Tree . . . 115
14.9 Special characters . . . 115
14.10 OpenIconic . . . 115

15 Defining and using sprites **117**
15.1 Encoding Sprite . . . 117
15.2 Importing Sprite . . . 118
15.3 Examples . . . 118

16 Skinparam command **119**
16.1 Usage . . . 119
16.2 Nested . . . 119
16.3 List . . . 119
16.4 Black and White . . . 119
16.5 Reverse colors . . . 120
16.6 Colors . . . 121
16.7 Font color, name and size . . . 121
16.8 Text Alignment . . . 122
16.9 Examples . . . 122

17 Preprocessing **126**
17.1 Including files . . . 126
17.2 Including URL . . . 126
17.3 Constant definition . . . 126
17.4 Macro definition . . . 127
17.5 Adding date and time . . . 128
17.6 Other special variables . . . 128
17.7 Macro on several lines . . . 128
17.8 Default values for macro parameters . . . 128
17.9 Conditions . . . 129
17.10 Building custom library . . . 130
17.11 Search path . . . 130
17.12 Advanced features . . . 130

18 Unicode **132**
18.1 Examples . . . 132
18.2 Charset . . . 134

19 Standard Library **135**
19.1 AWS library . . . 135
19.2 Cloud Insight . . . 135
19.3 Devicons and Font Awesome library . . . 136

19.4 Google Material Icons 137
19.5 Office 138
19.6 ArchiMate 140
19.7 Miscellaneous 141