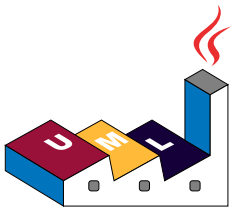
Diagramando UML con PlantUML



Guía de Referencia del Lenguaje PlantUML

(Version 1.2019.6)

PlantUML es un proyecto Open Source (código abierto) que permite escribir rápidamente:

- Diagramas de Secuencia
- Diagramas de Casos de uso
- Diagramas de Clases
- Diagramas de Actividades
- Diagramas de Componentes
- Diagramas de Estados
- Diagramas de Objetos
- Diagramas de Despliegue
- Timing diagram

Los siguientes diagramas no-UML también están soportados:

- Wireframe graphical interface
- Archimate diagram
- Specification and Description Language (SDL)
- Ditaa diagram
- Diagrama de Gantt
- MindMap diagram
- Work Breakdown Structure diagram
- Mathematic with AsciiMath or JLaTeXMath notation

Los diagramas son definidos usando un lenguaje simple e intuitivo.

1 Diagrama de Secuencia

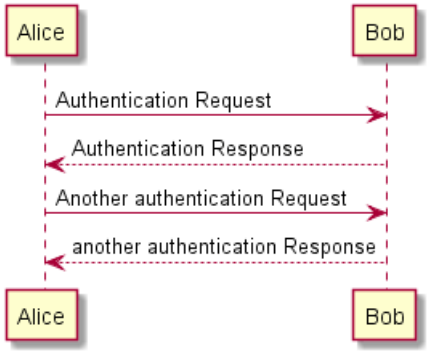
1.1 Ejemplo básico

La secuencia `->` es usada para dibujar un mensaje entre dos participantes. Los participantes tienen que ser declarados explícitamente.

Para definir una flecha punteada, se debe usar `-->`

También se puede usar `<-` y `<--`. No provoca cambios en el dibujo, pero puede mejorar la legibilidad. Tenga en cuenta que esto sólo es posible en diagramas de secuencia, las reglas son diferentes para otros diagramas.

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

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



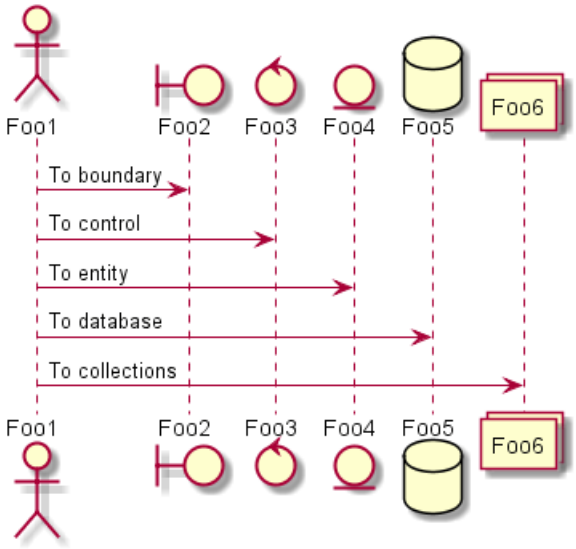
1.2 Declarando participantes

Es posible cambiar el orden de los participantes usando la palabra reservada `participant`.

También es posible el uso de otras palabras reservadas para declarar un participante:

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

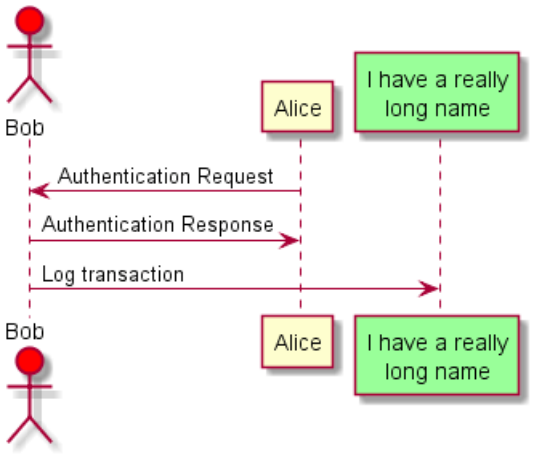
```
@startuml
actor Foo1
boundary Foo2
control Foo3
entity Foo4
database Foo5
collections Foo6
Foo1 -> Foo2 : To boundary
Foo1 -> Foo3 : To control
Foo1 -> Foo4 : To entity
Foo1 -> Foo5 : To database
Foo1 -> Foo6 : To collections

@enduml
```



Se puede renombrar un participante usando la palabra reservada `as`.

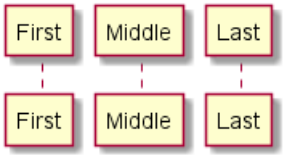
También es posible cambiar el color de fondo de los actores o participantes.

```
@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.

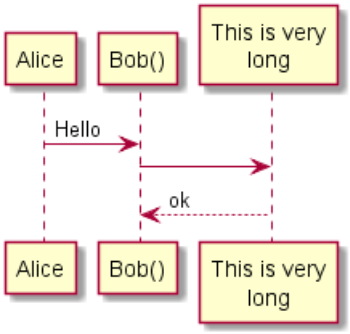
```
@startuml
participant Last order 30
participant Middle order 20
participant First order 10
@enduml
```



1.3 Sin usar letras en participantes

Puedes usar comillas para definir participantes. Y puedes usar la palabra reservada `as` para asignar un alias a esos participantes.

```
@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
```

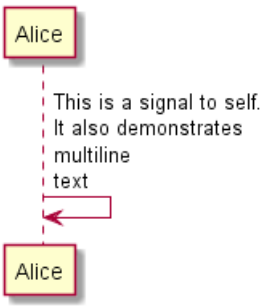


1.4 Auto-Mensaje

Un participante puede enviar mensajes asi mismo.

También es posible tener un mensaje multi-líneas usando `\n`.

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



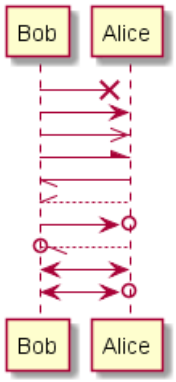
1.5 Cambiar estilo de la flecha

Puede cambiar el estilo de la flecha de diferentes formas:

- añade una x al final para indicar un mensage perdido
- utilice \ o / en lugar de < o > para tener solo la parte inferior o superior de la flecha
- repite la cabeza de la flecha (por ejemplo, >> o //) para tener un trazo más fino

- Utilice -- en lugar de - para obtener una flecha punteada.
- añade una "o" al final de la cabeza de una flecha
- utilice flechas bidireccionales <->

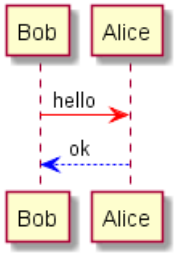
```
@startuml
Bob ->x Alice
Bob -> Alice
Bob ->> Alice
Bob -\ Alice
Bob \\- Alice
Bob //-- Alice

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

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



1.6 Cambiar el color de la flecha

Puede cambiar el color de flechas individuales usando la siguiente notación:

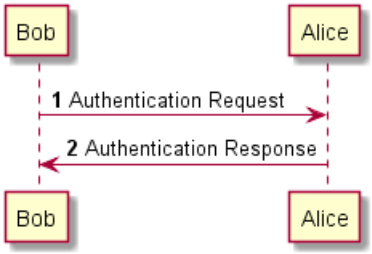
```
@startuml
Bob -[#red]> Alice : hello
Alice -[#0000FF]->Bob : ok
@enduml
```



1.7 Numeración de la secuencia de mensajes

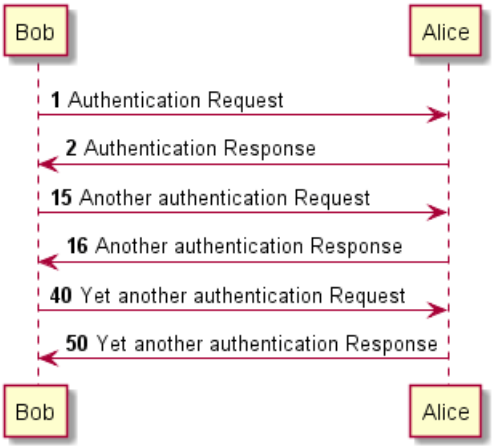
La palabra clave `autonumber` es usada para añadir automáticamente números a los mensajes.

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



Puedes especificar un número de comienzo con `autonumber` *`número inicial`* , y también un incremento con `autonumber` *`número inicial incremento`*.

```
@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
```



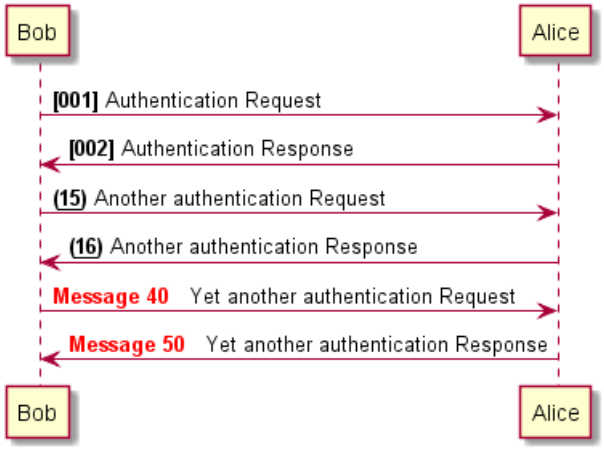
Puedes especificar un formato para su número usándolo entre comillas dobles.

El formateo se hace mediante la calse Java `DecimalFormat` (`0` denota un dígito, `#` denota un digito y cero si está ausente).

Puedes usar alguna etiqueta HTML en el formato.

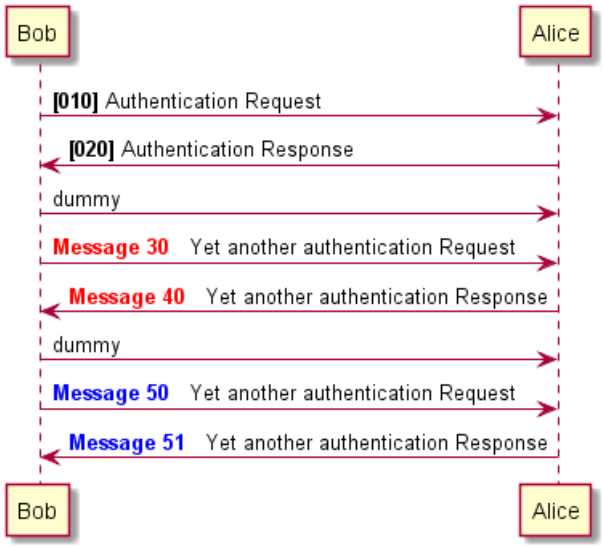
```
@startuml
autonumber "<b>[000]"
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response

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
```



También puedes usar `autonumber stop` y `autonumber resume` *`increment format`* para pausar y continuar la numeración automática, respectivamente.

```
@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
```

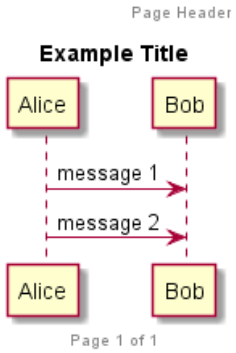


1.8 Page Title, Header and Footer

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
```



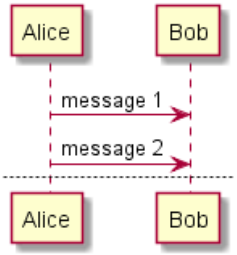
1.9 Dividiendo diagramas

La palabra reservada `newpage` es empleada para dividir un diagrama en varias imágenes.

Puedes colocar un título para la página nueva justo después de la palabra reservada `newpage` .

Esto es bastante práctico con *Word* para devolver diagramas grandes en varias páginas.

```
@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 Agrupando mensajes

Es posible agrupar mensajes usando las siguientes palabras reservadas:

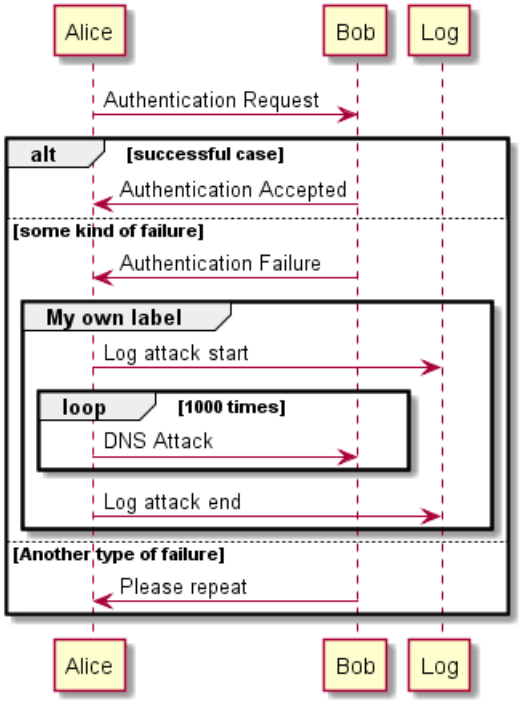
- `alt/else`
- `opt`
- `loop`
- `par`
- `break`
- `critical`
- `group`, seguida de un texto para mostrar

Es posible añadir un texto que será mostrado en el encabezado (excepto para `group`).

La palabra reservada `end` es usada para cerrar el grupo.

Tenga en cuenta que es posible anidar grupos.

```
@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
```

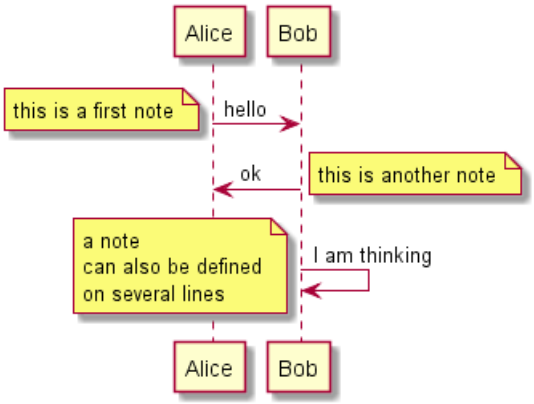


1.11 Notas en mensajes

Es posible colocar notas en mensajes usando las palabras reservadas `note left` o `note right` *inmediatamente después del mensaje*.

Puedes tener una nota multi-líneas usando la palabra reservada `end note` .

```
@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
```



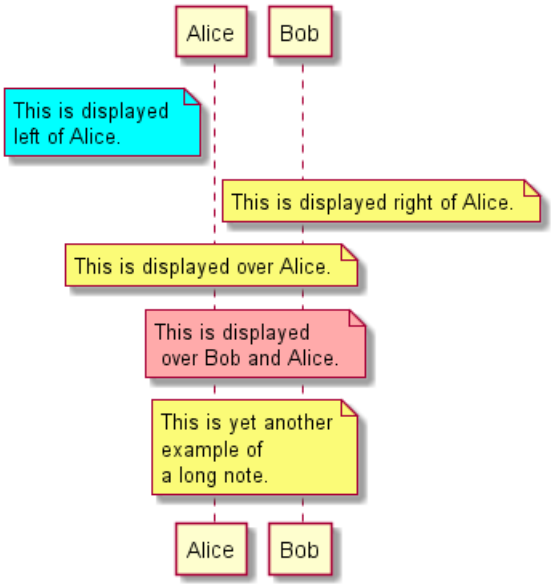
1.12 Algunas otras notas

También es posible colocar notas relativas al participante con las palabras reservadas <code>note left of</code> , `note right of` o `note over` .

Es posible resaltar una nota cambiando su color de fondo.

También puedes tener una nota multi-líneas usando la palabra reservada `end note` .

```
@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
```

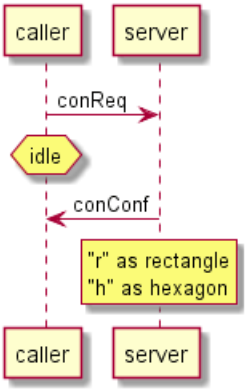


1.13 Cambiando el aspecto de las notas

Puedes usar las palabras reservadas `hnote` y `rnote` para cambiar el aspecto de las notas.

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

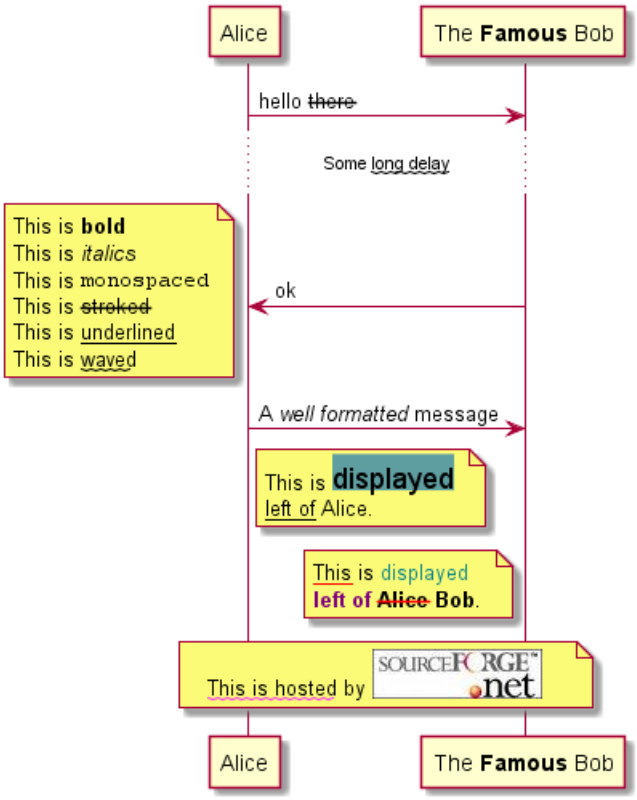
```
endrnote
@enduml
```



1.14 Creole y HTML

También es posible usar sintexis de WikiCreole:

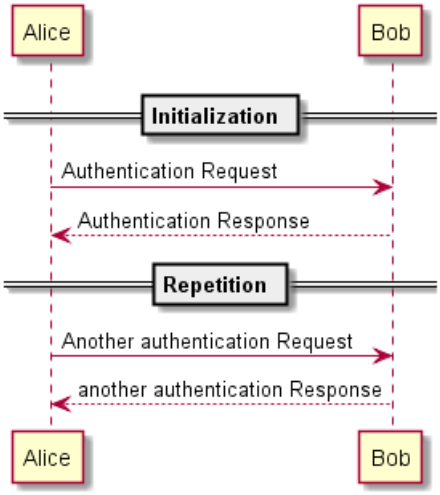
```
@startuml
participant Alice
participant "The **Famous** Bob" as Bob

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 Divisor

Si quieres, puedes dividir un diagrama usando el separador == para separar su diagrama en pasos lógicos.

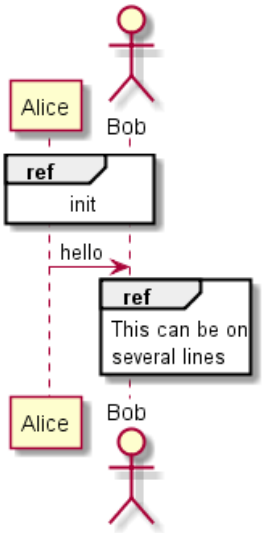
```
@startuml

== Initialization ==

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

== Repetition ==

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

@enduml
```



1.16 Referencia

Puedes referenciar en un diagrama utilizando la palabra clave `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
```

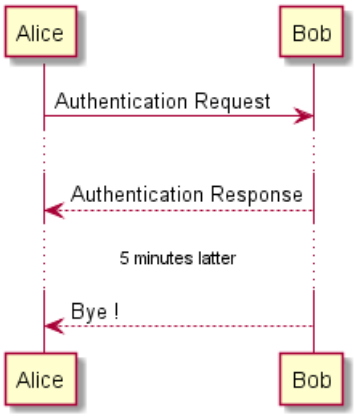


1.17 Retardo

Puedes usar `...` para indicar un retardo en el diagrama. Y también es posible colocar un mensaje con ese retardo.

```
@startuml
```

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

@enduml
```

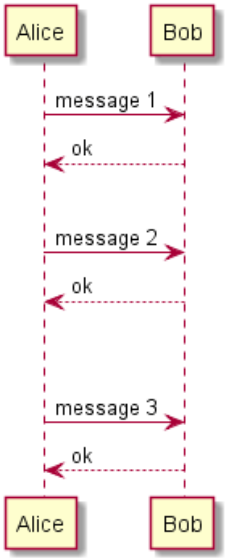


1.18 Espaciado

Puedes usar `|||` para indicar espaciado en el diagrama.

También es posible especificar un número de píxel para ser usado.

```
@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 Activación y Destrucción de la Línea de vida

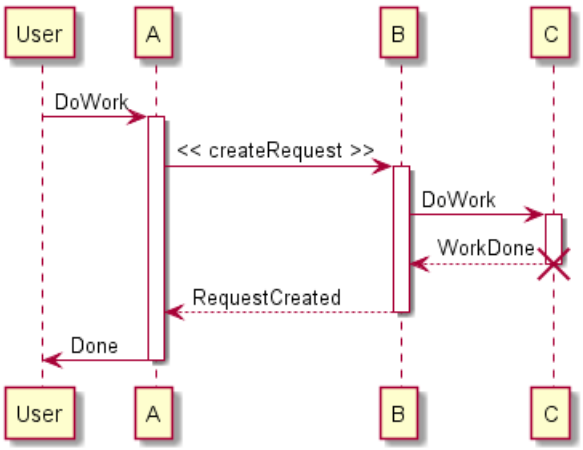
`activate` y `deactivate` son usados para denotar la activación de un participante.

Una vez que un participante es activado, su línea de vida aparece.

`activate` y `deactivate` aplica en el mensaje anterior.

`destroy` denota el final de la línea de vida de un participante.

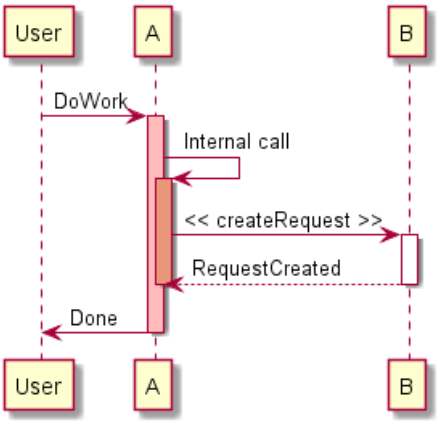
```
@startuml
participant User

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
```



Puede usarse anidamiento de líneas de vida, y es posible agregar un color a dicha línea de vida.

```
@startuml
participant User

User -> A: DoWork
activate A #FFBBBB

A -> A: Internal call
activate A #DarkSalmon

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

B --> A: RequestCreated
```

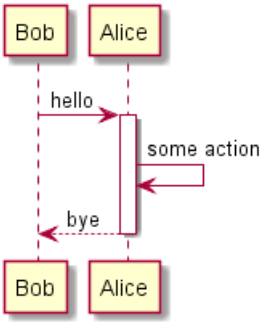
```
deactivate B
deactivate A
A -> User: Done
deactivate A

@enduml
```



1.20 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
```

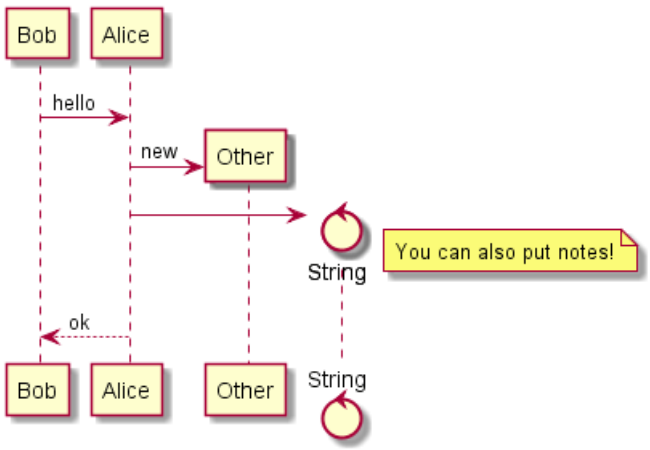


1.21 Creación de participante

Puedes usar la palabra reservada `create` justo antes de la primera recepción de un mensaje para recalcar el hecho de que ese mensaje se encuentra *creando* ese nuevo objeto.

```
@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
```

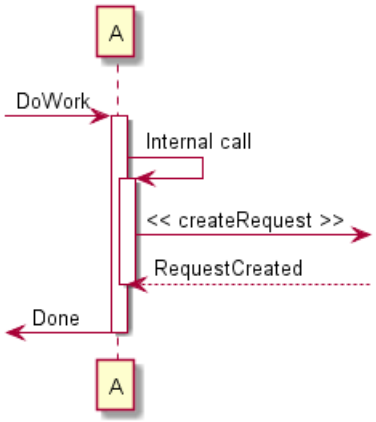


1.22 Mensajes entrantes y salientes

Puedes usar flechas entrantes y salientes si quieres centrarte en una parte del diagrama.

Utilice corchetes para denotar el lado izquierdo "[" o el lado derecho "]" del diagrama.

```
@startuml
[-> A: DoWork

activate A

A -> A: Internal call
activate A

A ->] : << createRequest >>

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



También puedes tener la siguiente sintaxis:

```
@startuml
[-> Bob
```

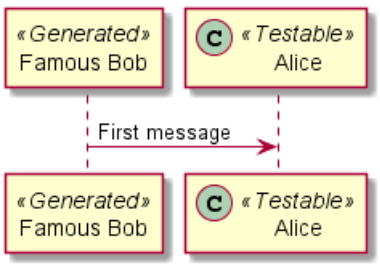
```
[o-> Bob
[o->o Bob
[x-> Bob

[<- Bob
[x<- Bob

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

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

1.23 Estereotipos y marcas

Es posible añadir estereotipos a participantes usando << y >>.

En el estereotipo, puedes añadir un carácter marcado en un círculo coloreado usando la sintaxis `(X,color)`.

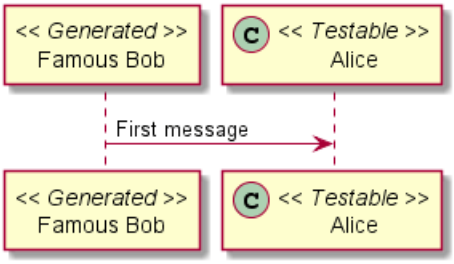
```
@startuml

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

Bob->Alice: First message

@enduml
```



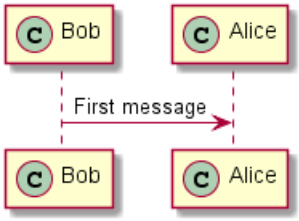
Por defecto, *guillemet* (comillas) son usadas para mostrar el estereotipo. Puedes cambiar este comportamiento usando skinparam `guillemet`:

```
@startuml

skinparam guillemet false
```

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

Bob->Alice: First message

@enduml
```



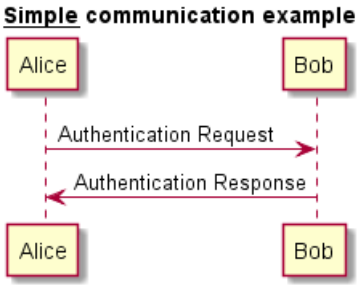

```
@startuml

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

Bob->Alice: First message

@enduml
```



1.24 Mayor información en los títulos

Puedes usar sintaxis de Creole en el título.

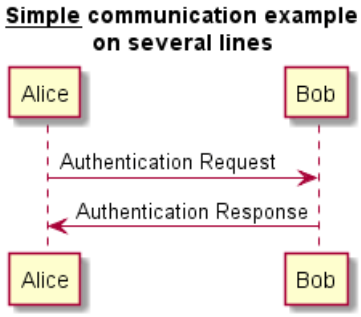
```
@startuml

title __Simple__ **communication** example

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

@enduml
```



Puedes añadir una nueva línea usando \n en la descripción del título.

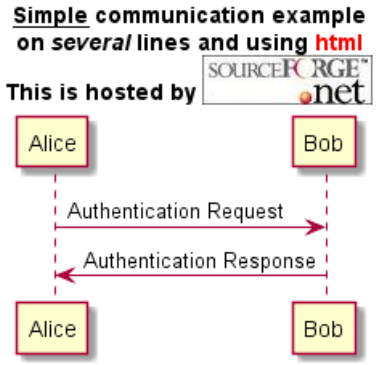
```
@startuml

title __Simple__ communication example\non several lines
```

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

@enduml
```



Además puedes definir un título en varias líneas usando las palabras reservadas `title` y `end title` .

```
@startuml

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

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

@enduml
```



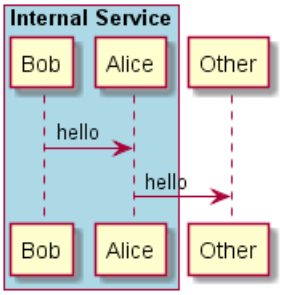
1.25 Entorno de participante

Es posible dibujar una caja alrededor de algunos participantes, usando los comandos `box` y `end box` .

Puedes añadir un título opcional o un color de fondo opcional, después de la palabra reservada `box` .

```
@startuml

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

Bob -> Alice : hello
```

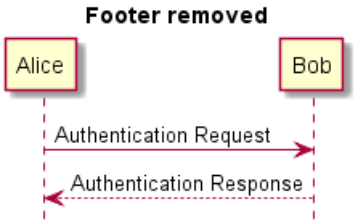
```
Alice -> Other : hello

@enduml
```



1.26 Removiendo pie de página

Puedes usar las palabras reservadas `hide footbox` para remover el pie de página del diagrama.

```
@startuml

hide footbox
title Footer removed

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

@enduml
```



1.27 Personalización (Skinparam)

Puedes usar el comando `skinparam` para cambiar los colores y las fuentes de los dibujos

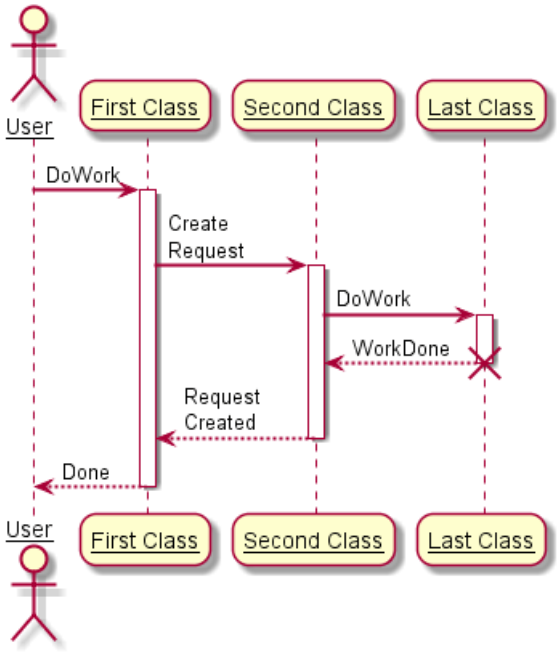
Puedes usar este comando:

- En la definición del diagrama, como cualquier otro comando,
- En un archivo incluido,
- En un archivo de configuración, proporcionado en la consola de comandos o en el ANT task.

También puedes cambiar otros parámetros de renderización, como se ve en los siguientes ejemplos

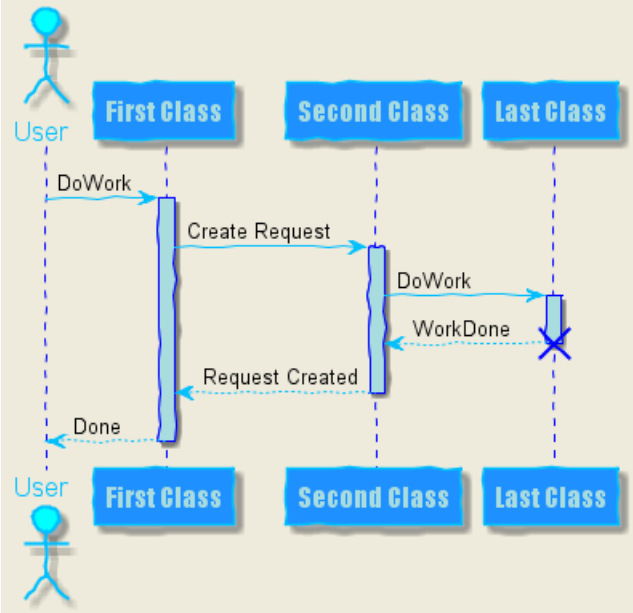
```
@startuml
skinparam sequenceArrowThickness 2
skinparam roundcorner 20
skinparam maxmessagesize 60
skinparam sequenceParticipant underline

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
```

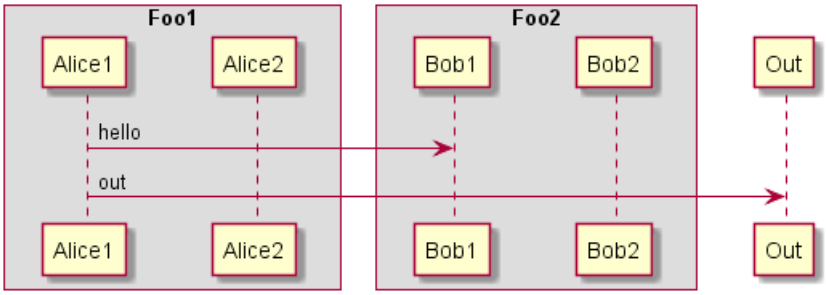


1.28 Cambiando el relleno

Es posible ajustar algunos parámetros de relleno

```
@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 Diagrama de Casos de Uso

Veamos algunos ejemplos:

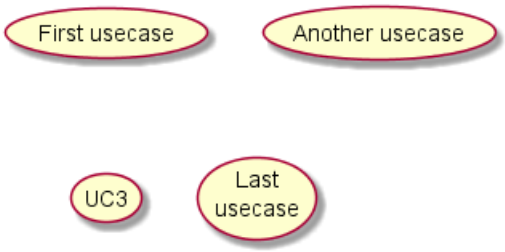
Puedes deshabilitar el sombreado usando el comando "skinparam shadowing false"

2.1 Casos de uso

Los casos de uso estan encerrados entre paréntesis (los paréntesis tienen un aspecto similar a un óvalo).

También puede usar la palabra `usecase` para crear un caso de uso. Ademas puede crear un alias, usando la palabra `as`. Este alias será usado mas adelante, cuando defina relaciones.

```
@startuml

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

@enduml
```



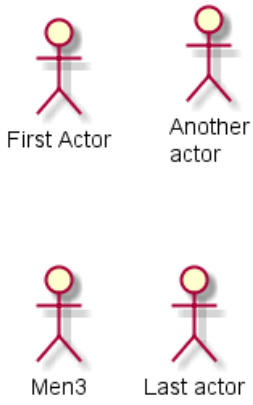
2.2 Actores

Los actores se encierran entre dos puntos.

También puedes usar la palabra reservada `actor` para definir un actor. Además puedes definir un alias, usando la palabra reservada `as`. Este alias será usado más adelante, cuando definamos relaciones.

Veremos más adelante que las declaraciones de los actores son opcionales.

```
@startuml

:First Actor:
:Another\nactor: as Men2
actor Men3
actor :Last actor: as Men4

@enduml
```

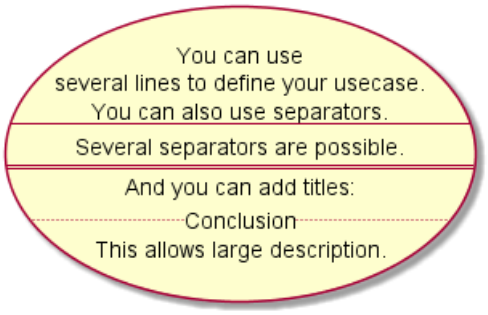


2.3 Descripción de Casos de uso

Si quiere realizar una descripción en varias líneas, puede usar citas (`" "`).

También puede usar los siguientes separadores: `--` `..` `==` `__`. Y puede introducir títulos dentro de los separadores.

```
@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 Ejemplo básico

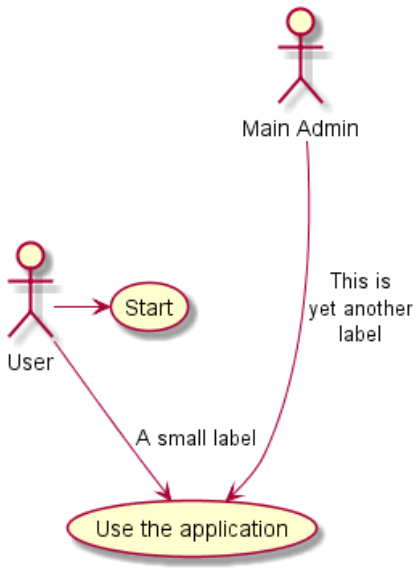
Para relacionar actores y casos de uso, la flecha `-->` es usada.

Cuanto más guiones `-` en la flecha, más larga será la misma. Puedes añadir una etiqueta en la flecha, añadiendo el carácter `:` en la definición de la flecha.

En este ejemplo, puedes ver que *User* no ha sido definido, y es usado como un actor.

```
@startuml

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

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

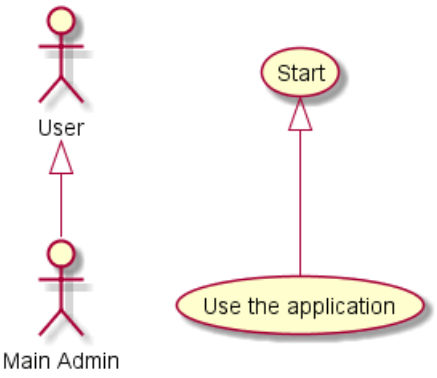
```
@enduml
```



2.5 Extensión

Si un actor/caso de uso extiende a otro, puedes usar el símbolo `<|--`.

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

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

@enduml
```



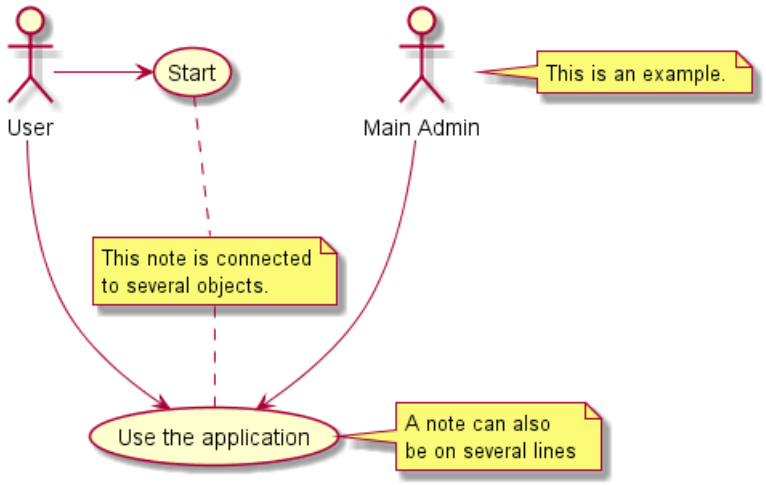
2.6 Usando notas

Puedes usar las palabras claves: `note left of` , `note right of` , `note top of` , `note bottom of`, para añadir notas relacionadas a un objeto en particular.

También se puede añadir un nota solitaria con la palabra clave `note`, y después realacionarla con otro objeto usando el símbolo `. . .`

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

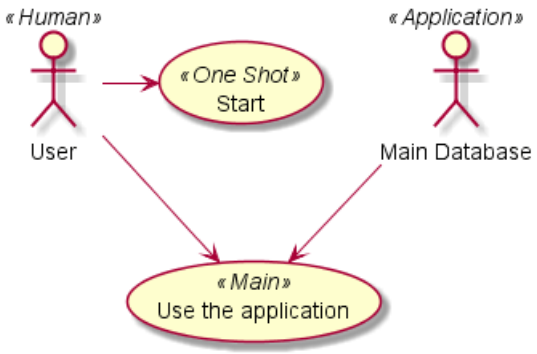
```
User -> (Start)
User --> (Use)

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 Estereotipos

Puedes añadir estereotipos mientras defines actores y casos de uso, usando << y >>.

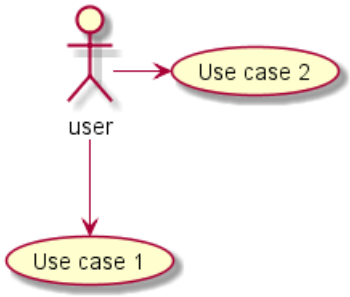
```
@startuml
User << Human >>
:Main Database: as MySql << Application >>
(Start) << One Shot >>
(Use the application) as (Use) << Main >>

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

MySql --> (Use)

@enduml
```



2.8 Cambiar dirección a las flechas

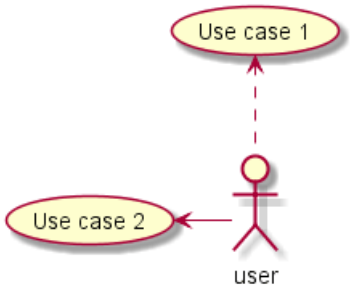
Por defecto, conexiones entre clases tiene dos guiones `--` y son verticalmente orientadas. Es posible usar una conexión horizontal, colocando un solo guión (o punto) de esta forma:

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



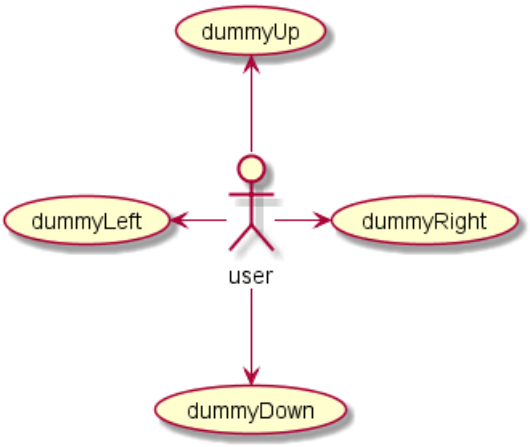
También puedes cambiar de dirección revirtiendo la conexión:

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



También es posible cambiar la dirección de una flecha añadiendo las palabras clave `left`, `right`, `up` or `down`, dentro de la misma:

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



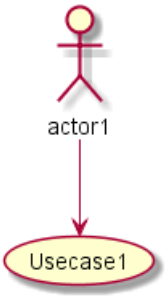
Puedes acortar la flecha usando sólo el primer carácter de la dirección (por ejemplo, `-d-` en lugar de `-down-`) o los primeros dos caracteres (`-do-`).

Por favor tenga en cuenta que no debería abusar de esta funcionalidad : *Graphviz* usualmente devuelve buenos resultados sin realizar muchos ajustes.

2.9 Dividiendo los diagramas

La palabra clave `newpage` divide su diagrama en varias páginas o imágenes.

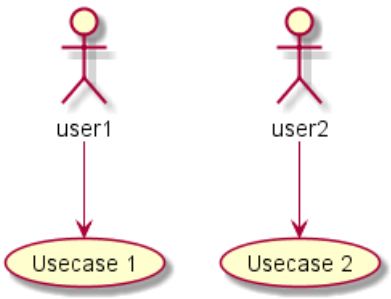
```
@startuml
:actor1: --> (Usecase1)
newpage
:actor2: --> (Usecase2)
@enduml
```



2.10 Dirección: de izquierda a derecha

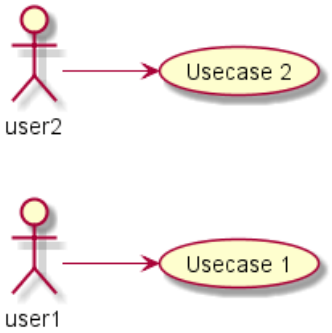
El comportamiento general cuando se construye un diagrama, es **top to bottom**.

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

@enduml
```



Puede cambiar a **left to right** usando el comando `left to right direction`. En ocaciones, el resultado es mejor con esta dirección.

```
@startuml

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

@enduml
```



2.11 Personalización (Skinparam)

Puedes usar el comando `skinparam` para cambiar los colores y las fuentes de los dibujos

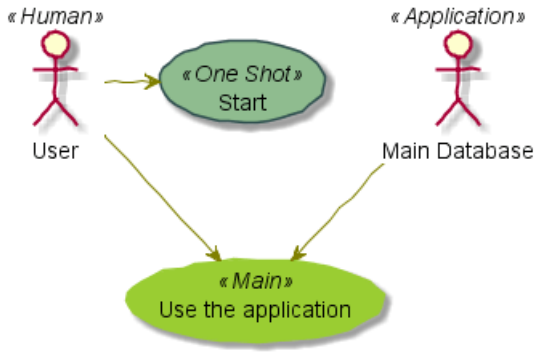
Puedes usar este comando:

- En la definición del diagrama, como cualquier otro comando,
- En un archivo incluido,
- En un archivo de configuración, proporcionado en la consola de comandos o en el ANT task.

Puedes definir colores y fuentes específicas para los actores y casos de uso estereotipados.

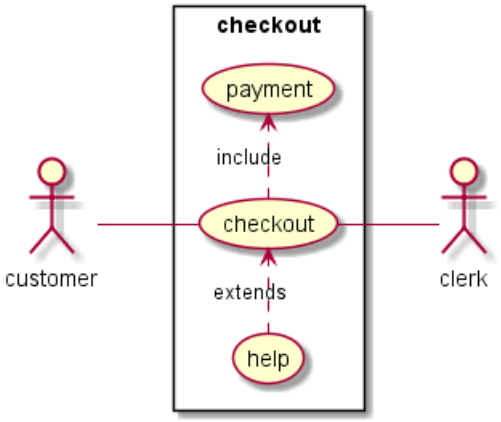
```
@startuml
skinparam handwritten true

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 Un ejemplo completo

```
@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 Diagrama de Clases

3.1 Relación entre clases

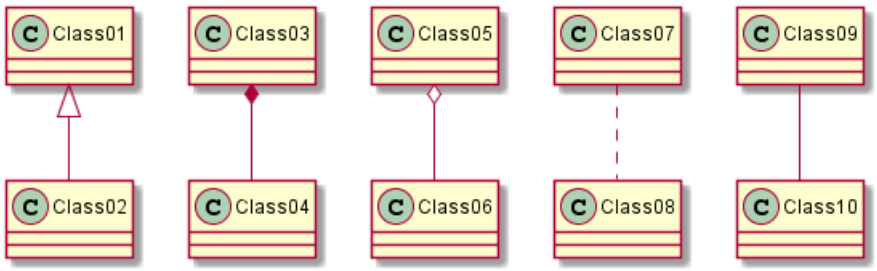
Las relaciones entre clases se definen usando los siguientes símbolos:

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

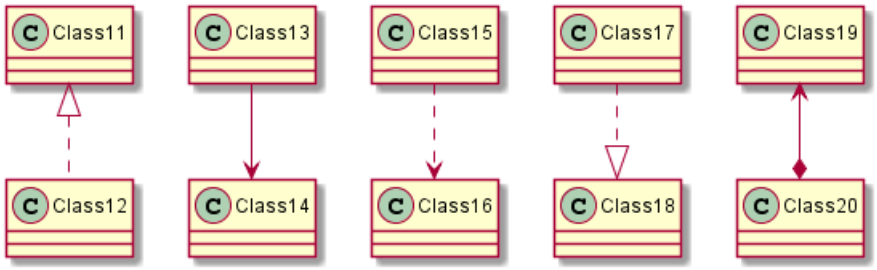
Es posible intercambiar `--` por `..` para tener lineas punteadas.

Sabiendo esas reglas, es posible sacar los siguientes dibujos:

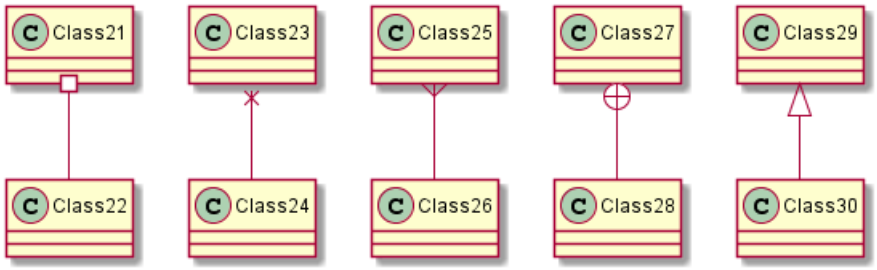
```
@startuml
Class01 <|-- Class02
Class03 *-- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10
@enduml
```




```
@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
```

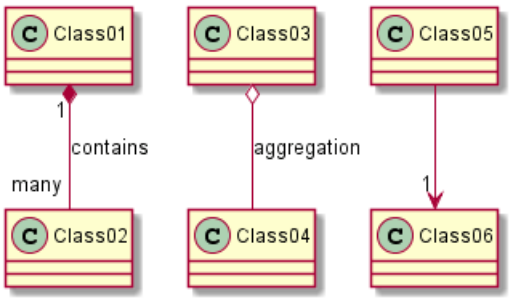


3.2 Etiquetas en las relaciones

Es posible añadir etiquetas en las relaciones, usando `:`, seguido del texto de la etiqueta.

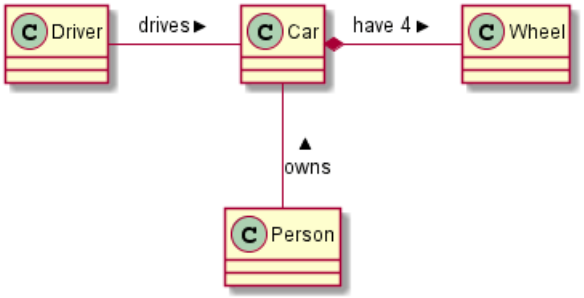
Para la cardinalidad, puede usar comillas dobles `""` en cada lado de la relación.

```
@startuml

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

Class03 o-- Class04 : aggregation

Class05 --> "1" Class06

@enduml
```



Se puede añadir una flecha extra apuntando a un objeto, mostrando que objeto actúa sobre el otro objeto, usando < o > al inicio o al final de la etiqueta.

```
@startuml
class Car

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

@enduml
```

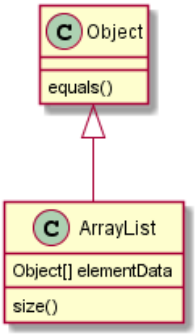


3.3 Añadiendo métodos

Para declarar las propiedades y métodos, se puede usar el símbolo : seguido del nombre de la propiedad o el método.

El sistema busca por paréntesis para elegir entre métodos y propiedades.

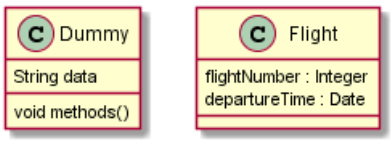
```
@startuml
Object <|-- ArrayList

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

@enduml
```



También es posible agrupar entre llaves {} todos las propiedades y métodos.

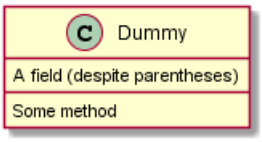
Tenga en cuenta que la sintexis es muy flexible acerca del orden tipo/nombre.

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

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



Puede usar los modificadores {field} y {method} para modificar el comportamiento por defecto del parse sobre los campos y métodos.

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

@enduml
```

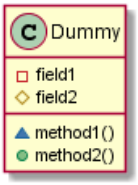


3.4 Definiendo la visibilidad

Cuando defines propiedades o métodos, puedes usar caracteres para establecer la visibilidad que les correspondan:

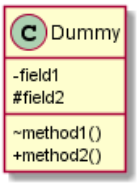
Character	Icon for field	Icon for method	Visibility
-			private
#			protected
~			package private
+			public

```
@startuml

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

@enduml
```



Puedes desactivar esta característica usando el comando `skinparam classAttributeIconSize 0` :

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

@enduml
```

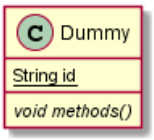


3.5 Abstracto y Estático

Puedes definir métodos o propiedades abstractas y estáticas usando los modificadores `{static}` o `{abstract}` .

Esos modificadores pueden ser usado al comienzo o al final de un línea. También puedes usar `{classifier}` en lugar de `{static}`.

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

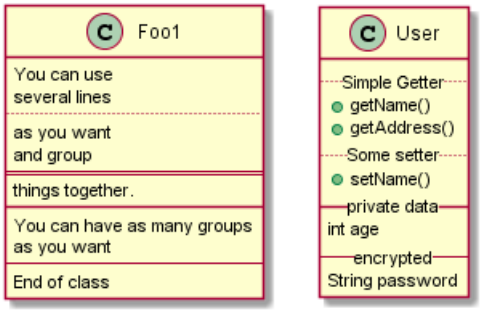


3.6 Cuerpo avanzado de las clases

Por defecto, las propiedades y los métodos son agrupados automáticamente por PlantUML. Puedes usar separadores para definir tu propia manera de ordenar las propiedades y los métodos. Son posibles los siguientes separadores: `--` `..` `==` `__`.

También puedes definir títulos dentro de los separadores:

```
@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 Notas y estereotipos

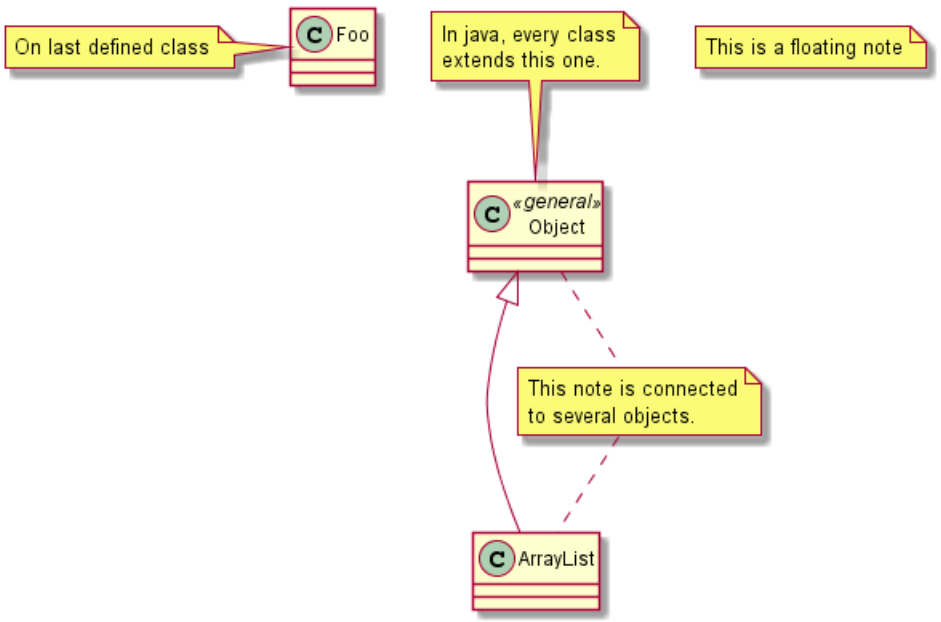
Los estereotipos son definidos con la palabra clave `class`, << and >>.

También puedes definir notas usando las palabras claves `note left of` , `note right of` , `note top of` , `note bottom of` .

Además puedes definir una nota en la última clase definida usando `note left`, `note right`, `note top`, `note bottom` .

Una nota también puede definirse solitariamente con la palabra clave `note`, y a continuación relacionarla con otro objeto usando el símbolo `...`

```
@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 Más acerca de notas

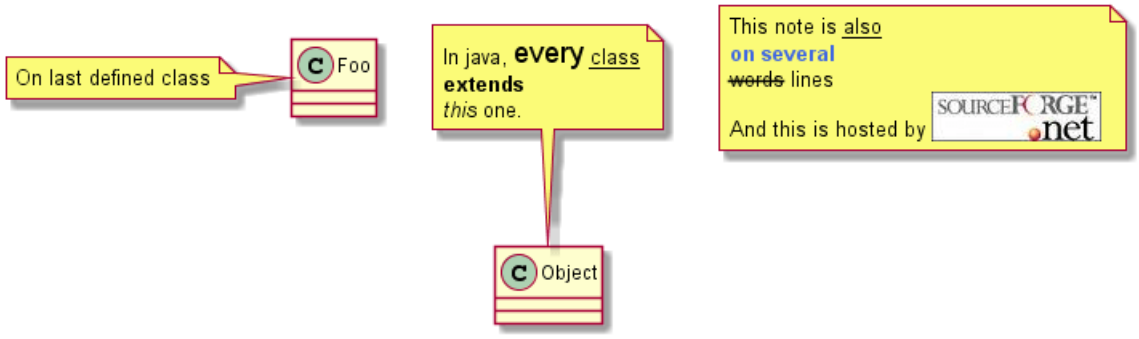
También es posible usar algunas etiquetas HTML como:

- `<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

También puedes tener una nota en varias líneas.

És possible definir una nota en la última clase definida usando `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
```



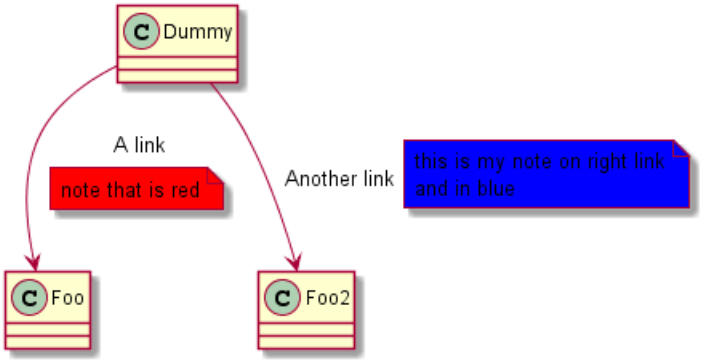
3.9 Notas en enlaces

Es posible añadir una nota en un enlace, justo después de la definición de dicho enlace, usando `note on link` .

También puedes usar `note left on link`, `note right on link`, `note top on link`, `note bottom on link` si quieres cambiar la posición de la nota, en relación a una etiqueta.

```
@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
```



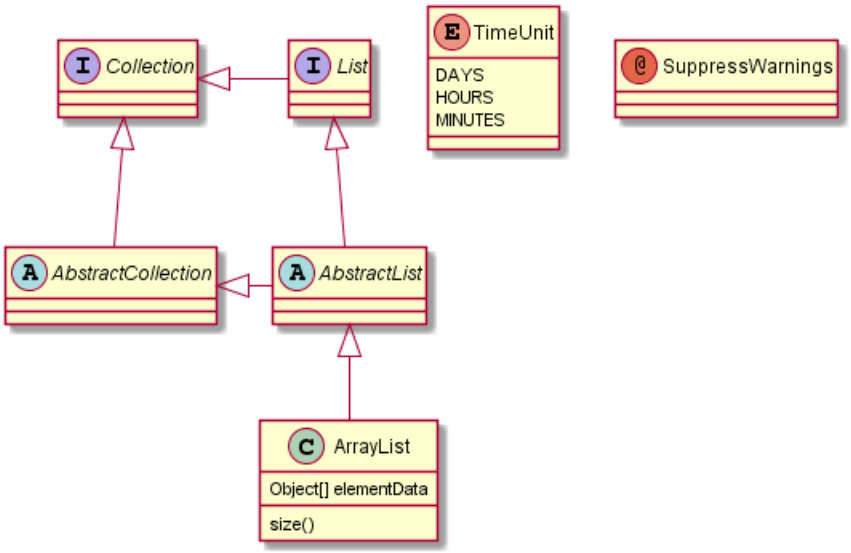
3.10 Clases abstractas e interfaces

Puedes declarar una clase como abstracta usando las palabras claves `abstract` or `abstract class` .

La clase será impresa en *italic* .

Puedes usar también las palabras claves `interface`, `annotation` and `enum` .

```
@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
```

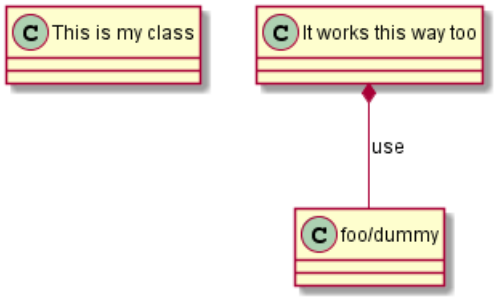


3.11 Sin usar letras

Si no desea usar letras en la visualización de la clase (o enum...), puede:

- Utilizar la palabra reservada `as` en la definición de la clase
- Colocar comillas `""` alrededor del nombre de la clase

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

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



3.12 Atributos, métodos... ocultos

Puede parametrizar la visualización de las clases usando el comando `hide/show` .

El comando básico es: `hide empty members`. Este comando ocultará atributos y métodos si están vacíos.

En lugar de `empty members` , puedes usar:

- `empty fields` o `empty attributes` para atributos vacíos.
- `empty methods` para métodos vacíos,
- `fields` o `attributes` que ocultará atributos, incluso si son descritos,
- `methods` que ocultará métodos, incluso si son descritos,
- `members` que ocultará atributos y métodos, incluso si son descritos.
- `circle` para el carácter encerrado en un círculo, en frente del nombre de clase.

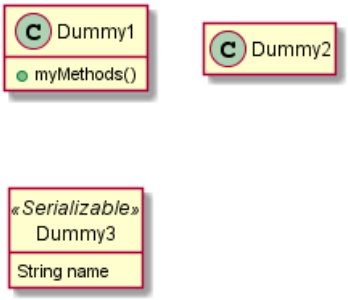
- `stereotype` para el estereotipo.

También puede proporcionar, justo después las palabras clave `hide` o `show`:

- `class` para todas las clases,
- `interface` para todas las interfaces,
- `enum` para todos los enums,
- `<<foo1>>` para clases que son estereotipadas con *foo1*,
- un nombre de clase existente.

Puedes usar varios comandos `show/hide` para definir reglas y excepciones.

```
@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 Clases ocultas

También puedes usar el comando `show/hide` para ocultar clases.

Esto puede llegar a ser útil si defines una archivo !included muy grande y si deseas ocultar algunas clases después de la inclusión de dicho archivo.

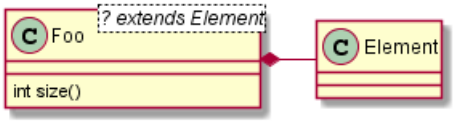
```
@startuml

class Foo1
class Foo2

Foo2 *-- Foo1

hide Foo2
```

```
@enduml
```

3.14 Uso de clases genéricas

También puedes usar los signos menor < y mayor > para definir el uso de clases genéricas.

```
@startuml

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

@enduml
```



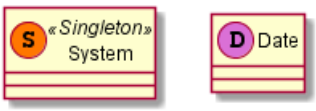
Es posible desactivar este dibujo con el comando `skinparam genericDisplay old`.

3.15 Círculo enmarcador específico

Usualmente, un carácter enmarcado en un círculo (C,I,E o A) es usado por clases, interfaces, enum y clases abstractas.

Pero puedes definir tu propio enmarcado para una clase cuando defines un estereotipo, añadiendo un carácter y un color, así como en el ejemplo:

```
@startuml

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



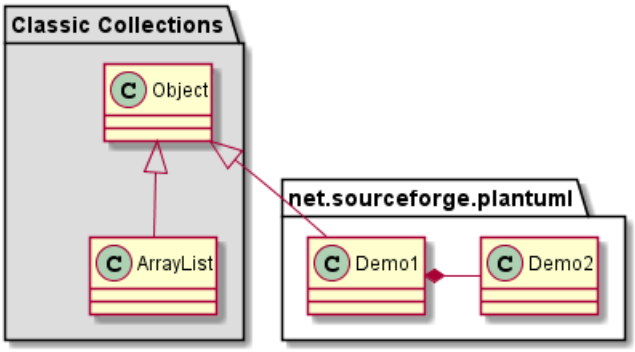
3.16 Paquetes

Puedes definir un paquete usando la palabra reservada `package`, y opcionalmente declarar un color de fondo para tu paquete (Usando el nombre o el código HTML del color).

Tenga en cuenta que las definiciones de paquetes pueden ser anidadas.

```
@startuml

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

```
}

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

@enduml
```



3.17 Estilos de paquetes

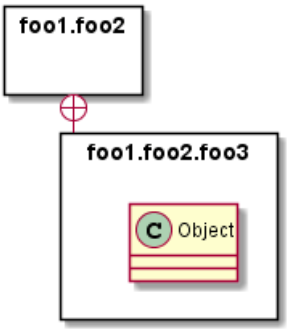
Hay diferentes estilos disponibles para paquetes.

Puedes especificarlos, ya sea configurando un estilo por defecto con el comando : `skinparam packageStyle` , o usando un estereotipo en el paquete.

```
@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
```

Puedes también definir enlaces entre paquetes, como en el siguiente ejemplo:

```
@startuml

skinparam packageStyle rectangle

package foo1.foo2 {
}

package foo1.foo2.foo3 {
  class Object
}

foo1.foo2 +-- foo1.foo2.foo3

@enduml
```



3.18 Espacios de nombre

En los paquetes, el nombre de una clase es el único identificador de esta clase. Quiere decir que no puedes tener dos clases con el mismo nombre en diferentes paquetes.

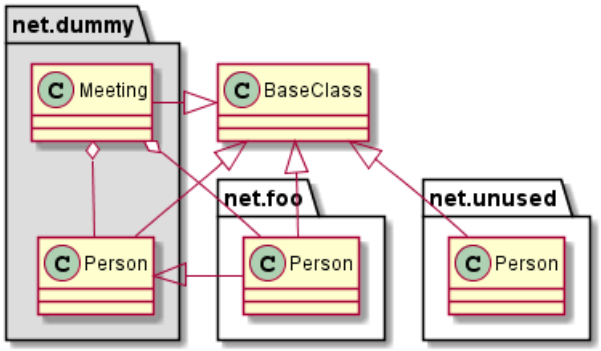
En este caso, deberías usar espacios de nombres en lugar de paquetes.

Puedes referir a clases de otros espacios de nombre describiendo su ruta completamente. A clases del espacio de nombre por defecto son descritas colocando un punto al inicio.

Tenga en cuenta que no tiene que especificar explícitamente un espacio de nombre : una clase altamente clasificada es automáticamente colocada en el espacio de nombre correcto.

```
@startuml

class BaseClass

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

.BaseClass <|- Meeting
}

namespace net.foo {
```

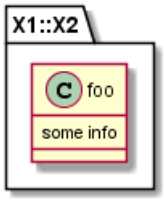
```
  net.dummy.Person  <|- Person
  .BaseClass <|-- Person

  net.dummy.Meeting o-- Person
}

BaseClass <|-- net.unused.Person

@enduml
```



3.19 Creación automática del espacio de nombre

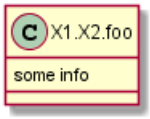
Puedes definir otro separador (otro además del punto) usando el comando : `set namespaceSeparator ???`.

```
@startuml

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

@enduml
```



Puedes deshabilitar la creación automática de paquetes usando el comando `set namespaceSeparator none`.

```
@startuml

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

@enduml
```

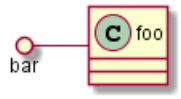


3.20 Interface Lollipop

También puedes definir interfaces lollipops en clases, usando la siguiente sintaxsis:

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

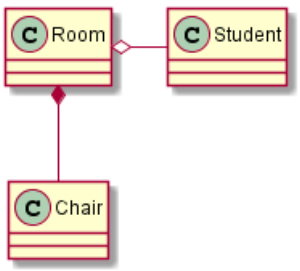
```
@startuml
class foo
bar ()- foo
@enduml
```



3.21 Cambiando la dirección de las flechas

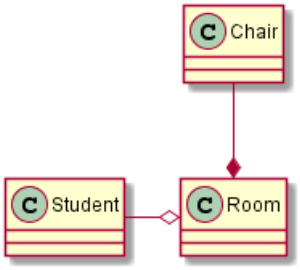
Por defecto, enlaces entre clases tienen dos guiones `--` y son verticalemnte orientados. Es posible usar un enlace horizontal colocando un solo guión (o punto), así:

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



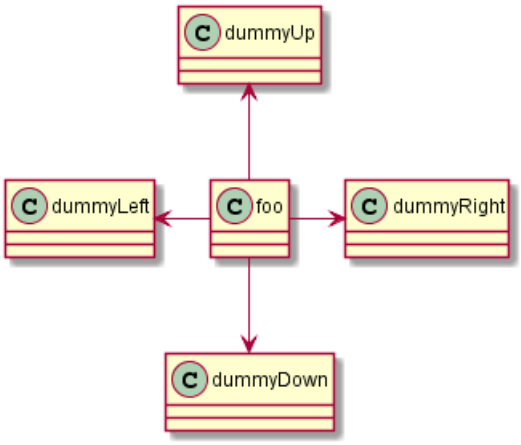
También puedes cambiar las direcciones revirtiendo el enlace:

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



También es posible cambiar la dirección de la flecha añadiendo las palabras claves `left`, `right`, `up` or `down` dentro de la flecha:

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



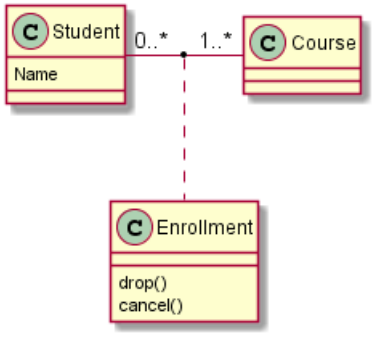
Puedes acortar la flecha usando el primer carácter de la dirección (por ejemplo, `-d-` en lugar de `-down-`) o los primeros dos caracteres.

Por favor tenga en cuenta que no debería abusar de esta funcionalidad : *Graphviz* usualmente otorga buenos resultados sin necesidad de ajustar.

3.22 Asociación de clases

Puedes definir *association class* después de que una relación haya sido establecida entre dos clases, como en este ejemplo:

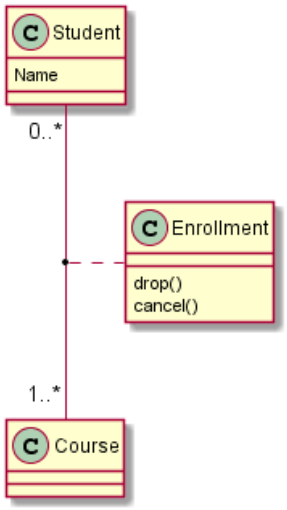
```
@startuml
class Student {
  Name
}
Student "0..*" - "1..*" Course
(Student, Course) .. Enrollment

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



Puedes definirla en otra dirección:

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

class Enrollment {
```

```
  drop()
  cancel()
}
@enduml
```



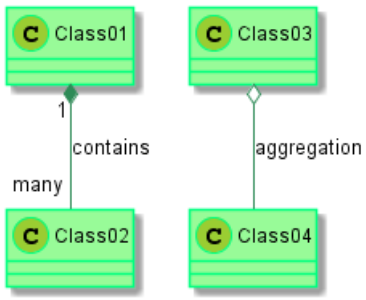
3.23 Personalización (Skinparam)

Puedes usar el comando `skinparam` para cambiar los colores y fuentes en el diagrama.

Puedes usar este comando:

- En la definición del diagrama, como cualquier otro comando,
- En un archivo incluido,
- En un archivo de configuración, proporcionado en la consola de comandos o en el ANT task.

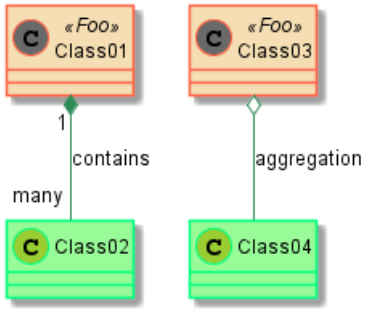
```
@startuml

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

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

Class03 o-- Class04 : aggregation

@enduml
```



3.24 Estereotipos personalizados

Puedes definir colores y fuentes específicas para clases esterotipadas.

```
@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 Degrado de colores

Es posible declarar colores individuales para clases o notas usando la notación #.

Puedes usar el nombre estándar del color o el código RGB.

También puedes usar degradación de color en el fondo, con la siguiente sintaxis: dos nombres de colores separados por cualquier de los siguientes:

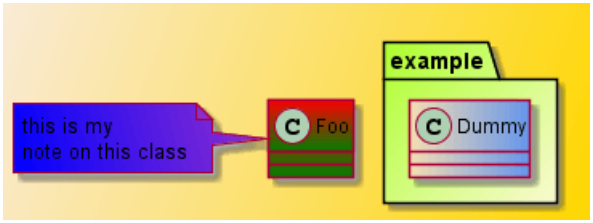
- |,
- /,
- \,
- o -

dependiendo de la dirección del degradado.

Por ejemplo, podrías tener:

```
@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
```



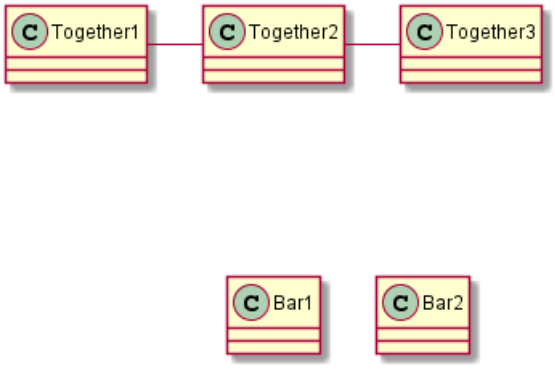
3.26 Ayudar en el diseño

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 Dividiendo archivos grandes

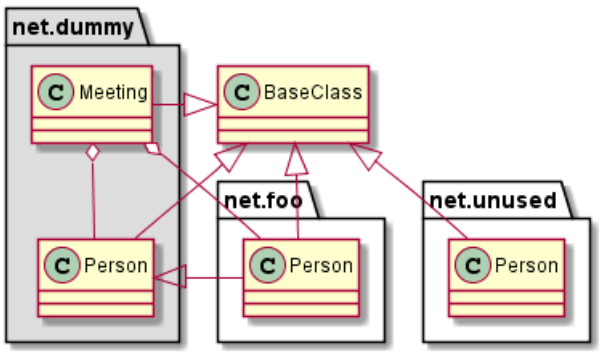
A veces, puedes obtener imágenes bastante grandes.

Puedes usar el comando `page (hpages)x(vpages)` para dividir la imágen generada, en varias imágenes:

`hpages` es el número que indica la cantidad de páginas horizontales, y `vpages` es el número que indica la cantidad de páginas verticales.

También puede utilizar algunos ajustes específicos skinparam poner fronteras en las páginas splitted (ver ejemplo).

```
@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 Diagrama de Actividades

4.1 Actividades simples

Puedes usar `(*)` para el punto de inicio y de final en un diagrama de actividades.

En algunos casos, quizás quieras usar `(*top)` para forzar a que el punto de inicio se ubique en la parte superior del diagrama.

Utilice `-->` para las flechas.

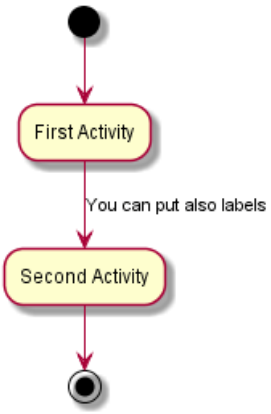
```
@startuml

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

@enduml
```

4.2 Etiquetas en las flechas

Por defecto, una flecha comienza en la última actividad usada.

Puedes colocar una etiqueta sobre una flecha usando corchetes, `[]` , justo después de la definición de la flecha.

```
@startuml

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

@enduml
```



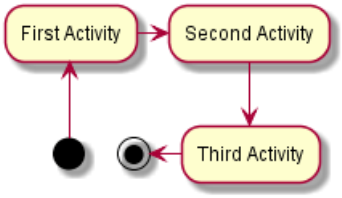
4.3 Cambiando la dirección de la flecha

Puedes usar `->` para flechas horizontales. Es posible formzar la dirección de una flecha usando la siguiente sintaxis:

- `-down->` (default arrow)

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

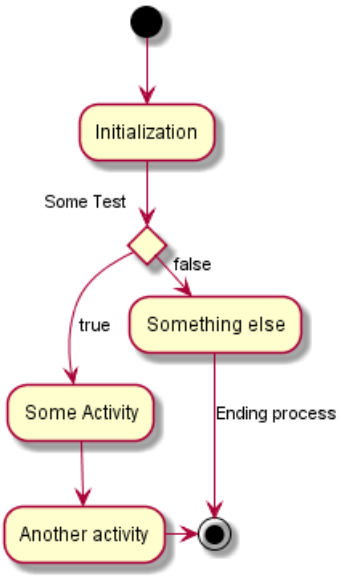
```
@startuml

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

@enduml
```



4.4 Ramas (bifurcaciones)

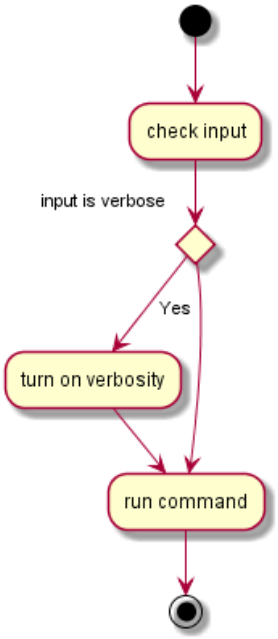
Puedes usar las palabras reservadas `if/then/else` para definir ramas o caminos alternos.

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

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

@enduml
```



Desafortunadamente, a veces tendrás que repetir la misma actividad en el texto del diagrama:

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



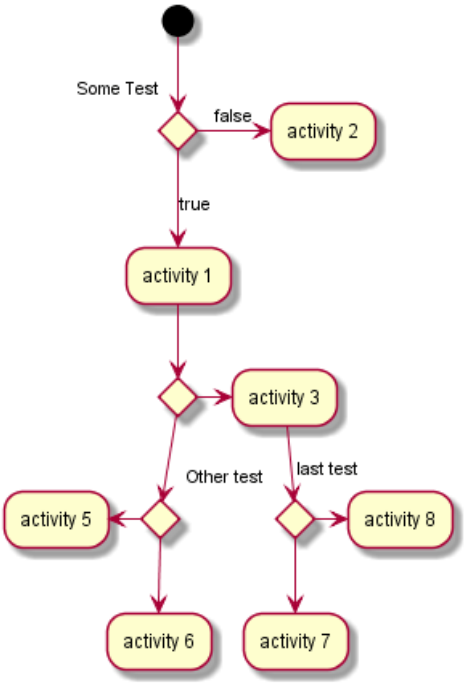
4.5 Más acerca de las Ramas

Por defecto, una rama es conectada con la última actividad definida, pero es posible sobrescribir esto y definir un enlace con la palabra reservada `if` .

También es posible anidar ramas.

```
@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"
```

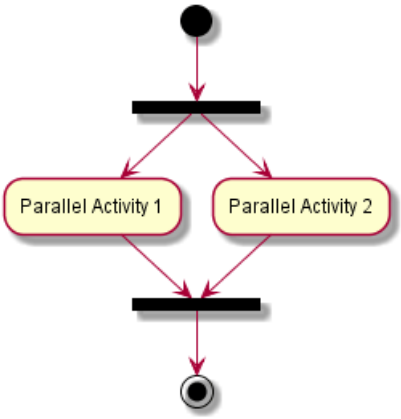
```
endif

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

@enduml
```



4.6 Sincronización

Puedes usar `===` `code` `===` para mostrar barras de sincronización.

```
@startuml

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

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

--> (*)

@enduml
```

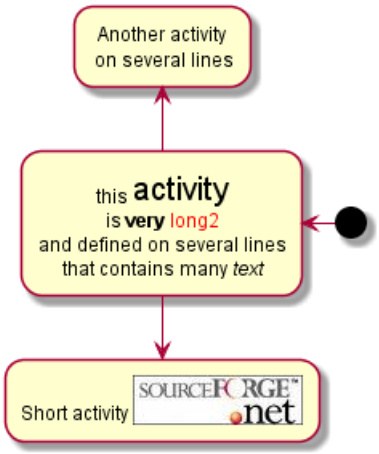


4.7 Descripción de actividades de gran contenido

Cuando declaras actividades, puedes abarcar la descripción del texto, en varias líneas. También puedes añadir \n en la descripción.

También puedes colocar una pequeña cantidad de código en la actividad, con la palabra reservada as . Este código puede usarse más adelante en la descripción del diagrama.

```
@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 Notas

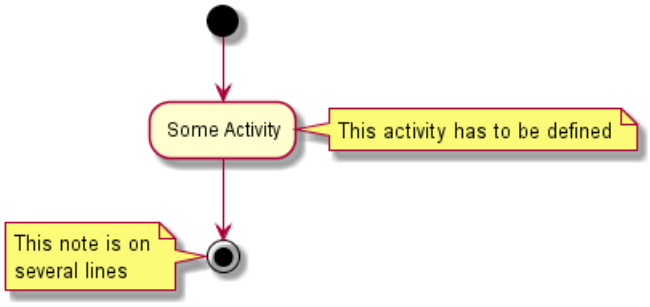
Puedes añadir notas sobre la actividad usando los comandos `note left`, `note right`, `note top` or `note bottom`, justo después de la descripción de la actividad a la quieres añadirle la nota.

Si quieres colocar una nota sobre el punto de inicio, define la nota al comienzo de la descripción del diagrama.

También puedes tener una nota de varias líneas, usando la palabra reservada `endnote` .

```
@startuml
```

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

@enduml
```



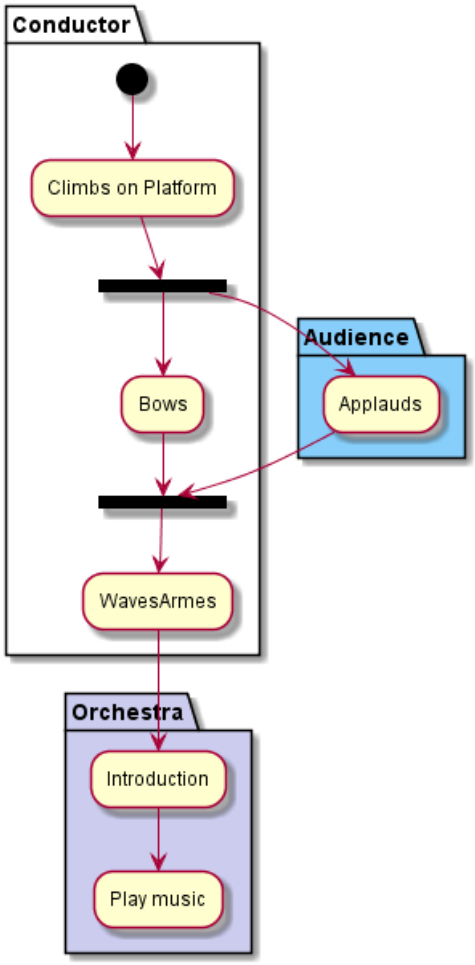
4.9 Partición

Puedes definir una partición usando la palabra reservada `partition` y opcionalmente declarar un color de fondo para tu partición (Usando el nombre o código HTML del color)

Cuando declaras actividades, éstas son automáticamente colocadas en la última partición usada.

Puedes cerrar la partición usando una llave de cierre `}`.

```
@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 Personalización (Skinparam)

Puedes usar el comando `skinparam` para cambiar las fuentes y colores en el diagrama.

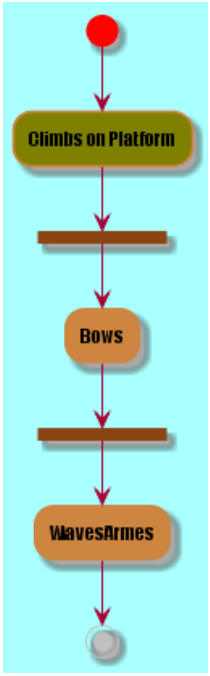
Puedes usar este comando :

- En la definición del diagrama, como cualquier otro comando,
- En un archivo incluido,
- En un archivo de configuración, proporcionado en la consola de comandos o en el ANT task.

Puedes definir colores y fuentes específicas para actividades estereotipadas.

```
@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 Octágono

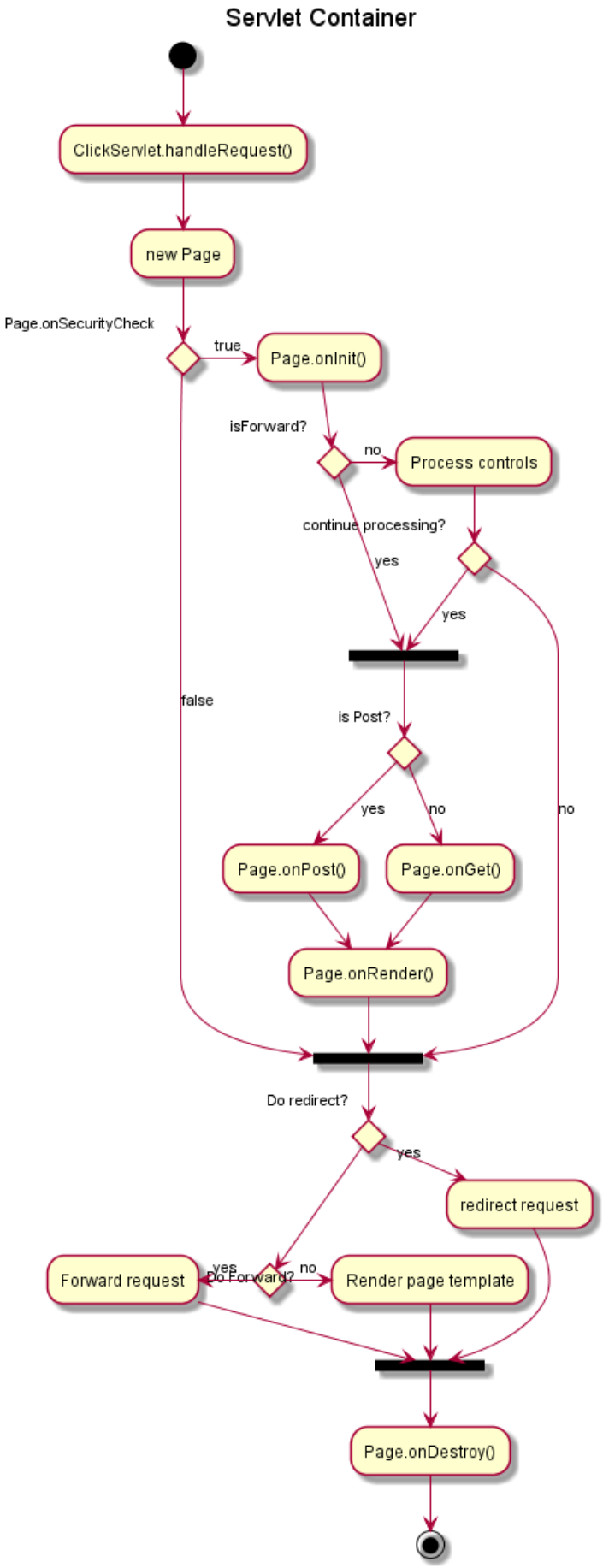
Puedes cambiar la forma de las actividades a un octágono usando el comando `skinparam activityShape octagon`.

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

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

@enduml
```

4.12 Un ejemplo completo

```
@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 Diagrama de Actividades (beta)

La actual sintaxis para los diagramas de actividades tiene varias limitaciones e inconvenientes (por ejemplo, es difícil de mantener).

Entonces, una implementación y sintaxis nueva propuesta como **versión beta** , es ofrecida a los usuarios (empezando con V7947), sólo así podremos definir un mejor formato y sintaxis.

Otra ventaja de esta nueva implementación es que acaba con la necesidad de tener Graphviz instalado (como en los diagramas de secuencia).

La nueva sintaxis reemplazará la anterior. Sin embargo, por razones de compatibilidad, la sintaxis vieja seguirá siendo reconocida, para asegurarse la *retrocompatibilidad* .

Los usuarios están siendo motivados para migrarse a la nueva sintaxis.

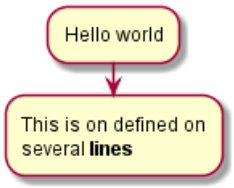
5.1 Una Actividad simple

Las etiquetas de las actividades inician con un dos puntos (`:`) y terminan con un punto y coma (`;`).

Se puede aplicar formato a un texto usando sintaxis de WikiCreole.

Están implícitamente enlazados en el orden de su definición.

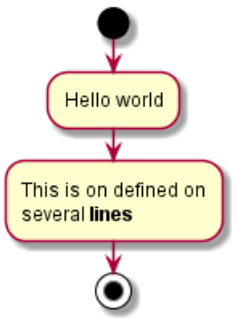
```
@startuml
:Hello world;
:This is on defined on
several **lines**;
@enduml
```



5.2 Start/Stop

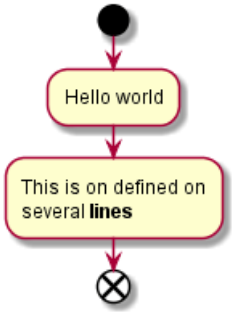
Puedes usar las palabras reservadas `start` y `stop` para denotar el comienzo y el final del diagrama.

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



También puedes usar la palabra reservada `end`.

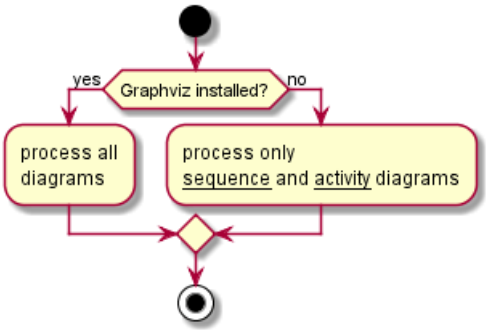
```
@startuml
start
:Hello world;
:This is on defined on
several **lines**;
end
@enduml
```



5.3 Condicionales

Puedes usar las palabras reservadas `if`, `then` y `else` para colocar condiciones en tus diagramas. Las etiquetas pueden ser proporcionadas usando paréntesis.

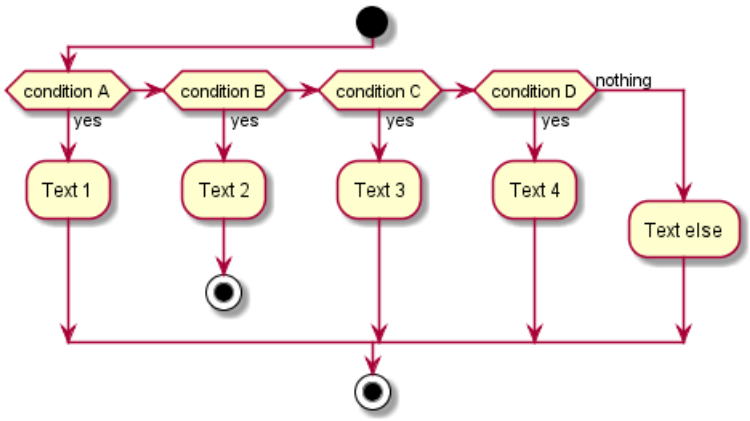
```
@startuml

start

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

stop

@enduml
```



Puedes usar la palabra reservada `elseif` para tener varias condiciones :

```
@startuml
start
if (condition A) then (yes)
  :Text 1;
elseif (condition B) then (yes)
  :Text 2;
  stop
```

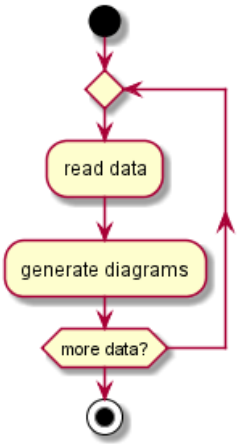
```
elseif (condition C) then (yes)
  :Text 3;
elseif (condition D) then (yes)
  :Text 4;
else (nothing)
  :Text else;
endif
stop
@enduml
```



5.4 El ciclo Repeat

Puedes usar las palabras reservadas `repeat` y `repeatwhile` para colocar bucles.

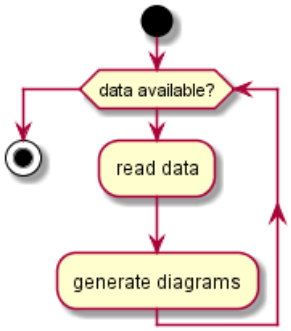
```
@startuml

start

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

stop

@enduml
```



5.5 El ciclo While

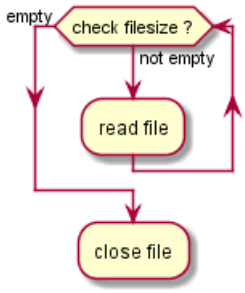
Puedes usar las palabras reservadas `while` y `end while` para un ciclo repetitivo.

```
@startuml

start

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

stop

@enduml
```



Es posible proporcionar una etiqueta después de la palabra reservada `endwhile`, o usar la palabra reservada `is` .

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

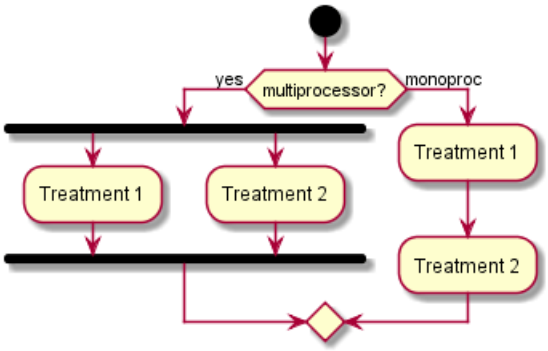


5.6 Procesamiento paralelo

Puedes usar las palabras reservadas `fork`, `fork again` y `end fork` para denotar procesamientos paralelos.

```
@startuml

start

if (multiprocessor?) then (yes)
  fork
:Treatment 1;
  fork again
```

```
:Treatment 2;
  end fork
else (monoproc)
  :Treatment 1;
  :Treatment 2;
endif

@enduml
```

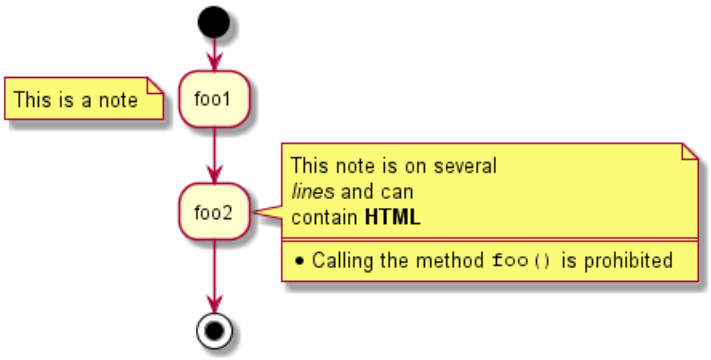


5.7 Notas

Se puede aplicar formato a un texto usando sintaxis de WikiCreole.

Una nota puede ser flotante, usando la palabra clave `floating`.

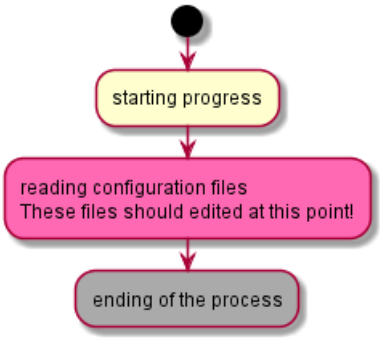
```
@startuml

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 Colores

Puedes especificar colores en algunas actividades.

```
@startuml

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

@enduml
```

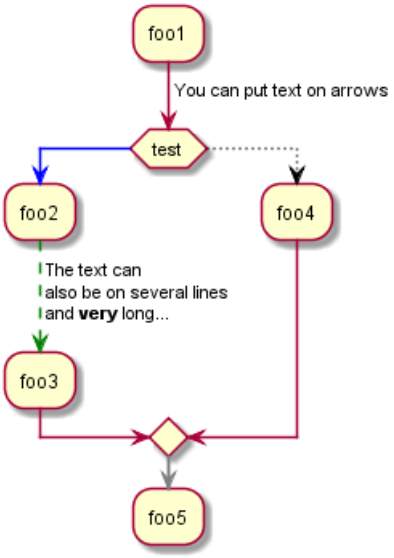


5.9 Flechas

Usando la notación ->, puedes añadir texto a una flecha y cambiar su color.

También es posible tener flechas punteadas, en linea discontinua, en negrita u ocultas.

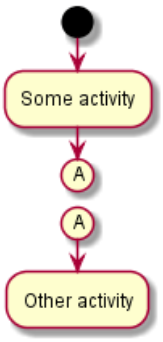
```
@startuml
:foo1;
-> You can put text on arrows;
if (test) then
  -[#blue]->
  :foo2;
  -[#green,dashed]-> The text can
  also be on several lines
  and **very** long...;
  :foo3;
else
  -[#black,dotted]->
  :foo4;
endif
-[#gray,bold]->
:foo5;
@enduml
```



5.10 Connector

You can use parentheses to denote connector.

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

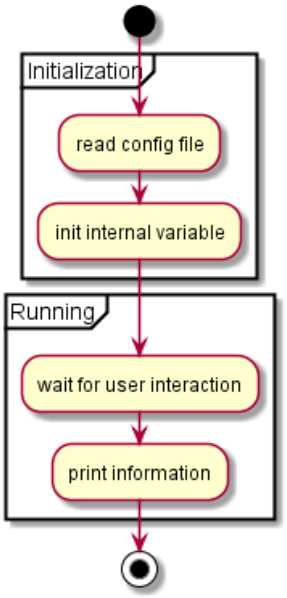


5.11 Agrupación

Puedes agrupar actividades definiendo particiones:

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

```
stop
@enduml
```

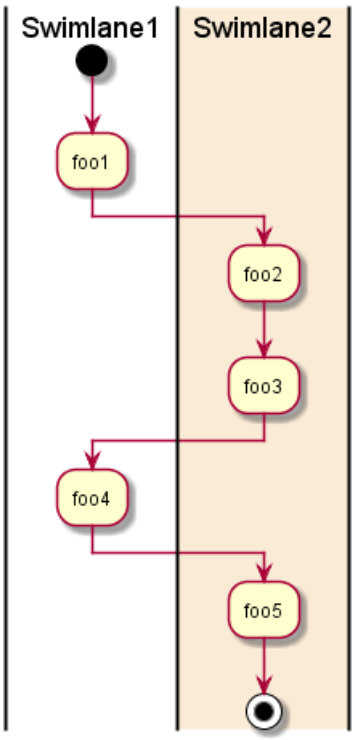


5.12 Carriles

Usando la tecla pipe |, puedes definir carriles.

También es posible cambiar el color de los carriles.

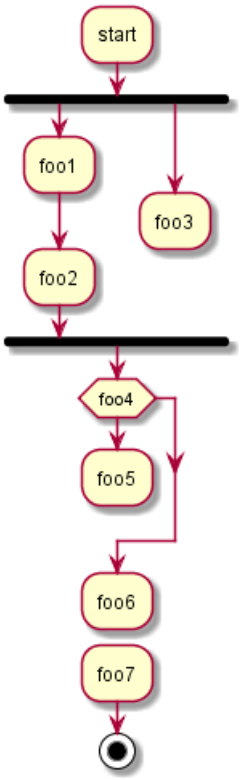
```
@startuml
|Swimlane1|
start
:foo1;
|#AntiqueWhite|Swimlane2|
:foo2;
:foo3;
|Swimlane1|
:foo4;
|Swimlane2|
:foo5;
stop
@enduml
```



5.13 Desacoplar y remover

Es posible remover una flecha usando la palabra reservada `detach` .

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



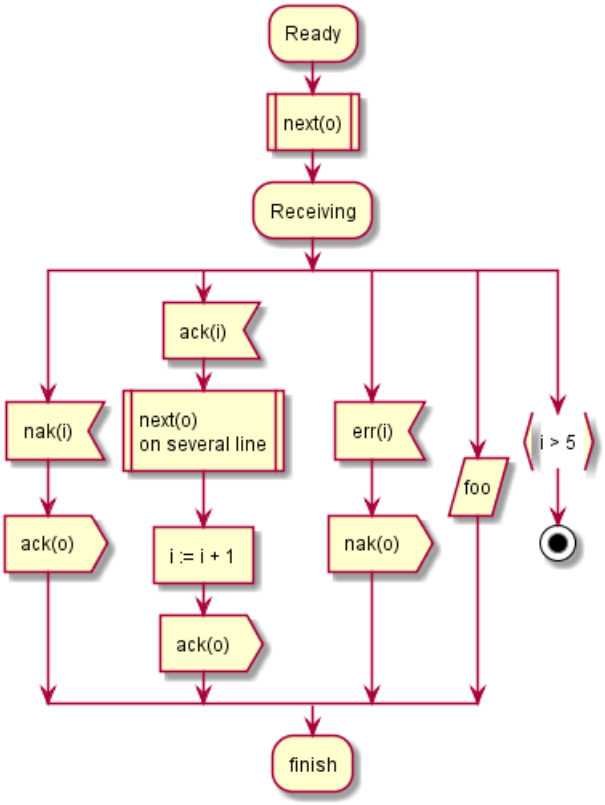
5.14 Otras formas de representación de actividades

Cambiando el sepador final, ; , puedes configurar diferentes representaciones para una actividad:

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

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

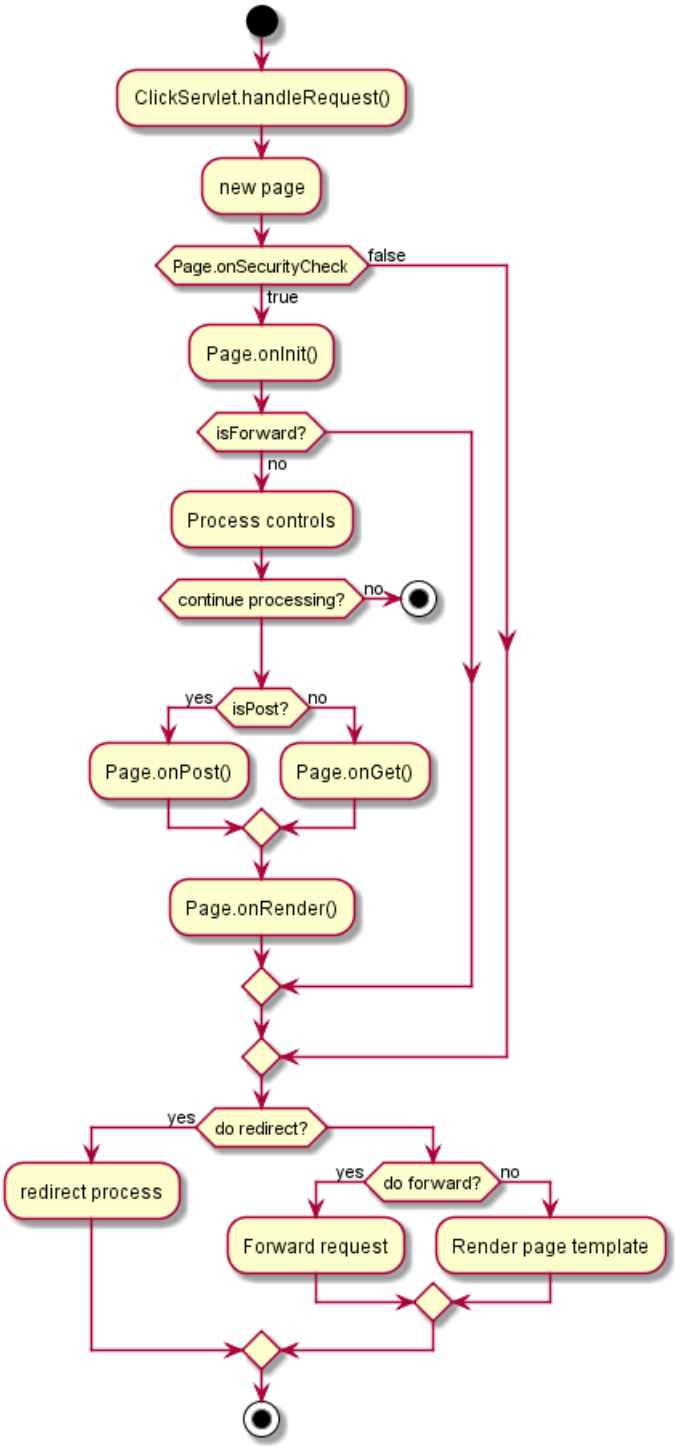
```
stop
end split
:finish;
@enduml
```



5.15 Un ejemplo completo

```
@startuml

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

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

if (do redirect?) then (yes)
```

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

stop

@enduml
```



6 Diagrama de Componentes

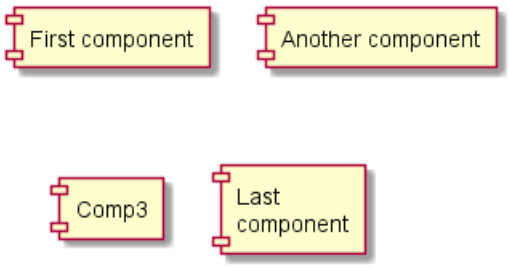
Veamos algunos ejemplos :

6.1 Componentes

Los componentes deberían ser encerrados entre corchetes `[]`.

También puedes usar la palabra reservada `component` para definir un componente. Y puedes definir un alias, usando la palabra reservada `as`. Este alias será usado más adelante, cuando definamos relaciones.

```
@startuml

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

@enduml
```



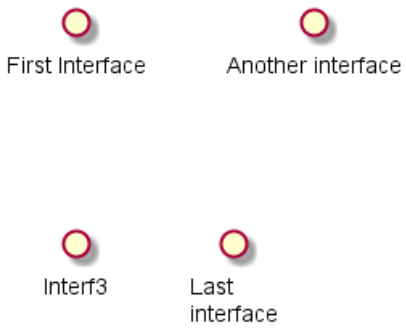
6.2 Interfaces

Puedes definir una interfaz usando el símbolo `()` (porque esto luce como un circulo).

Puedes usar también la palabra reservada `interface` para definir una interfaz. Y puedes definir un alias, usando la palabra reservada `as`. Este alias sera usado luego, definiendo las relaciones.

Nosotros veremos mas adelante que la definición de interfaz es opcional.

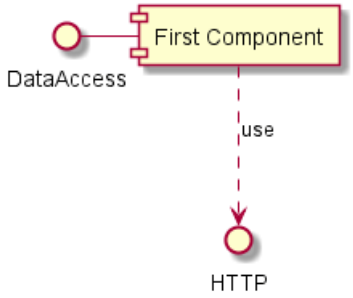
```
@startuml

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

@enduml
```



6.3 Ejemplos basicos

Los links entre los elementos se hacen usando la combinación de símbolos de linea de puntos (. .), linea recta (--), y flechas (-->)

```
@startuml

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

@enduml
```

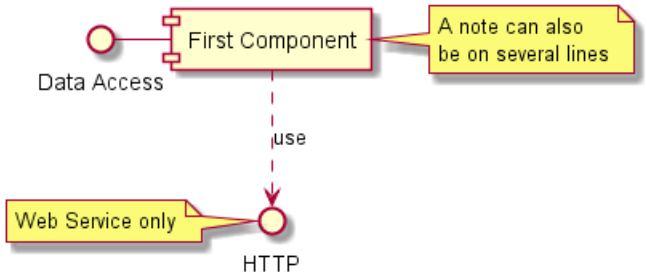


6.4 Usando notas

Puedes usar el `note left of` , `note right of` , `note top of` , `note bottom of` Las palabras reservadas para definir notas relacionadas a un objeto simple.

Una nota puede ser definida sola usando la plabra reservada `note`, luego linkea a otro objeto usando el símbolo `...`

```
@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
```

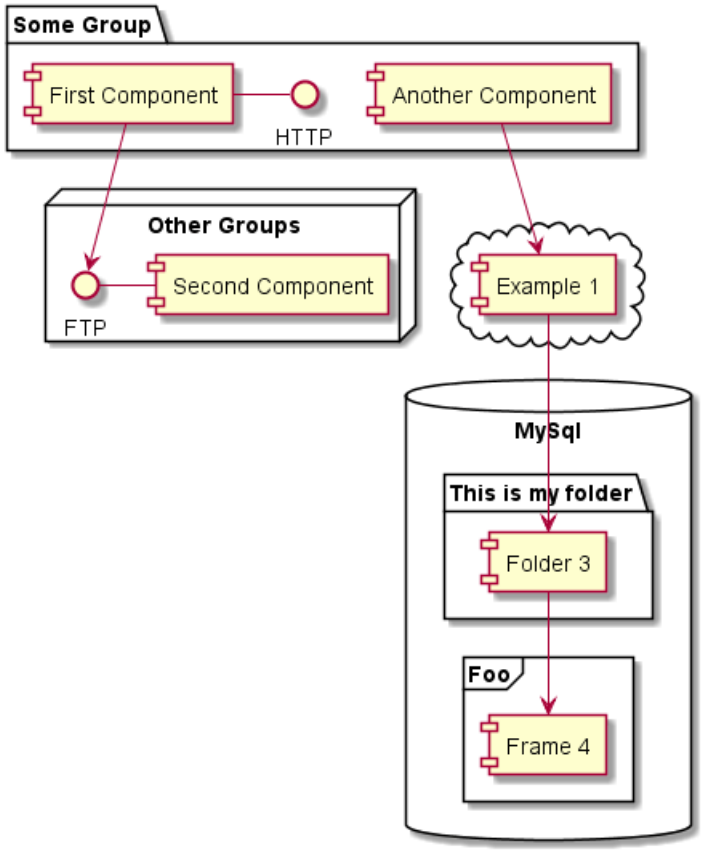


6.5 Agrupando componentes

Puedes usar varias palabras reservadas para agrupar componentes e interfaces juntos:

- package
- node
- folder
- frame
- cloud
- database

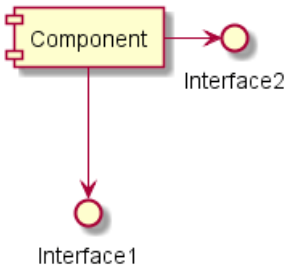
```
@startuml

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 Cambiando la dirección de las flechas

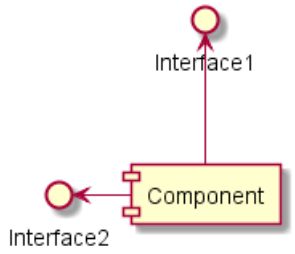
Por defecto los links entre clases tienen dos guiones `--`y son orientados verticalmente. Puedes usar la orientacion horizontal para un link poniendo un guion (o punto) como en el siguiente ejemplo:

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



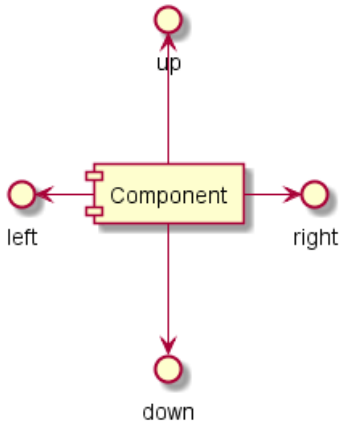
Puedes también cambiar direcciones invirtiendo el link:

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



También es posible cambiar la dirección de las flechas agregando la palabra reservada `left`, `right`, `up` o `down` dentro de la flecha:

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



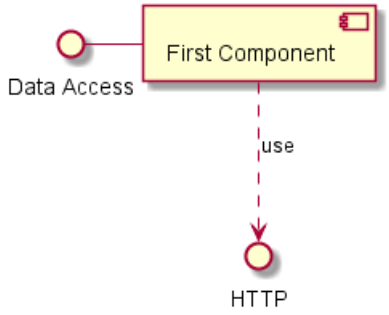
Puedes acortar la flecha usando el primer caracter (por ejemplo, `-d-` en lugar de `-down-`) o los dos primeros caracteres (`-do-`).

Por favor nota que no puedes abuzar de esta funcionalidad *Graphviz* que usualmente otorga buenos resultados sin ajustes.

6.7 Utiliza la notación UML2

El comando `skinparam componentStyle uml2` es usado para cambiar hacia la notación UML2.

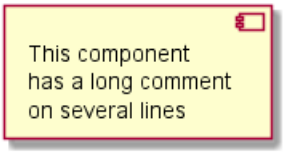
```
@startuml
skinparam componentStyle uml2

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 Colores individuales

Puedes especificar un color despues de la definición del componente.

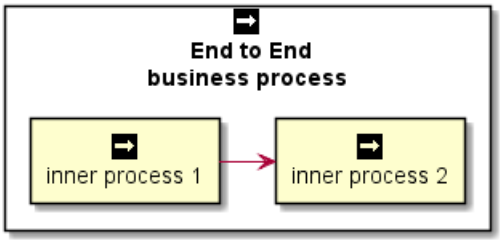
```
@startuml
component  [Web Server] #Yellow
@enduml
```

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 Personalización (Skinparam)

Puedes usar el comando `skinparam` para cambiar los colores y las fuentes de los dibujos

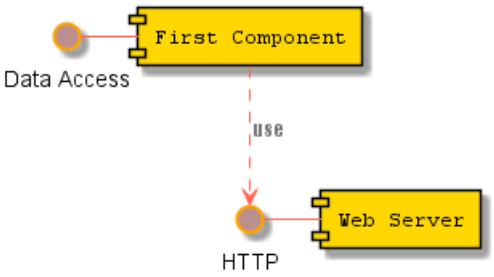
Puedes usar este comando:

- En la definición del diagrama, como cualquier otro comando,
- En un archivo incluido,
- En un archivo de configuración, proporcionado en la consola de comandos o en el ANT task.

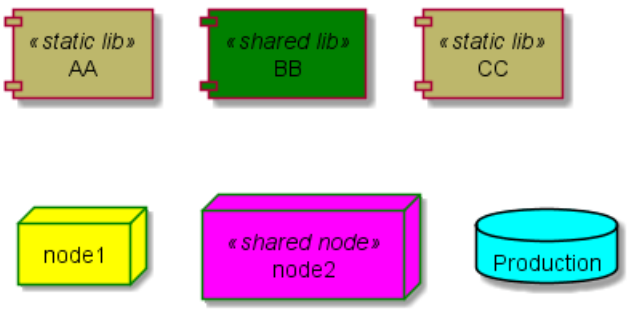
Puedes definir colores y fuentes específicas para interfaces y componentes estereotipados.

```
@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
```



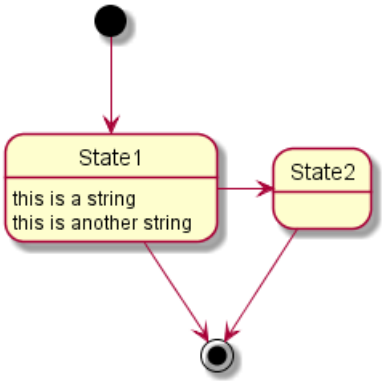
7 Diagrama de Estados

7.1 Un Estado Simple

Puedes usar `[*]` para el punto de inicio y finalización del diagrama de estados.

Utilice `-->` para las flechas.

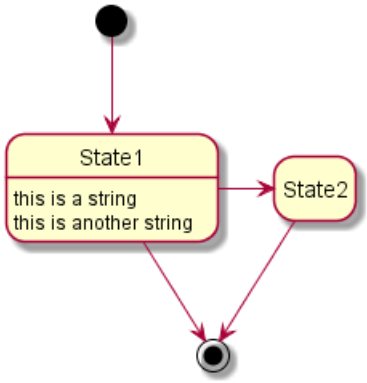
```
@startuml

[*] --> 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.

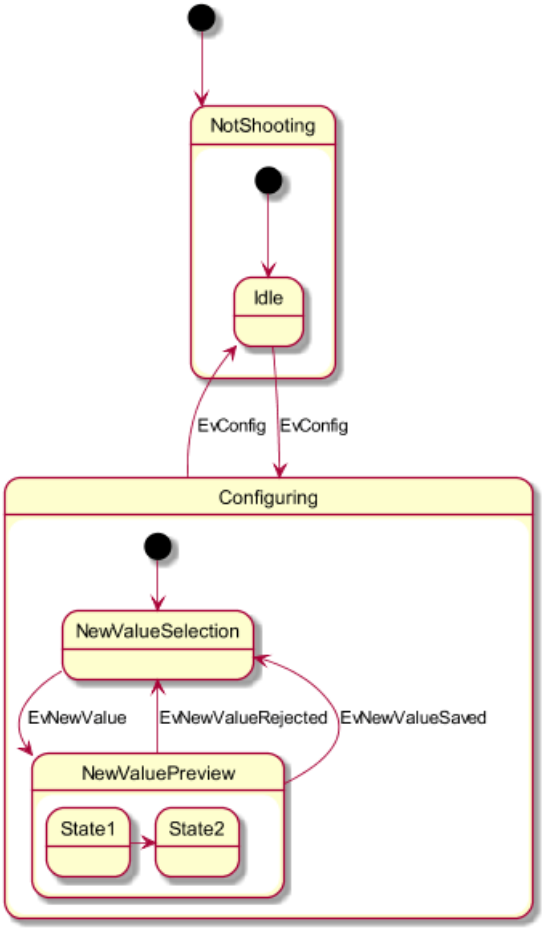
```
@startuml
hide empty description
[*] --> State1
State1 --> [*]
State1 : this is a string
State1 : this is another string

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



7.3 Estados compuestos

Un estado también puede ser compuesto. Puedes defirnirlo usando la palabra reservada `state` y llaves.

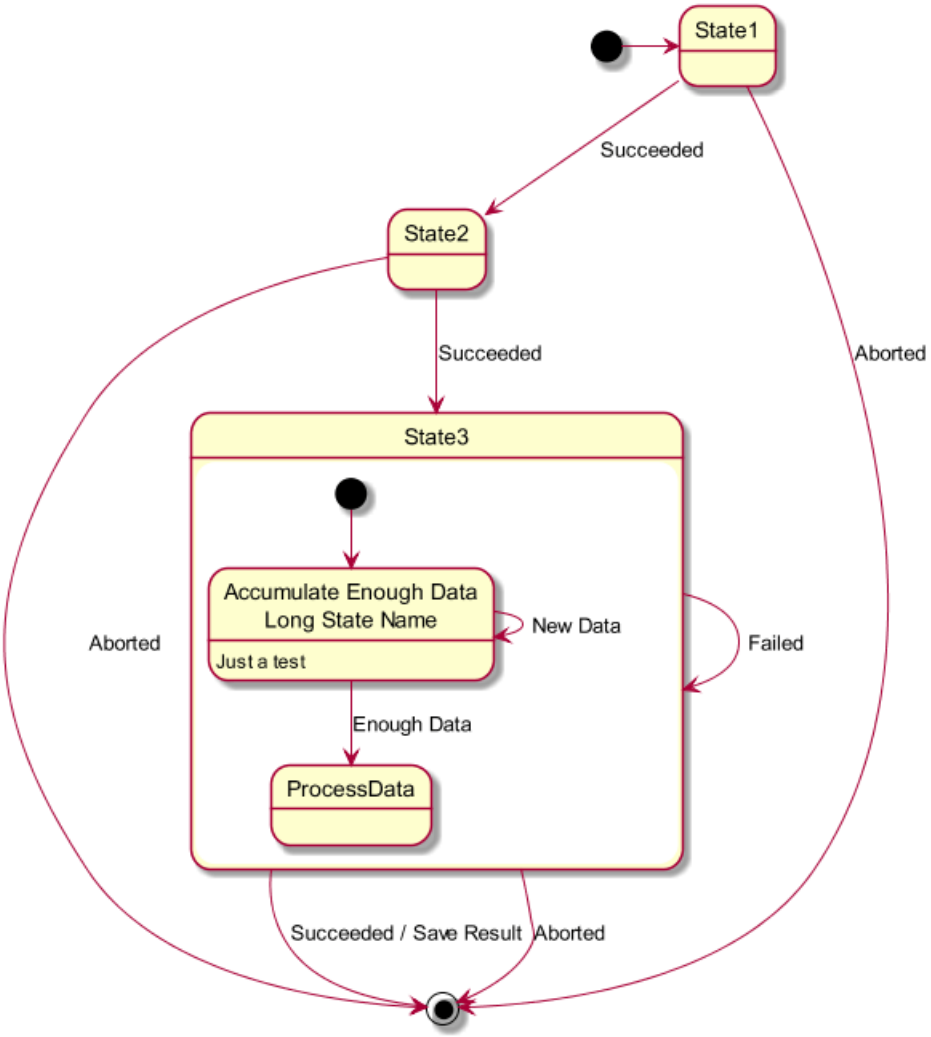
```
@startuml
scale 350 width
[*] --> NotShooting

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 Nombres largos

También puedes usar la palabra reservada `state` para definir largas descripciones en un estado.

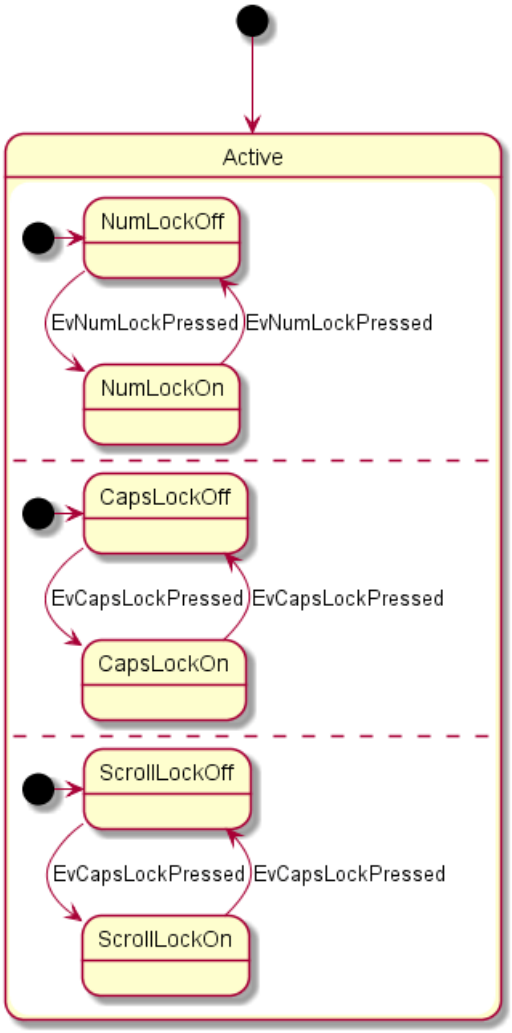
```
@startuml
scale 600 width

[*] -> 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 Estados simultáneos

Puedes definir estados simultáneos dentro de un estado compuesto usando los símbolos `--` or `||` como separadores.

```
@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
```

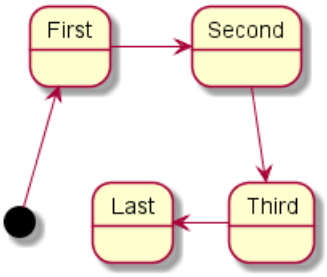


7.6 Dirección de la flecha

Puedes usar `->` para flechas horizontales. Es posible forzar la dirección de las flechas usando la siguiente sintaxis:

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

```
@startuml

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

@enduml
```



Puedes acortar un flecha usando sólamente el primer carácter del nombre de la dirección (por ejemplo, `-d-` en lugar de `-down-`) o los dos primeros caracteres (`-do-`).

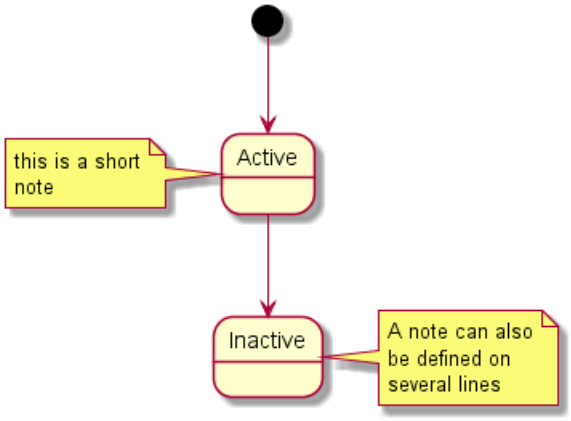
Por favor tenga en cuenta que no debería de esta funcionalidad : *Graphviz* usualmente devuelve buenos resultados sin necesidad de configuración.

7.7 Notas

También puedes definir notas usando las palabras reservadas `note left of`, `note right of`, `note top of`, `note bottom of` .

También puedes definir notas de varias líneas.

```
@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
```



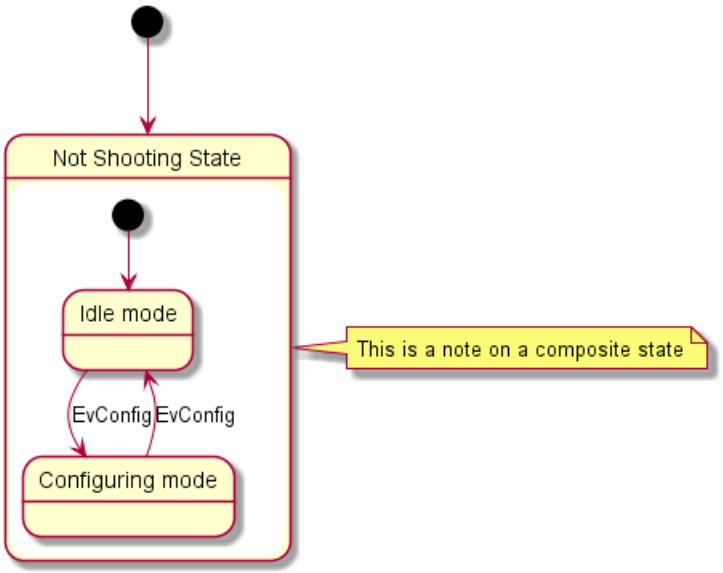
También puedes tener notas flotantes.

```
@startuml

state foo
note "This is a floating note" as N1

@enduml
```

7.8 Más sobre notas

Puedes colocar notas a estados compuestos.

```
@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 Personalización (Skinparam)

Puedes usar el comando `skinparam` para cambiar los colores y las fuentes de los dibujos

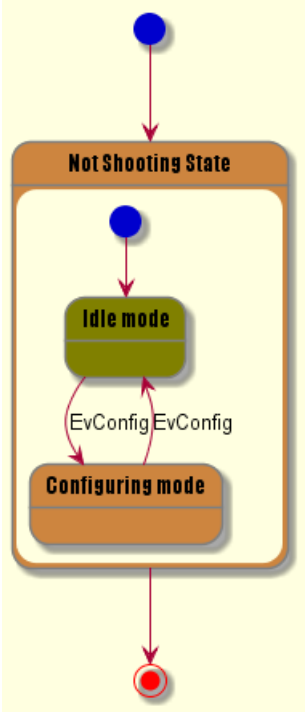
Puedes usar este comando:

- En la definición del diagrama, como cualquier otro comando,
- En un archivo incluido,
- En un archivo de configuración, proporcionado en la consola de comandos o en el ANT task.

Puedes definir colores y fuentes específicas para estados estereotipados.

```
@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
```

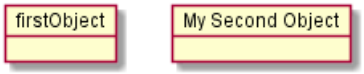


8 Diagrama de Objetos

8.1 Definición de objetos

Puedes definir instancias de objetos usando la palabra reservada `object`.

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



8.2 Relaciones entre objetos

Las relaciones entre objetos son definidas usando los siguientes símbolos:

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

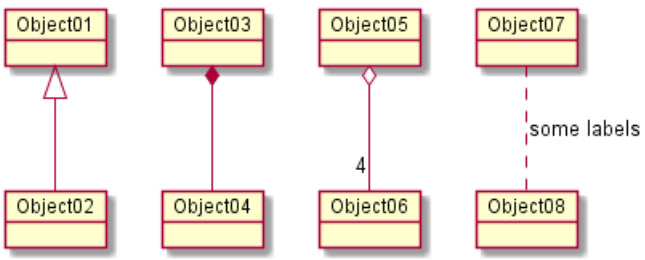
Es posible reemplazar `--` con `..` para obtener una línea de puntos.

Sabiendo estas reglas, es posible dibujar los siguientes diagramas.

Es posible agregar una etiqueta sobre una relación usando `:`, seguido del texto de la etiqueta.

Para la cardinalidad puedes usar doble comillas `""` en cada lado de la relación.

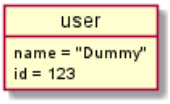
```
@startuml
object Object01
object Object02
object Object03
object Object04
object Object05
object Object06
object Object07
object Object08

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



8.3 Agregando campos

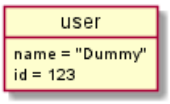
Para declarar campos, puedes usar el símbolo `:` seguido del nombre del campo.

```
@startuml

object user

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

@enduml
```



También es posible declarar entre llaves `{}` todos los campos.

```
@startuml

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

@enduml
```



8.4 Características comunes en diagramas de clases

- Visibilidad
- Definición de notas
- Uso de paquetes
- Personalización de la salida (skinparam)

9 Timing Diagram

This is only a proposal and subject to change.

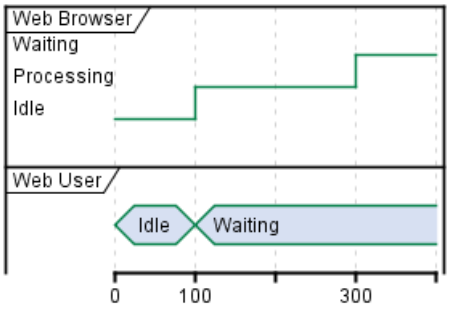
You are very welcome to create a new discussion on this future syntax. Your feedbacks, ideas and suggestions help us to find the right solution.

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
```

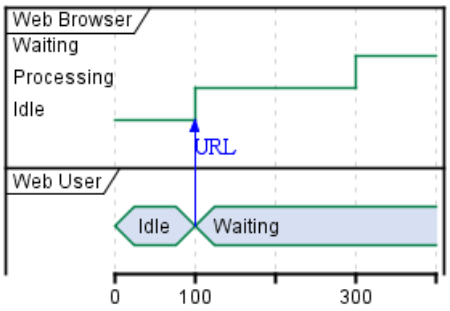


9.2 Adding message

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
```

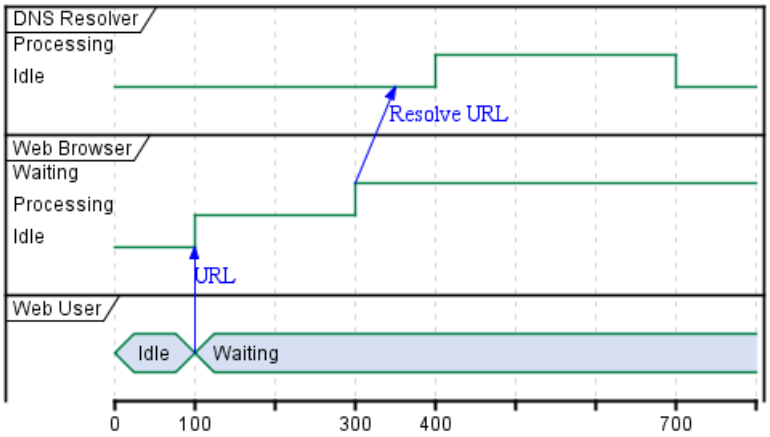
```
@enduml
```



9.3 Relative time

It is possible to use relative time with `@`.

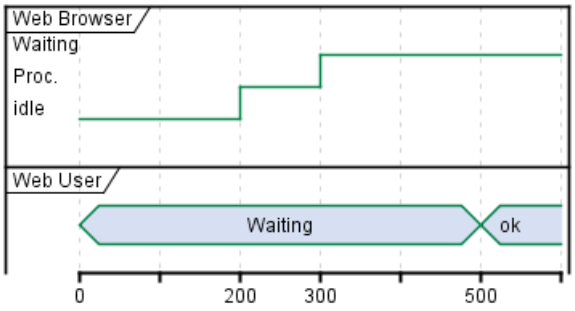
```
@startuml
robust "DNS Resolver" as DNS
robust "Web Browser" as WB
concise "Web User" as WU

@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.

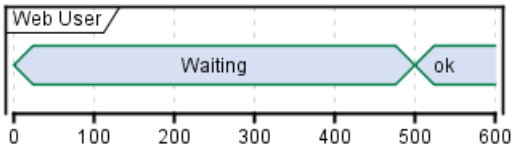
```
@startuml
robust "Web Browser" as WB
concise "Web User" as WU

@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
```

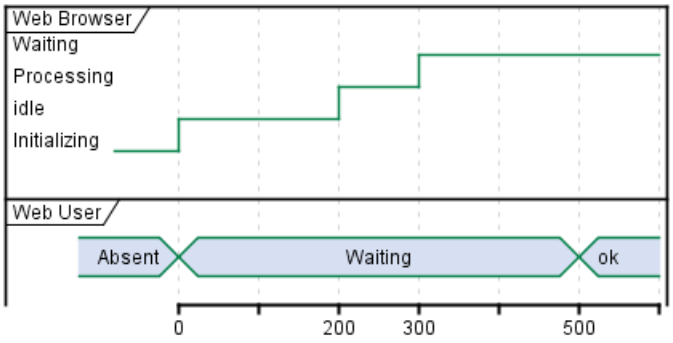


9.6 Initial state

You can also define an inital state.

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

WB is Initializing
WU is Absent
```

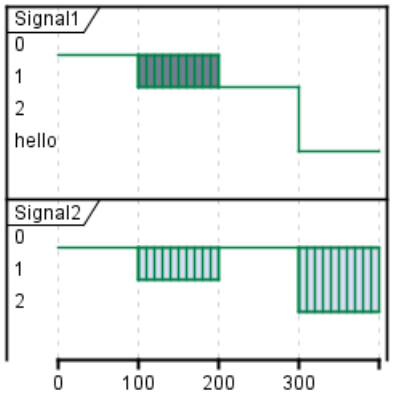
```
@WB
0 is idle
+200 is Processing
+100 is Waiting

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



9.7 Intricated state

A signal could be in some undefined state.

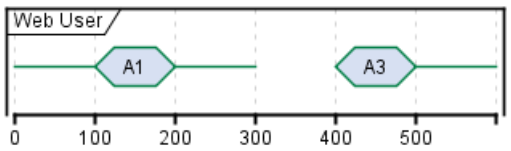
```
@startuml
robust "Signal1" as S1
robust "Signal2" as S2
S1 has 0,1,2,hello
S2 has 0,1,2
@0
S1 is 0
S2 is 0
@100
S1 is {0,1} #SlateGrey
S2 is {0,1}
@200
S1 is 1
S2 is 0
@300
S1 is hello
S2 is {0,2}
@enduml
```



9.8 Hidden state

It is also possible to hide some state.

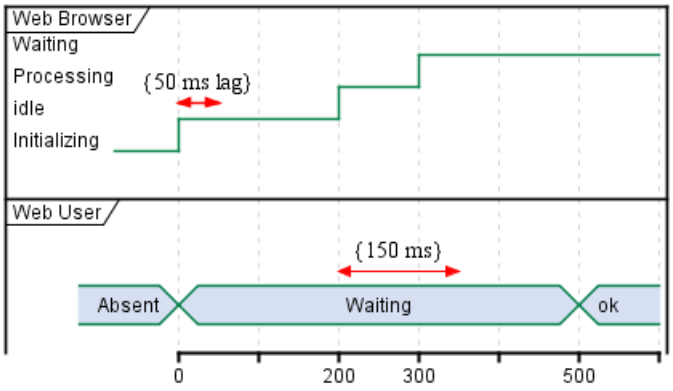
```
@startuml
concise "Web User" as WU

@0
WU is {-}

@100
WU is A1

@200
WU is {-}

@300
WU is {hidden}

@400
WU is A3

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



9.9 Adding constraint

It is possible to display time constraints on the diagrams.

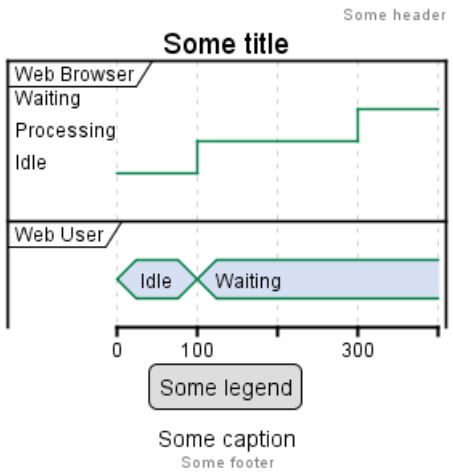
```
@startuml
robust "Web Browser" as WB
concise "Web User" as WU

WB is Initializing
WU is Absent

@WB
0 is idle
+200 is Processing
+100 is Waiting
WB@0 <-> @50 : {50 ms lag}

@WU
0 is Waiting
+500 is ok
@200 <-> @+150 : {150 ms}
@enduml
```



9.10 Adding texts

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

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

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

@0
WU is Idle
WB is Idle

@100
WU is Waiting
WB is Processing

@300
WB is Waiting
@enduml
```



10 Gantt Diagram

This is only a proposal and subject to change.

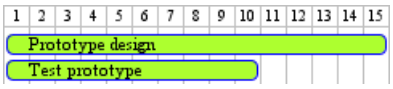
You are very welcome to create a new discussion on this future syntax. Your feedbacks, ideas and suggestions help us to find the right solution.

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

10.1 Declaring tasks

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

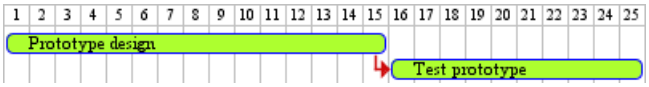
```
@startgantt
[Prototype design] lasts 15 days
[Test prototype] lasts 10 days
@endgantt
```



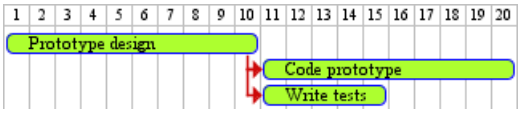
10.2 Adding constraints

It is possible to add constraints between task.

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



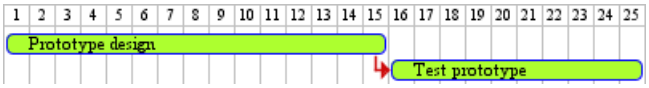

```
@startgantt
[Prototype design] lasts 10 days
[Code prototype] lasts 10 days
[Write tests] lasts 5 days
[Code prototype] starts at [Prototype design]'s end
[Write tests] starts at [Code prototype]'s start
@endgantt
```



10.3 Short names

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

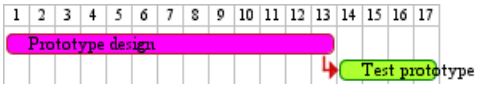
```
@startgantt
[Prototype design] as [D] lasts 15 days
[Test prototype] as [T] lasts 10 days
[T] starts at [D]'s end
@endgantt
```



10.4 Customize colors

It also possible to customize colors.

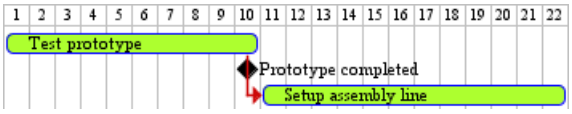
```
@startgantt
[Prototype design] lasts 13 days
[Test prototype] lasts 4 days
[Test prototype] starts at [Prototype design]'s end
[Prototype design] is colored in Fuchsia/FireBrick
[Test prototype] is colored in GreenYellow/Green
@endgantt
```



10.5 Milestone

You can define Milestones using the happens verb.

```
@startgantt
[Test prototype] lasts 10 days
[Prototype completed] happens at [Test prototype]'s end
[Setup assembly line] lasts 12 days
[Setup assembly line] starts at [Test prototype]'s end
@endgantt
```



10.6 Calendar

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

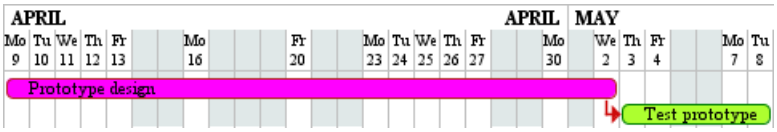
```
@startgantt
Project starts the 20th of september 2017
[Prototype design] as [TASK1] lasts 13 days
[TASK1] is colored in Lavender/LightBlue
@endgantt
```

10.7 Close day

It is possible to close some day.

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

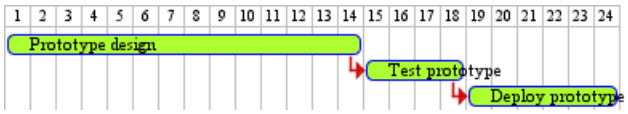
```
@endgantt
```



10.8 Simplified task succession

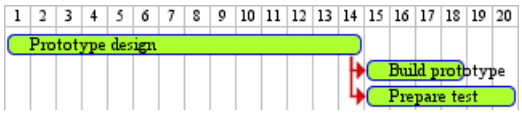
It's possible to use the then keyword to denote consecutive tasks.

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



You can also use arrow ->

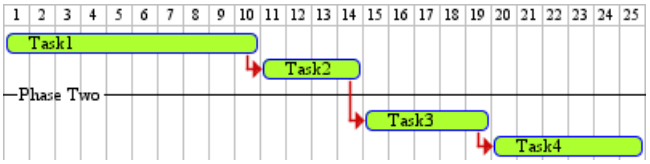
```
@startgantt
[Prototype design] lasts 14 days
[Build prototype] lasts 4 days
[Prepare test] lasts 6 days
[Prototype design] -> [Build prototype]
[Prototype design] -> [Prepare test]
@endgantt
```



10.9 Separator

You can use -- to separate sets of tasks.

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

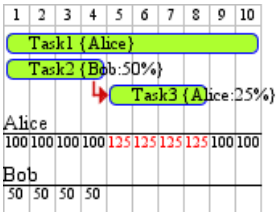


10.10 Working with resources

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

```
@startgantt
[Task1] on {Alice} lasts 10 days
[Task2] on {Bob:50%} lasts 2 days
```

```
then [Task3] on {Alice:25%} lasts 1 days
@endgantt
```

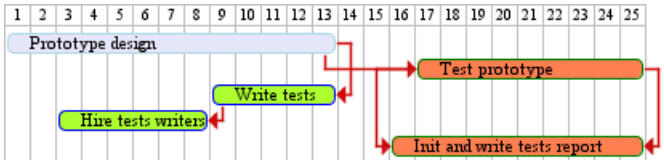


10.11 Complex example

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



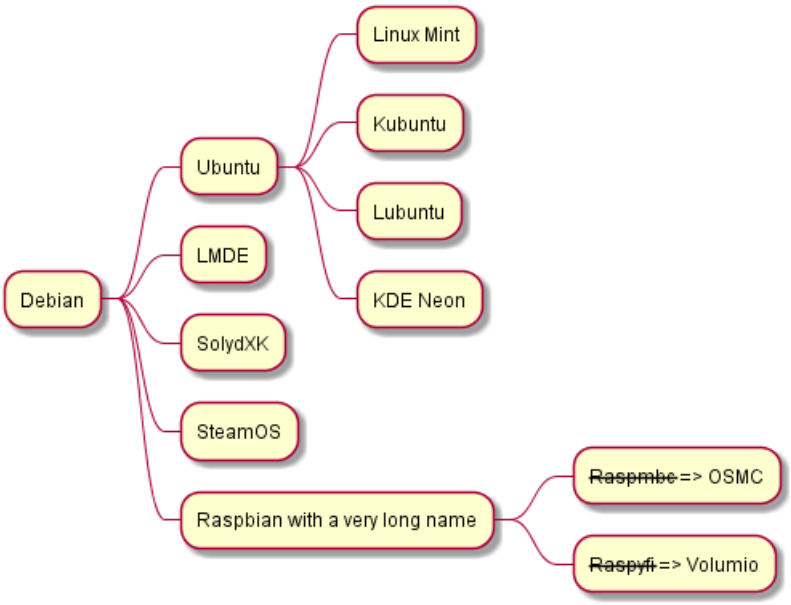
11 MindMap

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

11.1 OrgMode syntax

This syntax is compatible with OrgMode

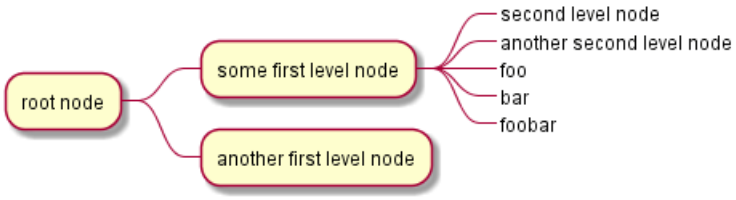
```
@startmindmap
* Debian
** Ubuntu
*** Linux Mint
*** Kubuntu
*** Lubuntu
*** KDE Neon
** LMDE
** SolydXK
** SteamOS
** Raspbian with a very long name
*** <s>Raspmbc</s> => OSMC
*** <s>Raspyfi</s> => Volumio
@endmindmap
```



11.2 Removing box

You can remove the box drawing using an underscore.

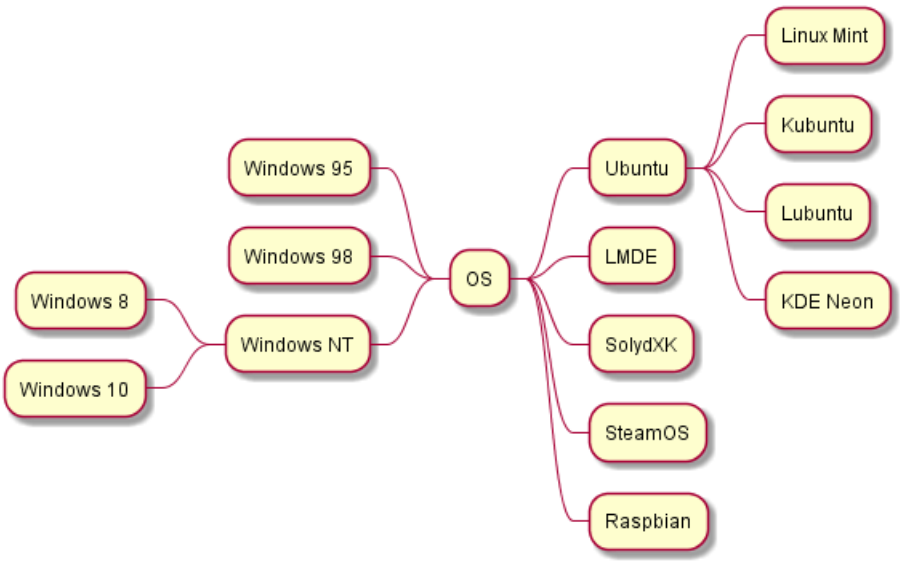
```
@startmindmap
* root node
** some first level node
***_ second level node
***_ another second level node
***_ foo
***_ bar
***_ foobar
** another first level node
@endmindmap
```



11.3 Arithmetic notation

You can use the following notation to choose diagram side.

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

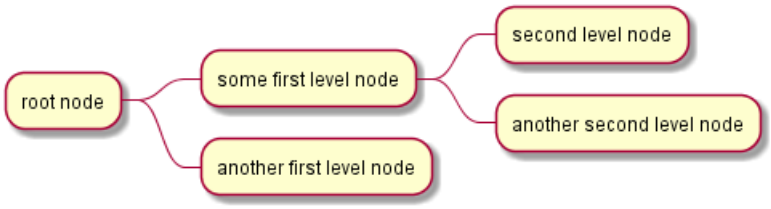


11.4 Markdown syntax

This syntax is compatible with Markdown

```
@startmindmap
* root node
* some first level node
* second level node
* another second level node
```

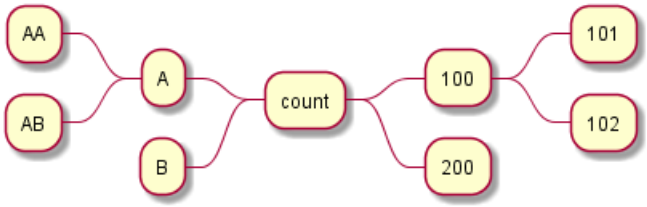
```
* another first level node
@endmindmap
```



11.5 Changing diagram direction

It is possible to use both sides of the diagram.

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

left side

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



11.6 Complete example

```
@startmindmap
caption figure 1
title My super title

* <&flag>Debian
** <&globe>Ubuntu
*** Linux Mint
*** Kubuntu
*** Lubuntu
*** KDE Neon
** <&graph>LMDE
** <&pulse>SolydXK
** <&people>SteamOS
** <&star>Raspbian with a very long name
*** <s>Raspmbc</s> => OSMC
*** <s>Raspyfi</s> => Volumio
```

```
header
My super header
endheader

center footer My super footer

legend right
  Short
  legend
endlegend
@endmindmap
```

figure 1

My super footer

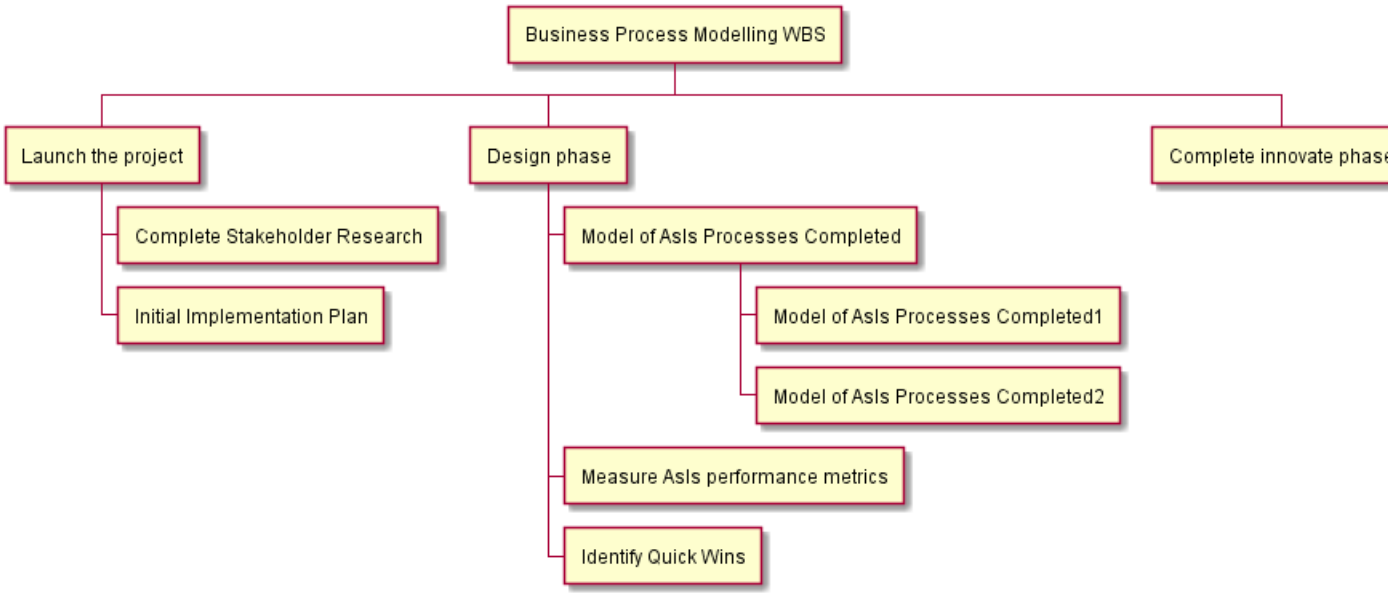
12 Work Breakdown Structure

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

12.1 OrgMode syntax

This syntax is compatible with OrgMode

```
@startwbs
* Business Process Modelling WBS
** Launch the project
*** Complete Stakeholder Research
*** Initial Implementation Plan
** Design phase
*** Model of AsIs Processes Completed
**** Model of AsIs Processes Completed1
**** Model of AsIs Processes Completed2
*** Measure AsIs performance metrics
*** Identify Quick Wins
** Complete innovate phase
@endwbs
```

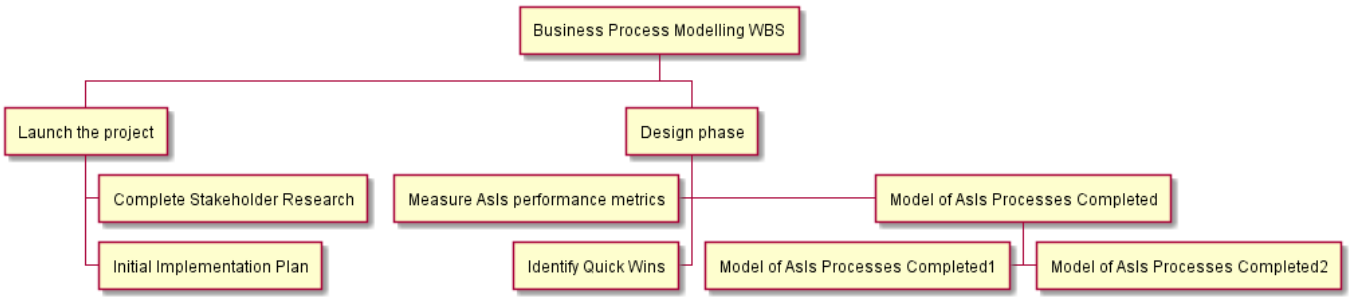


12.2 Change direction

You can change direction using < and >

```
@startwbs
* Business Process Modelling WBS
** Launch the project
*** Complete Stakeholder Research
*** Initial Implementation Plan
** Design phase
*** Model of AsIs Processes Completed
****< Model of AsIs Processes Completed1
****> Model of AsIs Processes Completed2
***< Measure AsIs performance metrics
***< Identify Quick Wins
```

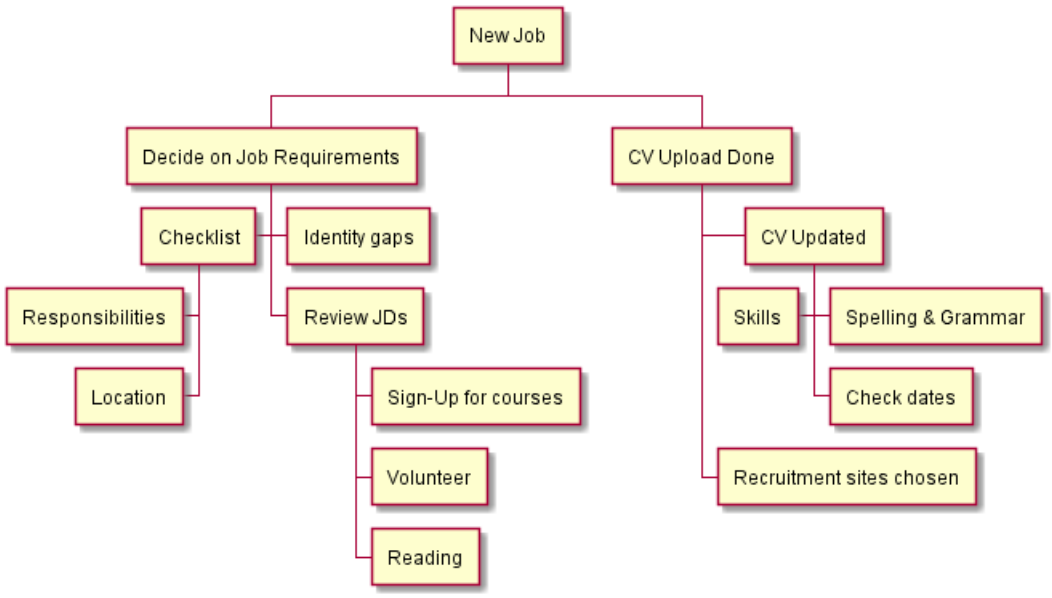
```
@endwbs
```



12.3 Arithmetic notation

You can use the following notation to choose diagram side.

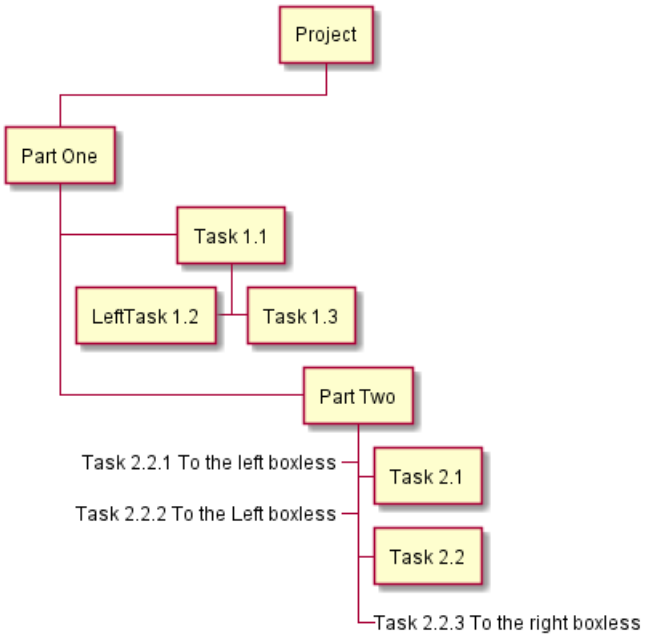
```
@startwbs
+ New Job
++ Decide on Job Requirements
+++ Identity gaps
+++ Review JDs
++++ Sign-Up for courses
++++ Volunteer
++++ Reading
++- Checklist
+++- Responsibilities
+++- Location
++ CV Upload Done
+++ CV Updated
++++ Spelling & Grammar
++++ Check dates
---- Skills
+++ Recruitment sites chosen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



You can use underscore _ to remove box drawing.

```
@startwbs
+ Project
```

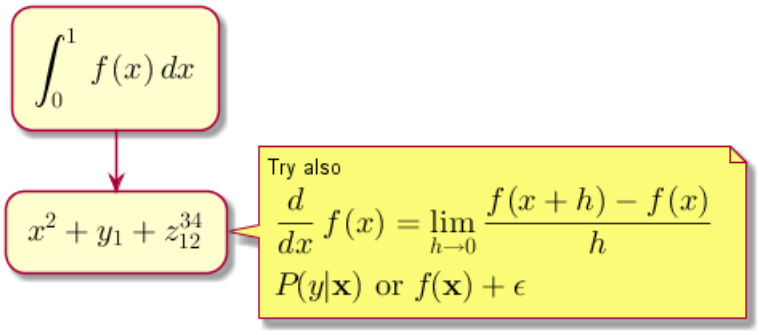
```
 + Part One
  + Task 1.1
   - LeftTask 1.2
   + Task 1.3
  + Part Two
   + Task 2.1
   + Task 2.2
   -_ Task 2.2.1 To the left boxless
   -_ Task 2.2.2 To the Left boxless
   +_ Task 2.2.3 To the right boxless
@endwbs
```



13 Maths

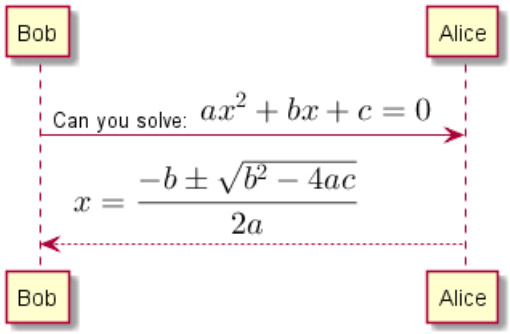
You can use AsciiMath or JLaTeXMath notation within PlantUML:

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



13.1 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
```

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

13.2 How is this working ?

To draw those formulas, PlantUML uses two OpenSource projects:

- AsciiMath that converts AsciiMath notation to LaTeX expression.
- JLatexMath that displays mathematical formulas written in LaTeX. JLaTeXMath is the best Java library to display LaTeX code.

ASCIIMathTeXImg.js is small enough to be integrated into PlantUML standard distribution.

Since JLatexMath is bigger, you have to download it separately, then unzip the 4 jar files (*batik-all-1.7.jar*, *jlatexmath-minimal-1.0.3.jar*, *jlm_cyrillic.jar* and *jlm_greek.jar*) in the same folder as PlantUML.jar.

14 Common commands

14.1 Comments

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

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

14.2 Footer and header

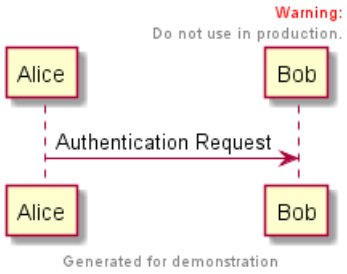
You can use the commands `header` or `footer` to add a footer or a header on any generated diagram.

You can optionally specify if you want a `center`, `left` or `right` footer/header, by adding a keyword.

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

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

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

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

center footer Generated for demonstration

@enduml
```



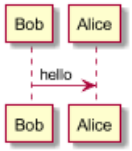
14.3 Zoom

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

You can use either a number or a fraction to define the scale factor. You can also specify either width or height (in pixel). And you can also give both width and height : the image is scaled to fit inside the specified dimension.

- `scale 1.5`
- `scale 2/3`
- `scale 200 width`
- `scale 200 height`
- `scale 200*100`
- `scale max 300*200`
- `scale max 1024 width`
- `scale max 800 height`

```
@startuml
scale 180*90
Bob->Alice : hello
@enduml
```

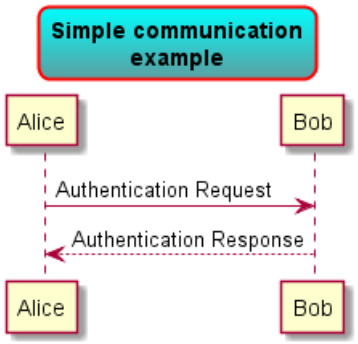


14.4 Title

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

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

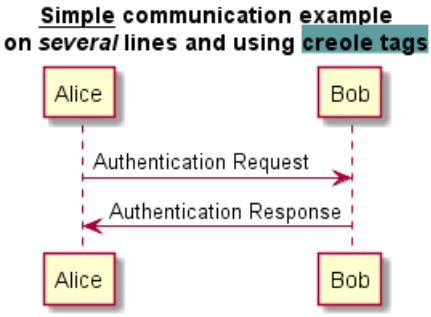
```
@startuml
skinparam titleBorderRoundCorner 15
skinparam titleBorderThickness 2
skinparam titleBorderColor red
skinparam titleBackgroundColor Aqua-CadetBlue

title Simple communication\nexample

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

@enduml
```



You can use creole formatting in the title.

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

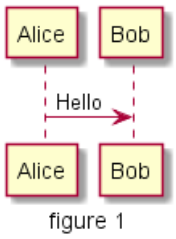
```
@startuml

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

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

@enduml
```



14.5 Caption

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

```
@startuml

caption figure 1
Alice -> Bob: Hello

@enduml
```

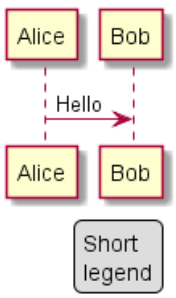


14.6 Legend the diagram

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

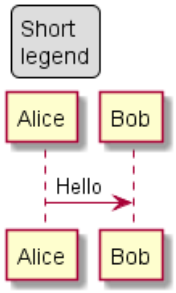
You can optionally specify to have `left`, `right`, `top`, `bottom` or `center` alignment for the legend.

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




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

```
  legend
endlegend
@enduml
```



15 Salt (wireframe)

Salt 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.

15.1 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.

15.2 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
```

15.3 Group box

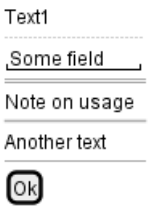
more info

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

15.4 Using separator

You can use several horizontal lines as separator.

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



15.5 Tree widget

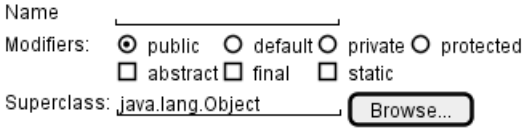
To have a Tree, you have to start with `{T` and to use `+` to denote hierarchy.

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

15.6 Enclosing brackets

You can define subelements by opening a new opening bracket.

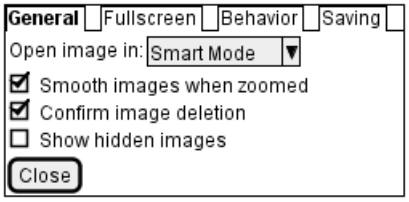
```
@startsalt
{
Name         | "                 "
Modifiers:   | { (X) public | () default | () private | () protected
            [] abstract | [] final   | [] static }
Superclass:  | { "java.lang.Object " | [Browse...] }
}
@endsalt
```



15.7 Adding tabs

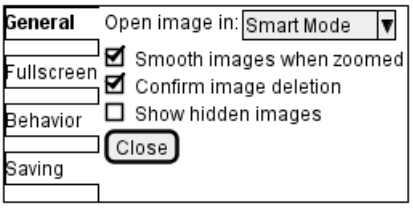
You can add tabs using {/ notation. Note that you can use HTML code to have bold text.

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



Tab could also be vertically oriented:

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

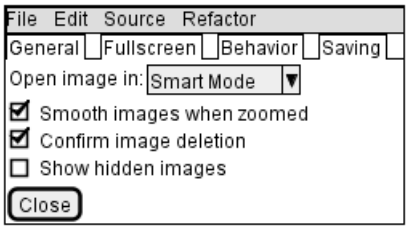


15.8 Using menu

You can add a menu by using {* notation.

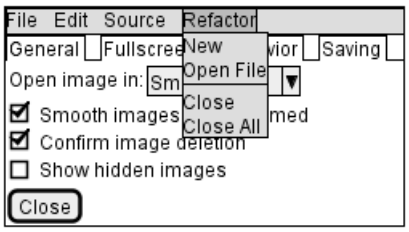
```
@startsalt
{+
{* File | Edit | Source | Refactor }
{/ General | Fullscreen | Behavior | Saving }
{
{ Open image in: | ^Smart Mode^ }
[X] Smooth images when zoomed
```

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



It is also possible to open a menu:

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



15.9 Advanced table

You can use two special notations for table :

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

15.10 OpenIconic

OpenIconic is an very nice open source icon set. Those icons have been integrated into the creole parser, so you can use them out-of-the-box. You can use the following syntax: `<&ICON_NAME>`.

```
@startsalt
{
  Login<&person> | "MyName   "
  Password<&key> | "****     "
  [Cancel <&circle-x>] | [OK <&account-login>]
}
@endsalt
```

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

```
@startuml
listopeniconic
@enduml
```

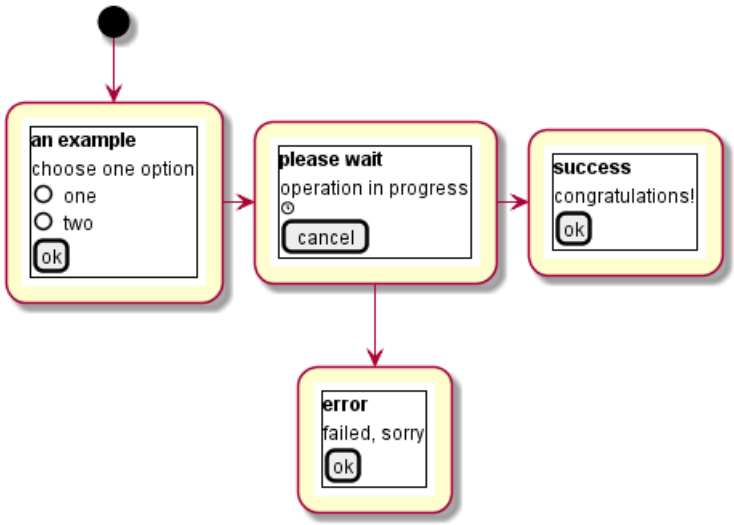
List Open Iconic	bell	cloud	excerpt	justify-right	musical-note	star
Credit to	bluetooth	cloudy	expand-down	key	paperclip	sun
https://useiconic.com/open	bold	code	expand-left	laptop	pencil	tablet
	bolt	cog	expand-right	layers	people	tag
account-login	book	collapse-down	expand-up	lightbulb	person	tags
account-logout	bookmark	collapse-left	external-link	link-broken	phone	target
action-redo	box	collapse-right	eye	link-intact	pie-chart	task
action-undo	briefcase	collapse-up	eyedropper	list-rich	pin	terminal
align-center	british-pound	command	file	list	play-circle	text
align-left	browser	comment-square	fire	location	plus	thumb-down
align-right	brush	compass	flag	lock-locked	power-standby	thumb-up
aperture	bug	contrast	flash	lock-unlocked	print	timer
arrow-bottom	bullhorn	copywriting	folder	loop-circular	project	transfer
arrow-circle-bottom	calculator	credit-card	fork	loop-square	pulse	trash
arrow-circle-left	calendar	crop	fullscreen-enter	loop	puzzle-piece	underline
arrow-circle-right	camera-slr	dashboard	fullscreen-exit	magnifying-glass	question-mark	vertical-align-bottom
arrow-circle-top	caret-bottom	data-transfer-download	globe	map-marker	rain	vertical-align-center
arrow-left	caret-left	data-transfer-upload	graph	map	random	vertical-align-top
arrow-right	caret-right	delete	grid-four-up	media-pause	reload	video
arrow-thick-bottom	caret-top	dial	grid-three-up	media-play	resize-both	volume-high
arrow-thick-left	cart	document	grid-two-up	media-record	resize-height	volume-low
arrow-thick-right	chat	dollar	hard-drive	media-skip-backward	resize-width	volume-off
arrow-thick-top	check	double-quote-sans-left	header	media-skip-forward	rss-alt	warning
arrow-top	chevron-bottom	double-quote-sans-right	headphones	media-step-backward	rss	wifi
audio-spectrum	chevron-left	double-quote-serif-left	heart	media-step-forward	script	wrench
audio	chevron-right	double-quote-serif-right	home	media-stop	share-boxed	x
badge	chevron-top	droplet	image	medical-cross	share	yen
ban	circle-check	eject	inbox	menu	shield	zoom-in
bar-chart	circle-x	elevator	infinity	microphone	signal	zoom-out
basket	clipboard	ellipses	info	minus	signpost	
battery-empty	clock	envelope-closed	italic	monitor	sort-ascending	
battery-full	cloud-download	envelope-open	justify-center	moon	sort-descending	
beaker	cloud-upload	euro	justify-left	move	spreadsheet	

15.11 Include Salt

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

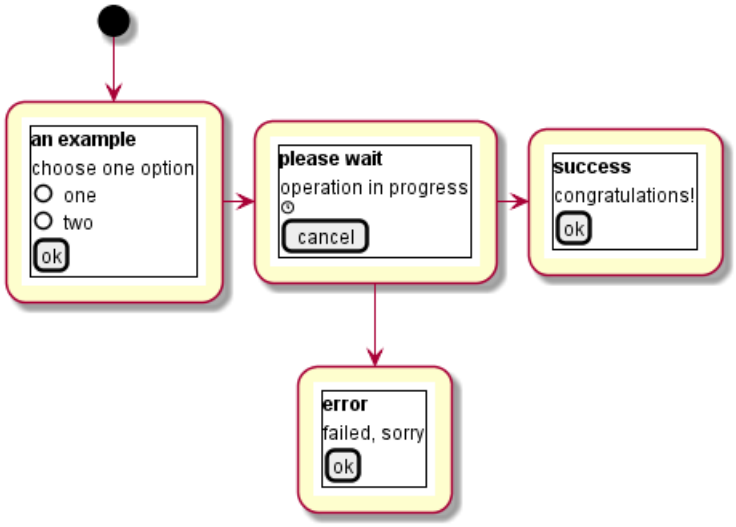
```
@startuml
(*) --> "
{{
salt
{+
<b>an example
choose one option
()one
```

```
()two
[ok]
}
}}
" as choose

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

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



It can also be combined with define macro.

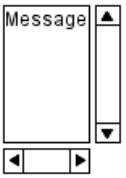
```
@startuml
!unquoted function SALT($x)
"{{
salt
%invoke_void_func("_"+$x)
}}" as $x
!endfunction

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

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

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

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

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



15.12 Scroll Bars

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

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




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


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

```
.
}
@endsalt
```

16 Creole

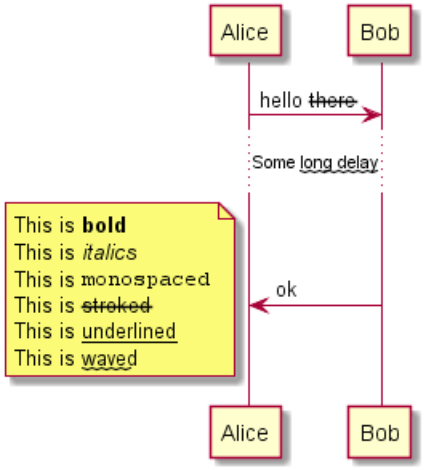
A light Creole engine has been integrated into PlantUML to have a standardized way of defining text style.

All diagrams are now supporting this syntax.

Note that ascending compatibility with HTML syntax is preserved.

16.1 Emphasized text

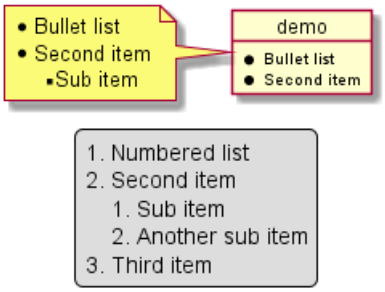
```
@startuml
Alice -> Bob : hello --there--
... Some ~~long delay~~ ...
Bob -> Alice : ok
note left
  This is **bold**
  This is //italics//
  This is ""monospaced""
  This is --stroked--
  This is __underlined__
  This is ~~waved~~
end note
@enduml
```



16.2 List

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

legend
  # Numbered list
  # Second item
  ## Sub item
  ## Another sub item
```

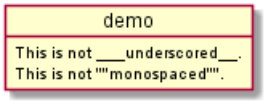
```
  # Third item
end legend
@enduml
```



16.3 Escape character

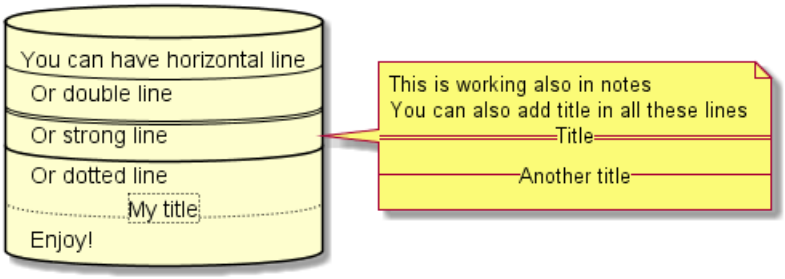
You can use the tilde ~ to escape special creole characters.

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



16.4 Horizontal lines

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

@enduml
```



16.5 Headings

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

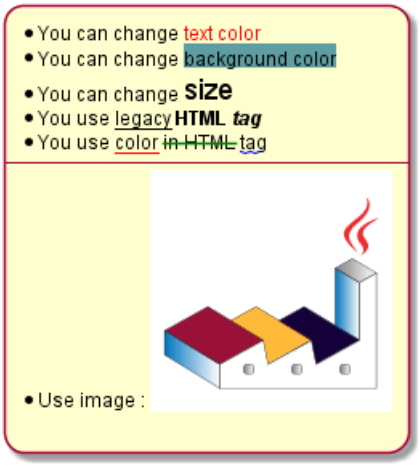
16.6 Legacy HTML

Some HTML tags are also working:

- `<b>` for bold text
- `<u>` or `<u:#AAAAAA>` or `<u:colorName>` for underline
- `<i>` for italic
- `<s>` or `<s:#AAAAAA>` or `<s:colorName>` for strike text
- `<w>` or `<w:#AAAAAA>` or `<w:colorName>` for wave underline text
- `<color:#AAAAAA>` or `<color:colorName>`
- `<back:#AAAAAA>` or `<back:colorName>` for background color
- `<size:nn>` to change font size
- `<img:file>` : the file must be accessible by the filesystem
- `<img:http://plantuml.com/logo3.png>` : the URL must be available from the Internet

```
@startuml
:* You can change <color:red>text color</color>
* You can change <back:cadetblue>background color</back>
* You can change <size:18>size</size>
```

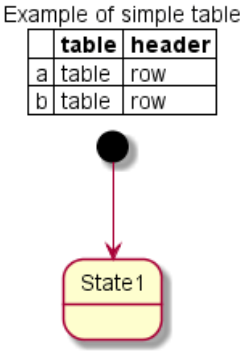
```
* You use <u>legacy</u> <b>HTML <i>tag</i></b>
* You use <u:red>color</u> <s:green>in HTML</s> <w:#0000FF>tag</w>
----
* Use image : <img:http://plantuml.com/logo3.png>
;
@enduml
```



16.7 Table

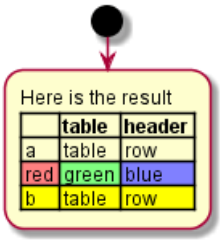
It is possible to build table.

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



You can specify background colors for cells and lines.

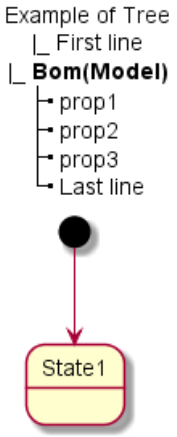
```
@startuml
start
:Here is the result
|= |= table |= header |
| a | table | row |
|<#FF8080> red |<#80FF80> green |<#8080FF> blue |
<#yellow>| b | table | row |;
@enduml
```



16.8 Tree

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

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



16.9 Special characters

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

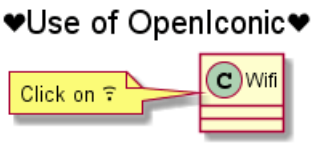
```
@startuml
usecase foo as "this is &#8734; long"
usecase bar as "this is also <U+221E> long"
@enduml
```

16.10 OpenIconic

OpenIconic is an very nice open source icon set. Those icons have been integrated into the creole parser, so you can use them out-of-the-box.

You can use the following syntax: <&ICON_NAME>.

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



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

```
@startuml
listopeniconic
@enduml
```

List Open Iconic	bell	cloud	excerpt	justify-right	musical-note	star
Credit to	bluetooth	cloudy	expand-down	key	paperclip	sun
https://useiconic.com/open	bold	code	expand-left	laptop	pencil	tablet
	bolt	cog	expand-right	layers	people	tag
account-login	book	collapse-down	expand-up	lightbulb	person	tags
account-logout	bookmark	collapse-left	external-link	link-broken	phone	target
action-redo	box	collapse-right	eye	link-intact	pie-chart	task
action-undo	briefcase	collapse-up	eyedropper	list-rich	pin	terminal
align-center	british-pound	command	file	list	play-circle	text
align-left	browser	comment-square	fire	location	plus	thumb-down
align-right	brush	compass	flag	lock-locked	power-standby	thumb-up
aperture	bug	contrast	flash	lock-unlocked	print	timer
arrow-bottom	bullhorn	copywriting	folder	loop-circular	project	transfer
arrow-circle-bottom	calculator	credit-card	fork	loop-square	pulse	trash
arrow-circle-left	calendar	crop	fullscreen-enter	loop	puzzle-piece	underline
arrow-circle-right	camera-slr	dashboard	fullscreen-exit	magnifying-glass	question-mark	vertical-align-bottom
arrow-circle-top	caret-bottom	data-transfer-download	globe	map-marker	rain	vertical-align-center
arrow-left	caret-left	data-transfer-upload	graph	map	random	vertical-align-top
arrow-right	caret-right	delete	grid-four-up	media-pause	reload	video
arrow-thick-bottom	caret-top	dial	grid-three-up	media-play	resize-both	volume-high
arrow-thick-left	cart	document	grid-two-up	media-record	resize-height	volume-low
arrow-thick-right	chat	dollar	hard-drive	media-skip-backward	resize-width	volume-off
arrow-thick-top	check	double-quote-sans-left	header	media-skip-forward	rss-alt	warning
arrow-top	chevron-bottom	double-quote-sans-right	headphones	media-step-backward	rss	wifi
audio-spectrum	chevron-left	double-quote-serif-left	heart	media-step-forward	script	wrench
audio	chevron-right	double-quote-serif-right	home	media-stop	share-boxed	x
badge	chevron-top	droplet	image	medical-cross	share	yen
ban	circle-check	eject	inbox	menu	shield	zoom-in
bar-chart	circle-x	elevator	infinity	microphone	signal	zoom-out
basket	clipboard	ellipses	info	minus	signpost	
battery-empty	clock	envelope-closed	italic	monitor	sort-ascending	
battery-full	cloud-download	envelope-open	justify-center	moon	sort-descending	
beaker	cloud-upload	euro	justify-left	move	spreadsheet	

17 Defining and using sprites

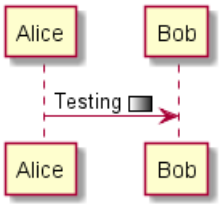
A *Sprite* is a small graphic element that can be used in diagrams.

In PlantUML, sprites are monochrome and can have either 4, 8 or 16 gray level.

To define a sprite, you have to use a hexadecimal digit between 0 and F per pixel.

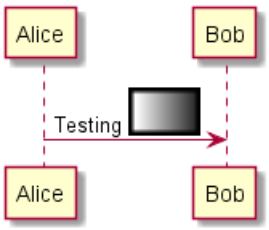
Then you can use the sprite using `<$XXX>` where XXX is the name of the sprite.

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



17.1 Encoding Sprite

To encode sprite, you can use the command line like:

```
java -jar plantuml.jar -encodesprite 16z foo.png
```

where `foo.png` is the image file you want to use (it will be converted to gray automatically).

After `-encodesprite`, you have to specify a format: `4, 8, 16, 4z, 8z` or `16z`.

The number indicates the gray level and the optional `z` is used to enable compression in sprite definition.

17.2 Importing Sprite

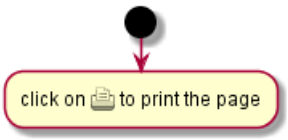
You can also launch the GUI to generate a sprite from an existing image.

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

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

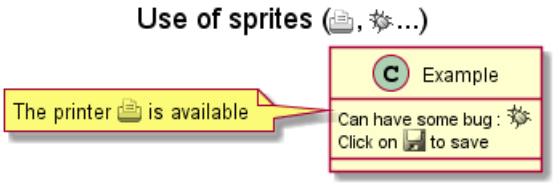
17.3 Examples

```
@startuml
sprite $printer [15x15/8z] NOtH3W0W208HxFz_kMAhj7lHWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvF
start
:click on <$printer> to print the page;
@enduml
```




```
@startuml
 sprite $bug [15x15/16z] PKzR2i0m2BFMi15p__FEjQEqB1z27aeqCqixa8S4OT7C53cKpsHpaYPDJY_12MHM-BLRyywPhrrlw
 sprite $printer [15x15/8z] NOtH3W0W208HxFz_kMAhj7lHWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvF
 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
```



18 Skinparam command

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

Example:

```
skinparam backgroundColor transparent
```

18.1 Usage

You can use this command :

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

18.2 Nested

To avoid repetition, it is possible to nest definition. So the following definition :

```
skinparam xxxxParam1 value1
skinparam xxxxParam2 value2
skinparam xxxxParam3 value3
skinparam xxxxParam4 value4
```

is strictly equivalent to:

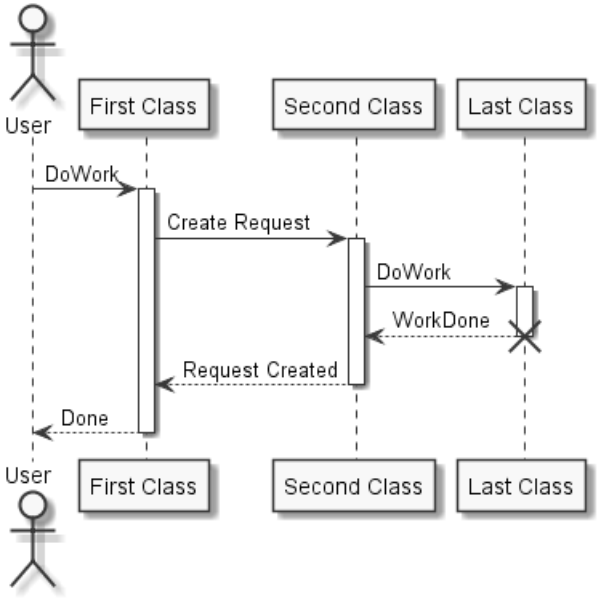
```
skinparam xxxx {
    Param1 value1
    Param2 value2
    Param3 value3
    Param4 value4
}
```

18.3 Black and White

You can force the use of a black&white output using `skinparam monochrome true` command.

```
@startuml

skinparam monochrome true

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

User -> A: DoWork
activate A

A -> B: Create Request
activate B

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

B --> A: Request Created
```

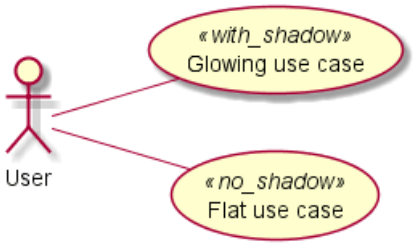
```
deactivate B

A --> User: Done
deactivate A

@enduml
```



18.4 Shadowing

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

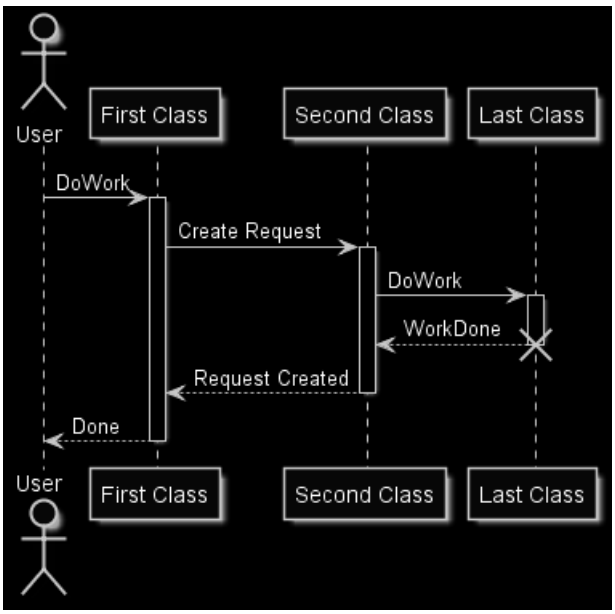
```
@startuml

left to right direction

skinparam shadowing<<no_shadow>> false
skinparam shadowing<<with_shadow>> true

actor User
(Glowing use case) <<with_shadow>> as guc
(Flat use case) <<no_shadow>> as fuc
User -- guc
User -- fuc

@enduml
```



18.5 Reverse colors

You can force the use of a black&white output using `skinparam monochrome reverse` command. This can be useful for black background environment.

```
@startuml

skinparam monochrome reverse

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

User -> A: DoWork
activate A

A -> B: Create Request
activate B

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

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml
```



18.6 Colors

You can use either standard color name or RGB code.

APPLICATION	Crimson	DeepPink	Indigo	LightYellow	Navy	RoyalBlue	Turquoise
AliceBlue	Cyan	DeepSkyBlue	Ivory	Lime	OldLace	STRATEGY	Violet
AntiqueWhite	DarkBlue	DimGray	Khaki	LimeGreen	Olive	SaddleBrown	Wheat
Aqua	DarkCyan	DimGrey	Lavender	Linen	OliveDrab	Salmon	White
Aquamarine	DarkGoldenRod	DodgerBlue	LavenderBlush	MOTIVATION	Orange	SandyBrown	WhiteSmoke
Azure	DarkGray	FireBrick	LawnGreen	Magenta	OrangeRed	SeaGreen	Yellow
BUSINESS	DarkGreen	FloralWhite	LemonChiffon	Maroon	Orchid	SeaShell	YellowGreen
Beige	DarkGrey	ForestGreen	LightBlue	MediumAquaMarine	PHYSICAL	Sienna	
Bisque	DarkKhaki	Fuchsia	LightCoral	MediumBlue	PaleGoldenRod	Silver	
Black	DarkMagenta	Gainsboro	LightCyan	MediumOrchid	PaleGreen	SkyBlue	
BlanchedAlmond	DarkOliveGreen	GhostWhite	LightGoldenRodYellow	MediumPurple	PaleTurquoise	SlateBlue	
Blue	DarkOrchid	Gold	LightGray	MediumSeaGreen	PaleVioletRed	SlateGray	
BlueViolet	DarkRed	GoldenRod	LightGreen	MediumSlateBlue	PapayaWhip	SlateGrey	
Brown	DarkSalmon	Gray	LightGrey	MediumSpringGreen	PeachPuff	Snow	
BurlyWood	DarkSeaGreen	Green	LightPink	MediumTurquoise	Peru	SpringGreen	
CadetBlue	DarkSlateBlue	GreenYellow	LightSalmon	MediumVioletRed	Pink	SteelBlue	
Chartreuse	DarkSlateGray	Grey	LightSeaGreen	MidnightBlue	Plum	TECHNOLOGY	
Chocolate	DarkSlateGrey	HoneyDew	LightSkyBlue	MintCream	PowderBlue	Tan	
Coral	DarkTurquoise	HotPink	LightSlateGray	MistyRose	Purple	Teal	
CornflowerBlue	DarkViolet	IMPLEMENTATION	LightSlateGrey	Moccasin	Red	Thistle	
Cornsilk	Darkorange	IndianRed	LightSteelBlue	NavajoWhite	RosyBrown	Tomato	

`transparent` can only be used for background of the image.

18.7 Font color, name and size

You can change the font for the drawing using `xxxFontColor`, `xxxFontSize` and `xxxFontName` parameters.

Example:

```
skinparam classFontColor red
skinparam classFontSize 10
skinparam classFontName Aapex
```

You can also change the default font for all fonts using `skinparam defaultFontName`.

Example:

```
skinparam defaultFontName Aapex
```

Please note the fontname is highly system dependent, so do not over use it, if you look for portability. `Helvetica` and `Courier` should be available on all system.

A lot of parameters are available. You can list them using the following command:

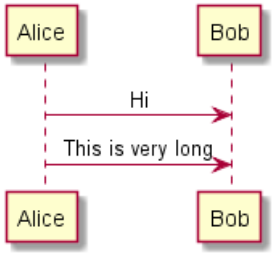
```
java -jar plantuml.jar -language
```

18.8 Text Alignment

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

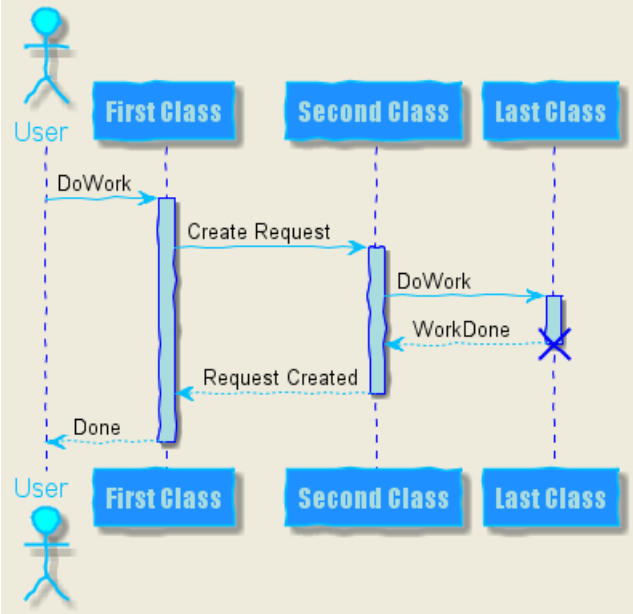
Param name	Default value	Comment
sequenceMessageAlign	left	Used for messages in sequence diagrams
sequenceReferenceAlign	center	Used for `ref over` in sequence diagrams

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

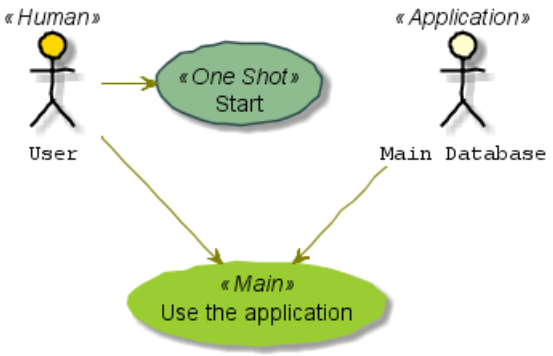


18.9 Examples

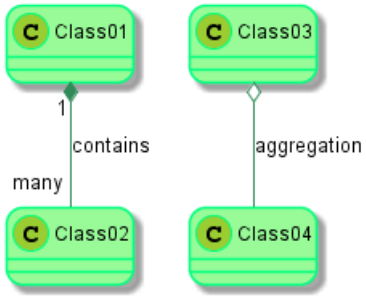
```
@startuml
skinparam backgroundColor #EEEBDC
skinparam handwritten true

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
```

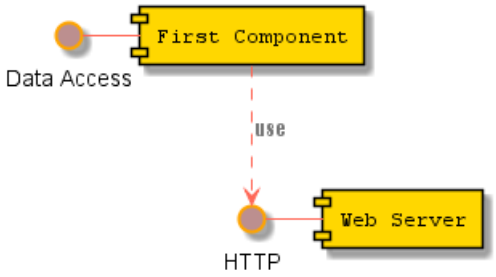



```
@startuml
skinparam roundcorner 20
skinparam class {
BackgroundColor PaleGreen
ArrowColor SeaGreen
BorderColor SpringGreen
}
skinparam stereotypeCBackgroundColor YellowGreen

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

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

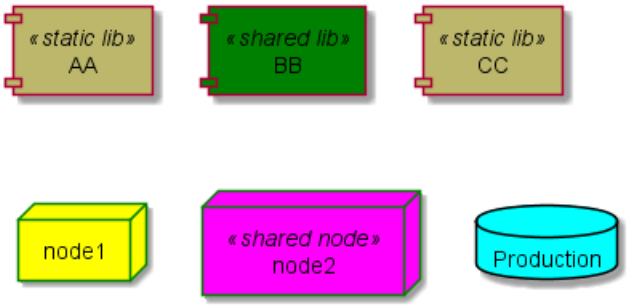



```
@startuml

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
```



18.10 List of all skinparam parameters

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

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

Or you can generate a "diagram" with a list of all the skinparam parameters using:

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

That will give you the following result:

Help on skinparam

The code of this command is located in *net.sourceforge.plantuml.help* package.
You may improve it on *https://github.com/plantuml/plantuml/tree/master/src/net/sourceforge/plantuml/help*

The possible skinparam are :

- ActivityBackgroundColor
- ActivityBarColor
- ActivityBorderColor
- ActivityBorderThickness
- ActivityDiamondBackgroundColor
- ActivityDiamondBorderColor
- ActivityDiamondFontColor
- ActivityDiamondFontName
- ActivityDiamondFontSize
- ActivityDiamondFontStyle
- ActivityEndColor
- ActivityFontColor
- ActivityFontName
- ActivityFontSize
- ActivityFontStyle
- ActivityStartColor
- ActorBackgroundColor
- ActorBorderColor
- ActorFontColor
- ActorFontName
- ActorFontSize
- ActorFontStyle
- ActorStereotypeFontColor
- ActorStereotypeFontName
- ActorStereotypeFontSize
- ActorStereotypeFontStyle
- AgentBackgroundColor
- AgentBorderColor
- AgentBorderThickness
- AgentFontColor
- AgentFontName
- AgentFontSize
- AgentFontStyle
- AgentStereotypeFontColor
- AgentStereotypeFontName
- AgentStereotypeFontSize
- AgentStereotypeFontStyle
- ArchimateBackgroundColor
- ArchimateBorderColor
- ArchimateBorderThickness
- ArchimateFontColor
- ArchimateFontName
- ArchimateFontSize
- ArchimateFontStyle
- ArchimateStereotypeFontColor
- ArchimateStereotypeFontName
- ArchimateStereotypeFontSize
- ArchimateStereotypeFontStyle
- ArrowColor
- ArrowFontColor
- ArrowFontName
- ArrowFontSize
- ArrowFontStyle
- ArrowLollipopColor
- ArrowMessageAlignment
- ArrowThickness
- ArtifactBackgroundColor

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

19 Preprocessing

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

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

19.1 Migration notes

The actual preprocessor is an update from some legacy preprocessor.

Even if some legacy feature are still supported with the actual preprocessor, you should not use them any more (they might be finally removed in some long term future).

- You should not use `!define` and `!definelong` anymore. Use `!function` and variable definition instead
- `!include` allows now multiple inclusions : you don't have to use `!include_many` anymore
- `!include` now accept URL, so you don't need `!includeurl`
- Some features (like `%date%`) have been replaced by builtin functions (for example `%date()`)
- When calling a legacy `!definelong` macro with no arguments, you do have to use parenthesis. That is you have to use `my_own_definelong()` because `my_own_definelong` without parenthesis is not recognized by the new preprocessor.

Please contact us if you have any issues.

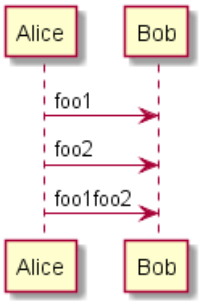
19.2 Variable definition

Although this is not mandatory, we highly suggest that variable name start with a `$`. There are two kind of data:

- Integer number
- String, that must be surrender by simple quote or double quote.

Variable created outside function are **global**, that is you can access to them from everywhere (including from functions). You can emphasize this by using the optional `global` keyword when defining a variable.

```
@startuml
!$ab = "foo1"
!$cd = "foo2"
!global $ef = $ab + $cd

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

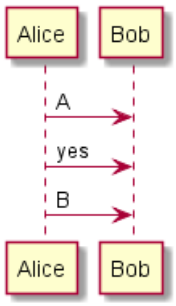


19.3 Conditions

- You can use expression in condition.

- *else* is also implemented

```
@startuml
!$a = 10
!$ijk = "foo"
Alice -> Bob : A
!if ($ijk == "foo") && ($a+10>=4)
Alice -> Bob : yes
!else
Alice -> Bob : This should not appear
!endif
Alice -> Bob : B
@enduml
```

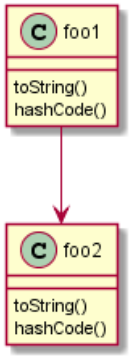


19.4 Void function

- Function name *should* start by a $
- Argument names *should* start by a $
- Void functions can call other void functions

Example:

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

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


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


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



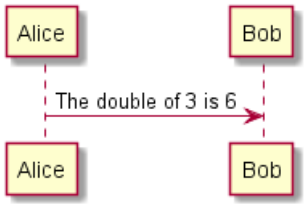
Variables defined in functions are **local**. It means that the variable is destroyed when the function is exited.

19.5 Return function

A return function does not output any text. It just define a function that you can call:

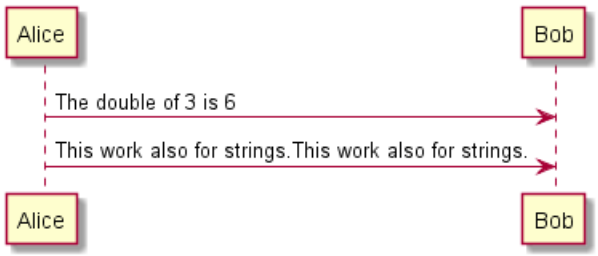
- directly in variable definition or in diagram text
- from other return function
- from other void function
- Function name *should* start by a $
- Argument names *should* start by a $

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

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



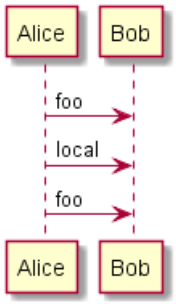
It is possible to shorten simple function definition in one line:

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

Alice -> Bob : The double of 3 is $double(3)
Alice -> Bob : $double("This work also for strings.")
@enduml
```



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

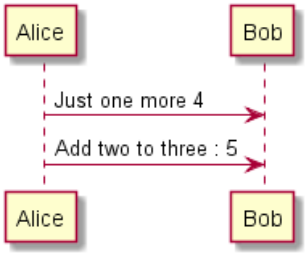
```
@startuml
!function $dummy()
!local $ijk = "local"
Alice -> Bob : $ijk
!endfunction

!global $ijk = "foo"

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



19.6 Default argument value

In both return and void function, you can define default value for argument.

```
@startuml
!function $inc($value, $step=1)
!if $step==0
!return $value
!endif
!return $value + $step
!endfunction

Alice -> Bob : Just one more $inc(3)
Alice -> Bob : Add two to three : $inc(3, 2)
@enduml
```

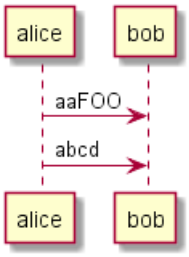


19.7 Unquoted function

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

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

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

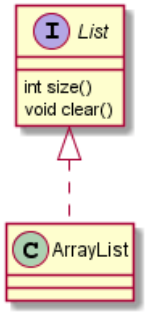


19.8 Including files or URL

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

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

```
@startuml

!include List.iuml
List <|.. ArrayList
@enduml
```



File List.iuml

```
interface List
List : int size()
List : void clear()
```

The file `List.iuml` can be included in many diagrams, and any modification in this file will change all diagrams that include it.

You can also put several `@startuml/@enduml` text block in an included file and then specify which block you want to include adding `!0` where `0` is the block number. The `!0` notation denotes the first diagram.

For example, if you use `!include foo.txt!1`, the second `@startuml/@enduml` block within `foo.txt` will be included.

You can also put an id to some `@startuml/@enduml` text block in an included file using `@startuml(id=MY_OWN_ID)` syntax and then include the block adding `!MY_OWN_ID` when including the file, so using something like `!include foo.txt!MY_OWN_ID`.

By default, a file can only be included once. You can use `!include_many` instead of `!include` if you want to include some file several times. Note that there is also a `!include_once` directive that raises an error if a file is included several times.

19.9 Including Subpart

You can also use `!startsub NAME` and `!endsub` to indicate sections of text to include from other files using `!includesub`. For example:

file1.puml:

```
@startuml

A -> A : stuff1
!startsub BASIC
B -> B : stuff2
!endsub
C -> C : stuff3
!startsub BASIC
D -> D : stuff4
!endsub
@enduml
```

file1.puml would be rendered exactly as if it were:

```
@startuml

A -> A : stuff1
B -> B : stuff2
C -> C : stuff3
D -> D : stuff4
@enduml
```

However, this would also allow you to have another file2.puml like this:

file2.puml

```
@startuml

title this contains only B and D
!includesub file1.puml!BASIC
@enduml
```

This file would be rendered exactly as if:

```
@startuml

title this contains only B and D
B -> B : stuff2
D -> D : stuff4
@enduml
```

19.10 Builtin functions

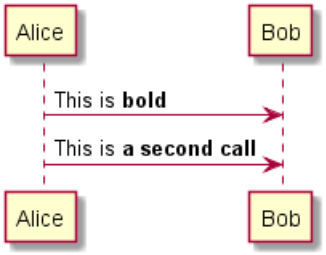
Some functions are defined by default. Their name starts by %

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

19.11 Logging

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

```
@startuml
!function bold($text)
!$result = "<b>"+ $text +"</b>"
!log Calling bold function with $text. The result is $result
!return $result
!endfunction

Alice -> Bob : This is bold("bold")
Alice -> Bob : This is bold("a second call")
@enduml
```

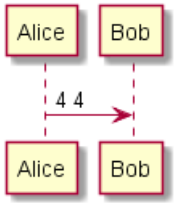


19.12 Memory dump

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

```
@startuml
!function $inc($string)
!$val = %intval($string)
!log value is $val
!dump_memory
!return $val+1
!endfunction

Alice -> Bob : 4 $inc("3")
```

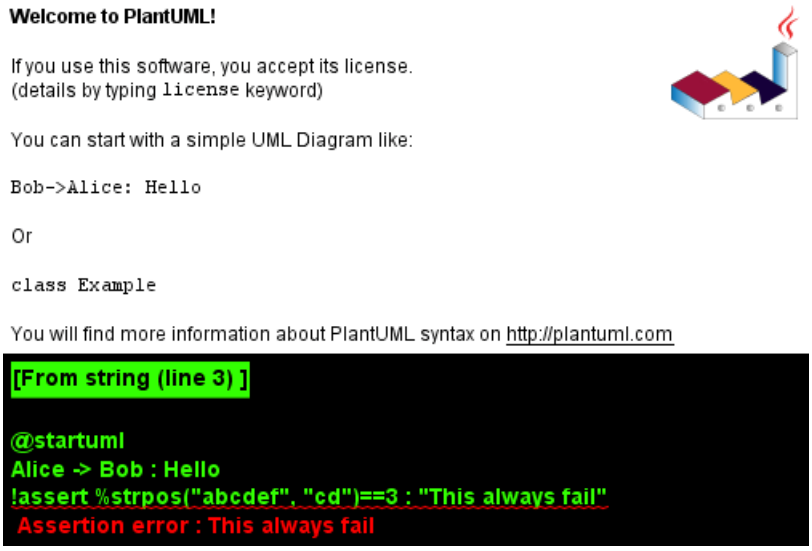
```
!unused = "foo"
!dump_memory EOF
@enduml
```



19.13 Assertion

You can put assertion in your diagram.

```
@startuml
Alice -> Bob : Hello
!assert %strpos("abcdef", "cd")==3 : "This always fail"
@enduml
```



19.14 Building custom library

It's possible to package a set of included files into a single .zip or .jar archive. This single zip/jar can then be imported into your diagram using `!import` directive.

Once the library has been imported, you can `!include` file from this single zip/jar.

Example:

```
@startuml

!import /path/to/customLibrary.zip
' This just adds "customLibrary.zip" in the search path

!include myFolder/myFile.iuml
' Assuming that myFolder/myFile.iuml is located somewhere
' either inside "customLibrary.zip" or on the local filesystem

...
```

19.15 Search path

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

For example:

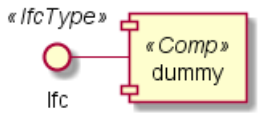
```
java -Dplantuml.include.path="c:/mydir" -jar plantuml.jar atest1.txt
```

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

19.16 Argument concatenation

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

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

COMP_TEXTGENCOMP(dummy)
@enduml
```

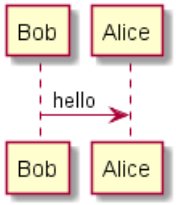


19.17 Dynamic function invocation

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

For example, you can have:

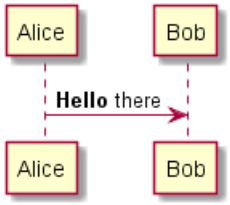
```
@startuml
!function $go()
 Bob -> Alice : hello
!endfunction

!$wrapper = "$go"

%invoke_void_func($wrapper)
@enduml
```



For return functions, you can use the corresponding special function `%call_user_func()` :

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

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



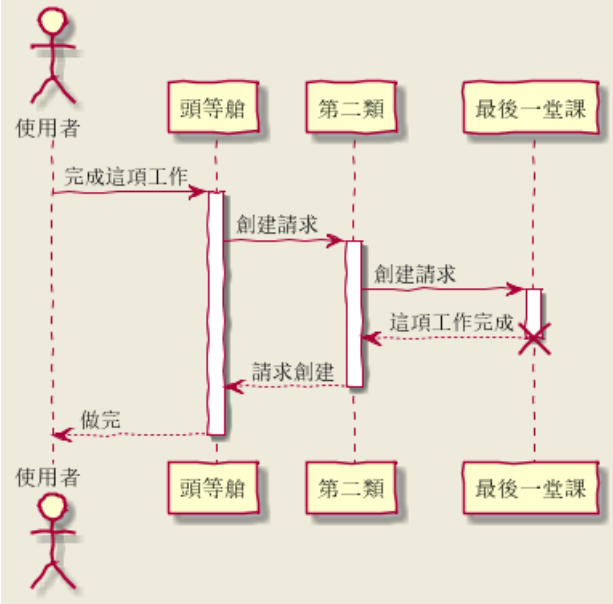
20 Unicode

The PlantUML language use *letters* to define actor, usecase and soon.

But *letters* are not only A-Z latin characters, it could be *any kind of letter from any language*.

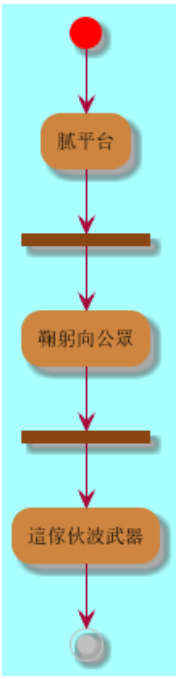
20.1 Examples

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

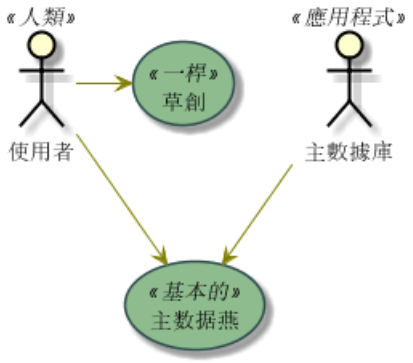



```
@startuml

(*) --> "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
```



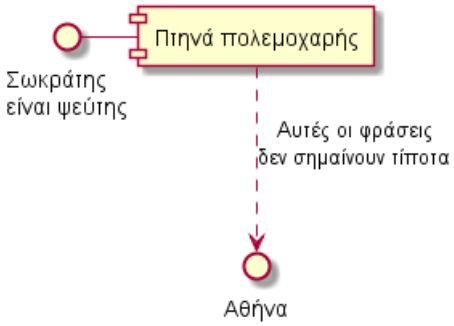
@startuml

() "Σωκράτηςψεύτης" as Σωκράτης

Σωκράτης - [Πτηνά πολεμοχαρής]

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

@enduml



20.2 Charset

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

Normally, it should just be fine, but in some case, you may want to 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`.

21 Standard Library

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

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

21.1 AWS library

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

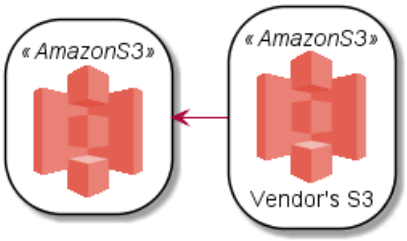
The AWS library consists of Amazon AWS icons, it provides icons of two different sizes.

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

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

Example of usage:

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

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



21.2 Azure library

https://github.com/RicardoNiepel/Azure-PlantUML/

The Azure library consists of Microsoft Azure icons.

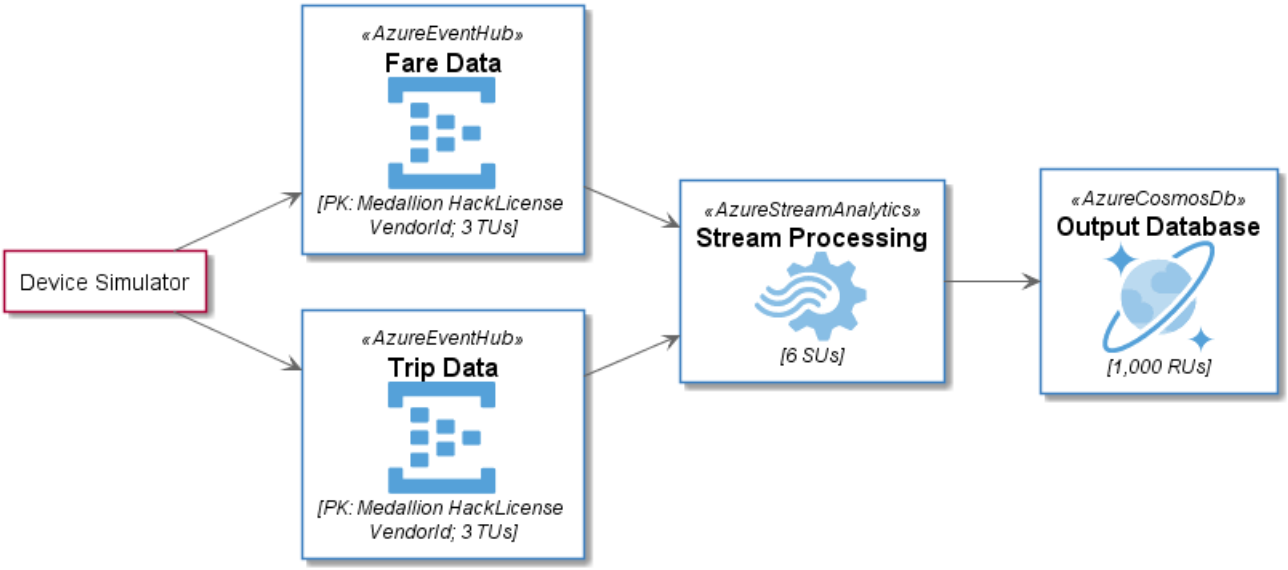
Use it by including the file that contains the sprite, eg: `!include <azure/Analytics/AzureEventHub.puml>`. When imported, you can use the sprite as normally you would, using `<$sprite_name>`.

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

Example of usage:

```
@startuml
!include <azure/AzureCommon.puml>
!include <azure/Analytics/AzureEventHub.puml>
!include <azure/Analytics/AzureStreamAnalytics.puml>
!include <azure/Databases/AzureCosmosDb.puml>

left to right direction
```

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

AzureEventHub(fareDataEventHub, "Fare Data", "PK: Medallion HackLicense VendorId; 3 TUs")
AzureEventHub(tripDataEventHub, "Trip Data", "PK: Medallion HackLicense VendorId; 3 TUs")
AzureStreamAnalytics(streamAnalytics, "Stream Processing", "6 SUs")
AzureCosmosDb(outputCosmosDb, "Output Database", "1,000 RUs")

devices --> fareDataEventHub
devices --> tripDataEventHub
fareDataEventHub --> streamAnalytics
tripDataEventHub --> streamAnalytics
streamAnalytics --> outputCosmosDb
@enduml
```



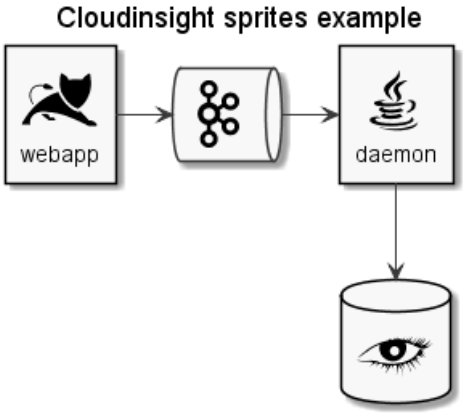
21.3 Cloud Insight

https://github.com/rabelenda/cicon-plantuml-sprites

This repository contains PlantUML sprites generated from Cloudinsight icons, which can easily be used in PlantUML diagrams for nice visual representation of popular technologies.

```
@startuml
!include <cloudinsight/tomcat>
!include <cloudinsight/kafka>
!include <cloudinsight/java>
!include <cloudinsight/cassandra>

title Cloudinsight sprites example

skinparam monochrome true

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

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

```
@enduml
```



21.4 Devicons and Font Awesome library

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

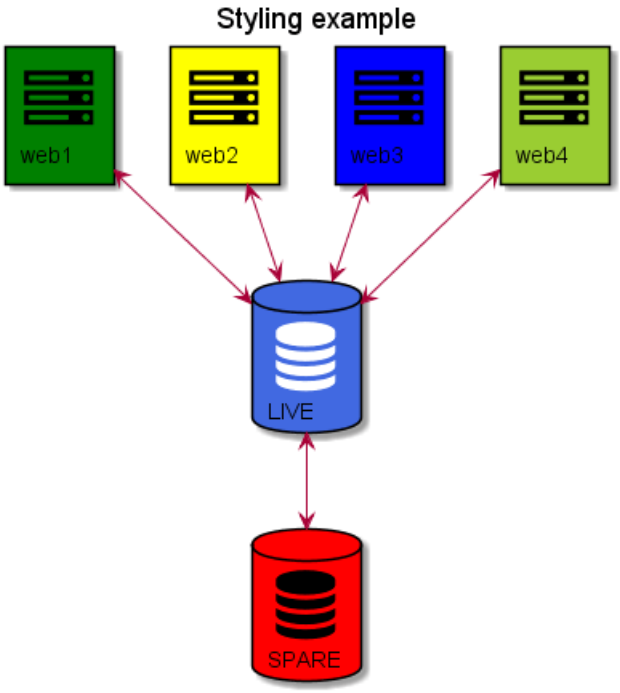
These two library consists respectively of Devicons and Font Awesome libraries of icons.

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

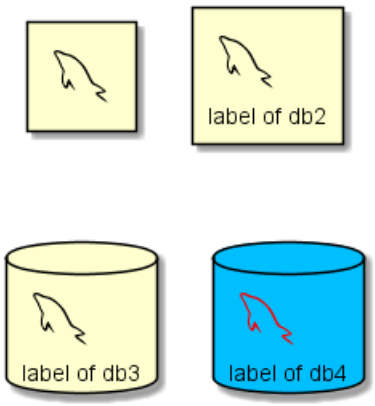
You may also include the `common.puml` file, eg: `!include <font-awesome/common>`, which contains helper macros defined. With the `common.puml` imported, you can use the `NAME_OF_SPRITE(parameters...)` macro.

Example of usage:

```
@startuml
!include <tupadr3/common>
!include <tupadr3/font-awesome/server>
!include <tupadr3/font-awesome/database>

title Styling example

FA_SERVER(web1,web1) #Green
FA_SERVER(web2,web2) #Yellow
FA_SERVER(web3,web3) #Blue
FA_SERVER(web4,web4) #YellowGreen

FA_DATABASE(db1,LIVE,database,white) #RoyalBlue
FA_DATABASE(db2,SPARE,database) #Red

db1 <--> db2

web1 <--> db1
web2 <--> db1
web3 <--> db1
web4 <--> db1
@enduml
```




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

DEV_MYSQL(db1)
DEV_MYSQL(db2,label of db2)
DEV_MYSQL(db3,label of db3,database)
DEV_MYSQL(db4,label of db4,database,red) #DeepSkyBlue
@enduml
```



21.5 Google Material Icons

https://github.com/Templarian/MaterialDesign

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

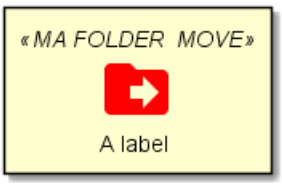
Use it by including the file that contains the sprite, eg: `!include <material/ma_folder_move>`. When imported, you can use the sprite as normally you would, using `<$ma_sprite_name>`. Notice that this library requires an `ma_` preffix on sprites names, this is to avoid clash of names if multiple sprites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

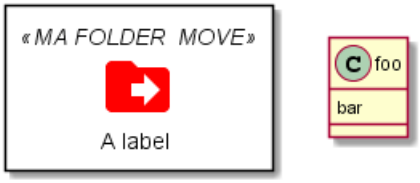


Notes

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
```



21.6 Office

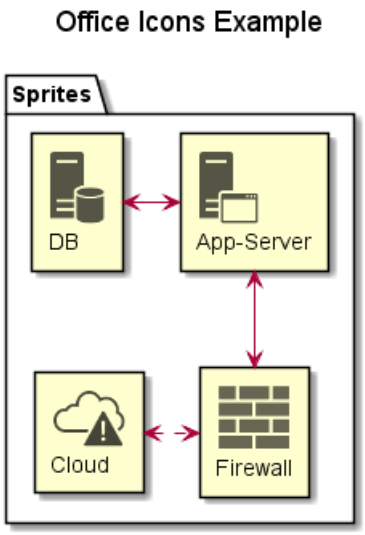
https://github.com/Roemer/plantuml-office

There are sprites (*.puml) and colored png icons available. Be aware that the sprites are all only 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.

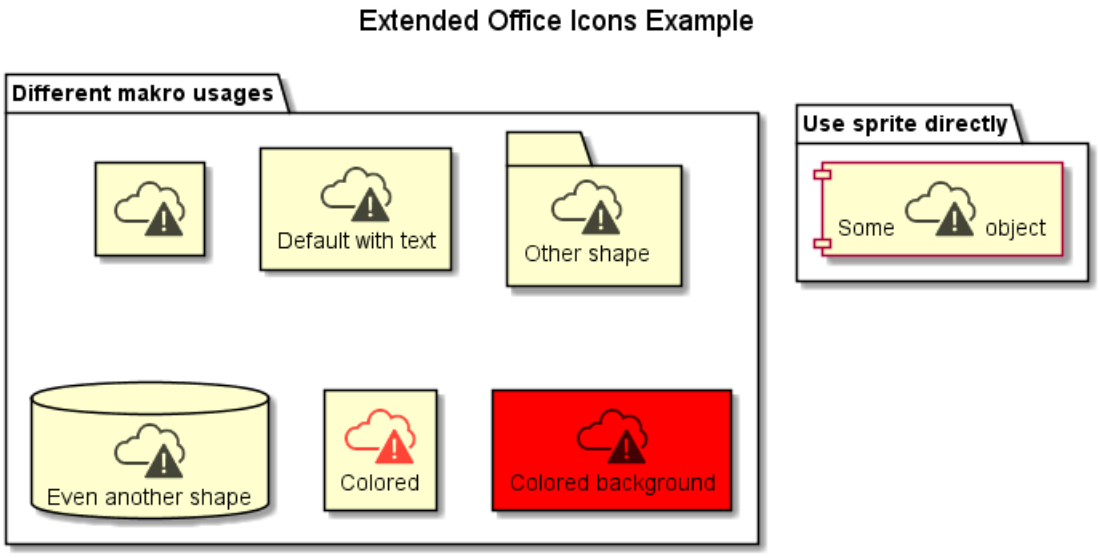
Example of usage:

```
@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
```




```
@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
```



21.7 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
```

```
[From string (line 24) ]

@startuml Internet Browser Example


skinparam defaultTextAlignment center
skinparam wrapWidth 400
...
... ( skipping 48 lines )
...


title Archimate Sample - Internet Browser

archimate #BUSINESS "A Business Object" <<business-object>> as businessObject
archimate #BUSINESS "Some Business Process" <<business-process>> as someBusinessProcess
archimate #BUSINESS "IT Support for Business (Application Service)" <<business-service>> as itSupportService

archimate #APPLICATION "Web Page Data \n 'on the fly'" <<application-data-object>> as dataObject
archimate #APPLICATION "Web page behaviour" <<application-function>> as webpageBehaviour
archimate #APPLICATION "Active Part of the web page \n 'on the fly'" <<application-component>> as ActivePartWebPage

archimate #TECHNOLOGY "in memory / 'on the fly' html/javascript" <<technology-artifact>> as inMemoryItem
archimate #TECHNOLOGY "Internet Browser Generic & Plugin" <<technology-infra-service>> as internetBrowser
archimate #TECHNOLOGY "Some Internet Browser Plugin" <<technology-infra-service>> as internetBrowserPlugin
archimate #TECHNOLOGY "Some web server" <<technology-infra-service>> as webServer

someBusinessProcess .LEFT.>> businessObject : ""
itSupportService -UP-> someBusinessProcess : ""
Rel_Specilization_Up(webpageBehaviour, itSupportService, "")
 Syntax Error?
```

21.8 Miscellaneous

You can list standard library folders using the special diagram:

```
@startuml
stdlib
@enduml
```

aws
Version 18.02.22
Delivered by https://github.com/milo-minderbinder/AWS-PlantUML

awslib
Version 0.0.1
Delivered by https://github.com/awslabs/aws-icons-for-plantuml

azure
Version 2.1.0
Delivered by https://github.com/RicardoNiepel/Azure-PlantUML

c4
Version 1.0.0
Delivered by https://github.com/RicardoNiepel/C4-PlantUML

cloudinsight
Version 0.0.1
Delivered by https://github.com/rabelenda/cicon-plantuml-sprites/

cloudogu
Version 0.0.1
Delivered by https://github.com/cloudogu/plantuml-cloudogu-sprites

material
Version 0.0.1
Delivered by https://github.com/Templarian/MaterialDesign

office
Version 0.0.1
Delivered by https://github.com/Roemer/plantuml-office

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