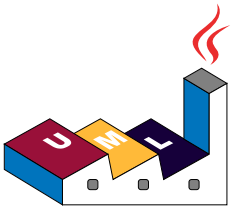
Dessiner de l'UML avec PlantUML



Guide de référence du langage PlantUML

(Version 1.2019.9)

PlantUML est un composant qui permet de dessiner rapidement des:

- diagrammes de séquence
- diagrammes de cas d'utilisation
- diagrammes de classes
- diagrammes d'activité
- diagrammes de composant
- diagrammes d'état
- diagrammes d'objet
- diagrammes de déploiement
- diagrammes de temps

Certains autres diagrammes (hors UML) sont aussi possibles:

- maquette d'interface graphique
- diagrammes Archimate
- Specification and Description Language (SDL)
- diagrammes Ditaa
- diagrammes de Gantt
- MindMap diagram
- Work Breakdown Structure diagram
- notation mathématique avec AsciiMath ou JLaTeXMath

Les diagrammes sont définis à l'aide d'un langage simple et intuitif.

1 Diagramme de séquence

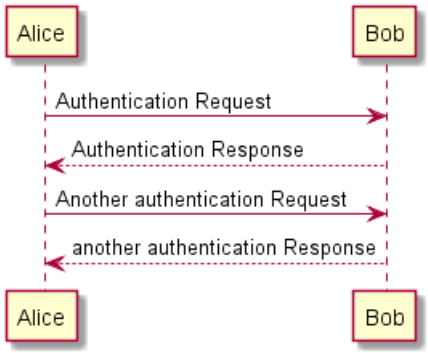
1.1 Exemples de base

Le symbole `->` est utilisé pour dessiner un message entre deux participants. Les participants n'ont pas besoin d'être explicitement déclarés.

Pour avoir une flèche en pointillée, il faut utiliser `-->`.

Il est aussi possible d'utiliser `<-` et `<--`. Cela ne change pas le dessin, mais cela peut améliorer la lisibilité du texte source. Ceci est uniquement vrai pour les diagrammes de séquences, les règles sont différentes pour les autres diagrammes.

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

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



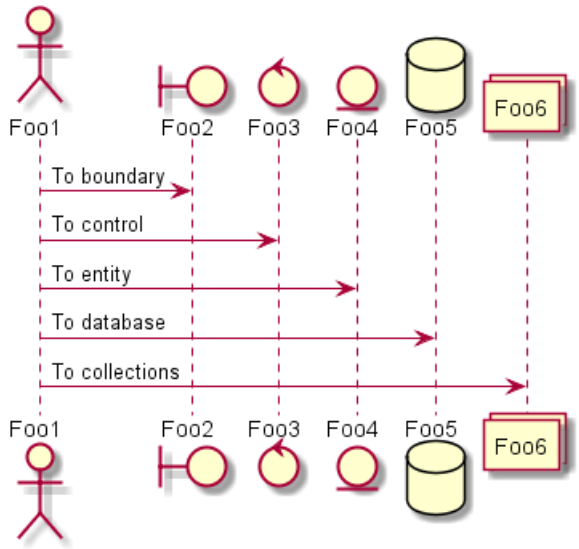
1.2 Déclaration de participants

Il est possible de changer l'ordre des participants à l'aide du mot clé `participant`.

Il est aussi possible d'utiliser d'autres mot-clés pour déclarer un participant :

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

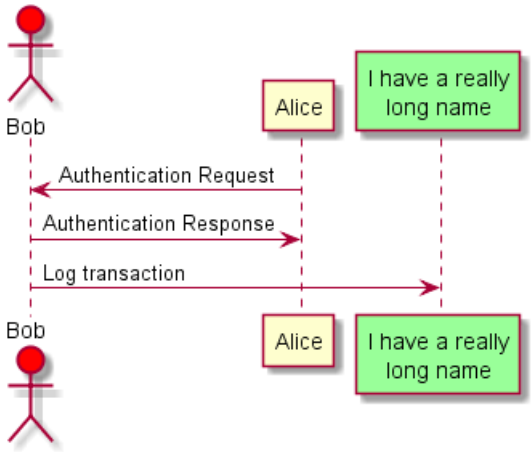
```
@startuml
actor Foo1
boundary Foo2
control Foo3
entity Foo4
database Foo5
collections Foo6
Foo1 -> Foo2 : To boundary
Foo1 -> Foo3 : To control
Foo1 -> Foo4 : To entity
Foo1 -> Foo5 : To database
Foo1 -> Foo6 : To collections

@enduml
```



On peut aussi utiliser un nom court à l'aide grâce au mot-clé `as`.

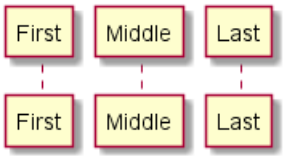
La couleur d'un acteur ou d'un participant peut être définie avec son code ou son nom HTML.

```
@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
```



Vous pouvez utiliser le mot-clé `order` pour modifier l'ordre des participants

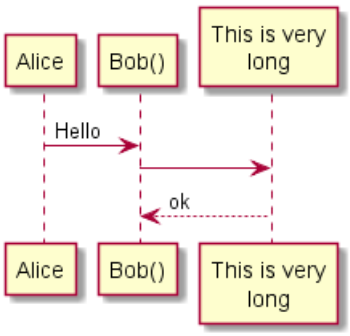
```
@startuml
participant Last order 30
participant Middle order 20
participant First order 10
@enduml
```



1.3 Caractères non alphanumérique dans les participants

Si vous voulez mettre des charactères non alphanumériques, il est possible d'utiliser des guillemets. Et on peut utiliser le mot clé `as` pour définir un alias pour ces participants.

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

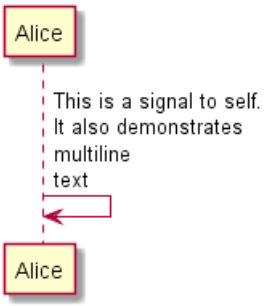


1.4 Message à soi-même

Un participant peut très bien s'envoyer un message.

Il est possible de mettre un message sur plusieurs lignes grâce à `\n`.

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



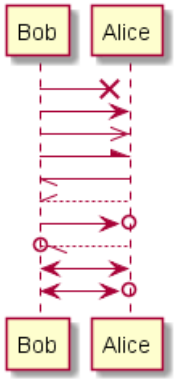
1.5 Autre style de flèches

Vous pouvez changer les flèches de plusieurs façons :

- Pour indiquer un message perdu, terminer la flèche avec `x`
- Utiliser `\` ou `/` à la place de `<` ou `>` pour avoir seulement la partie supérieure ou inférieure de la flèche.
- Doubler un des caractères (par exemple, `>>` ou `//`)pour avoir une flèche plus fine.

- Utiliser -- à la place de - pour avoir des pointillés.
- Utiliser "o" après la flèche
- Utiliser une flèche bi-directionnelle <->

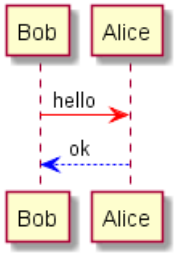
```
@startuml
Bob ->x Alice
Bob -> Alice
Bob ->> Alice
Bob -\ Alice
Bob \\- Alice
Bob //-- Alice

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

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



1.6 Changer la couleur des flèches

Changer la couleur d'une flèche ainsi:

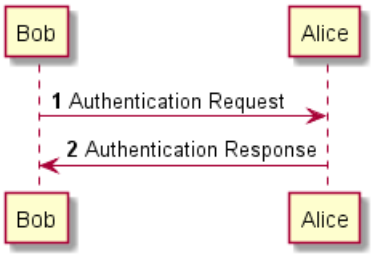
```
@startuml
Bob -[#red]> Alice : hello
Alice -[#0000FF]->Bob : ok
@enduml
```



1.7 Numérotation automatique des messages

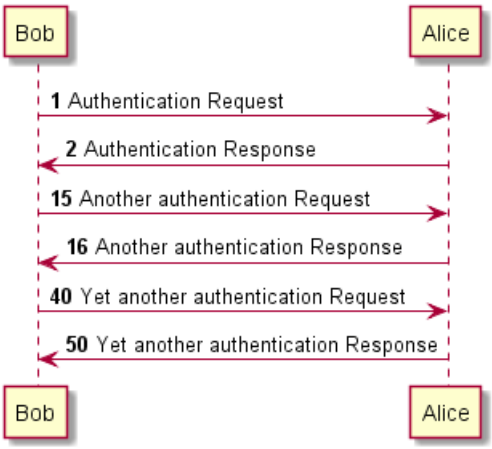
Le mot clé `autonumber` est utilisé pour ajouter automatiquement des numéros aux messages.

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



Spécifier le numéro de départ avec `autonumber` *`start`* , et l'incrément avec `autonumber` *`start increment`*.

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

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

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

@enduml
```

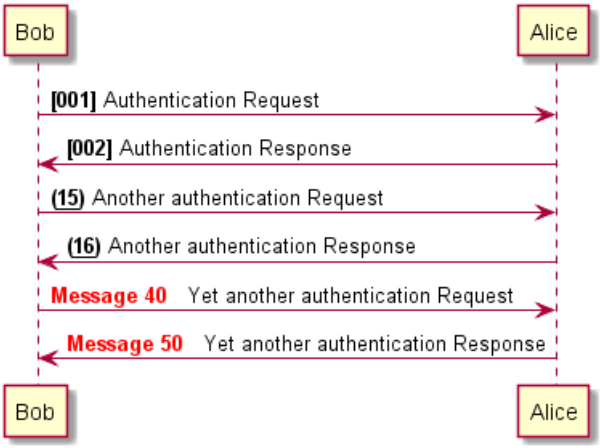


Spécifier le format d'un nombre entre guillemets anglais.

Le formatage est fait par la classe `DecimalFormat` (0 signifie un chiffre, # signifie un chiffre ou zéro si absent).

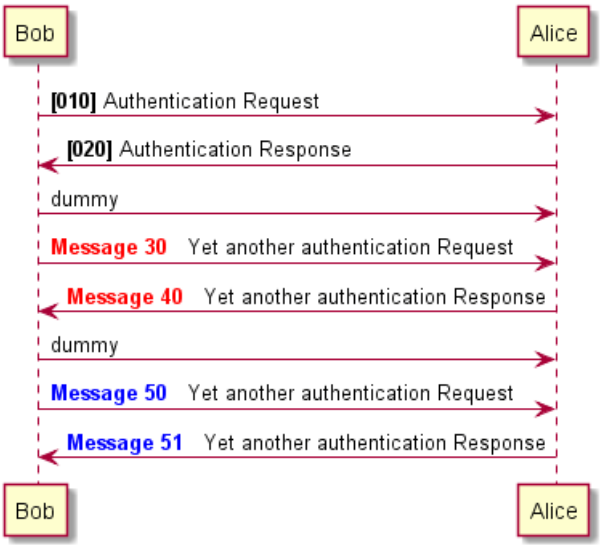
Des balises HTML sont permises dans le format.

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

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

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

@enduml
```



Vous pouvez utiliser `autonumber` *`stop`* et `autonumber resume` *`increment format`* pour respectivement arrêter et reprendre la numérotation automatique.

```
@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
```

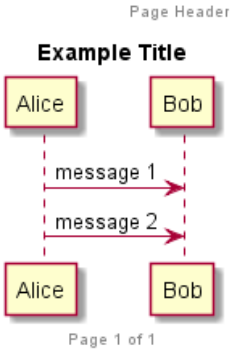


1.8 Page Title, Header and Footer

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
```



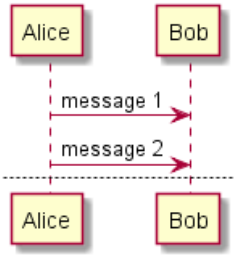
1.9 Découper un diagramme

Le mot clé `newpage` est utilisé pour découper un digramme en plusieurs images.

Vous pouvez mettre un titre pour la nouvelle page juste après le mot clé `newpage`.

Ceci est très pratique pour mettre de très longs digrammes sur plusieurs pages.

```
@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 Regrouper les messages (cadres UML)

Il est possible de regrouper les messages dans un cadre UML à l'aide d'un des mot clés suivants:

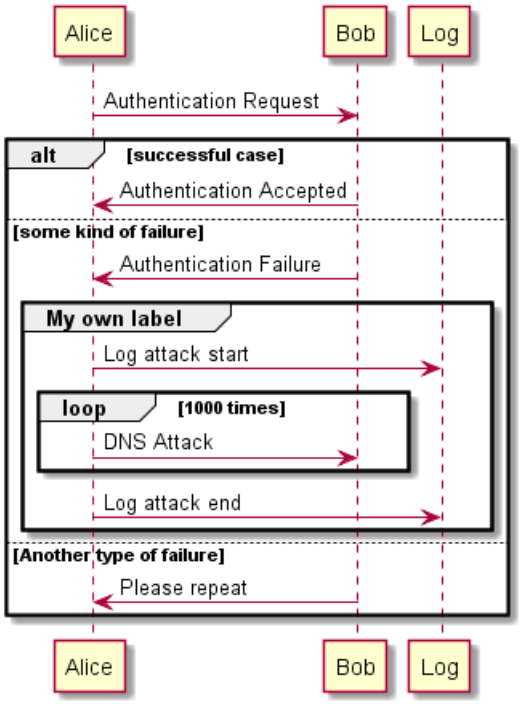
- `alt/else`
- `opt`
- `loop`
- `par`
- `break`
- `critical`
- `group`, suivi par le texte à afficher

Il est aussi possible de mettre un texte à afficher dans l'entête. Le mot-clé `end` est utiliser pour fermer le groupe. Il est aussi possible d'imbriquer les groupes.

Terminer le cadre avec le mot-clé `end`.

Il est possible d'imbriquer les cadres.

```
@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
```

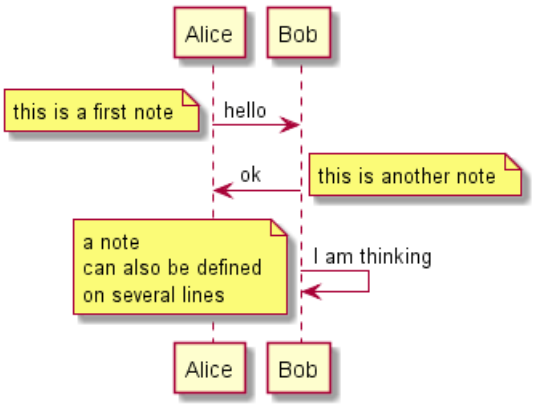


1.11 Note sur les messages

Pour attacher une note à un message, utiliser les mots-clés `note left` (pour une note à gauche) ou `note right` (pour une note à droite) *juste après le message*.

Il est possible d'avoir une note sur plusieurs lignes avec le mot clé `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
```



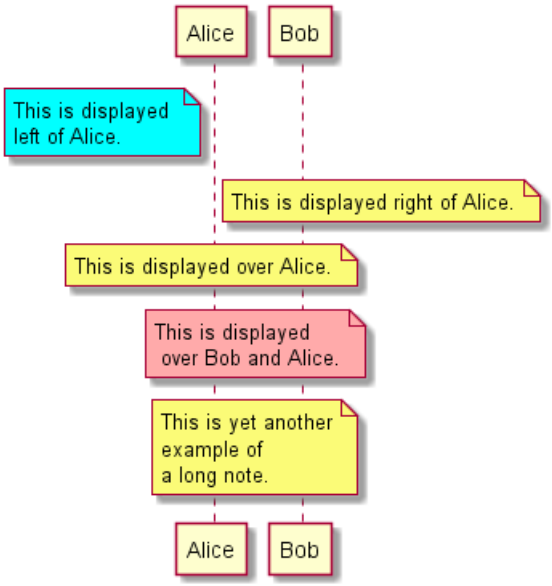
1.12 Encore plus de notes

Il est aussi possible de mettre des notes placées par rapport aux participants.

Il est aussi possible de faire ressortir une note en changeant sa couleur de fond.

On peut aussi avoir des notes sur plusieurs lignes à l'aide du mot clé `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
```

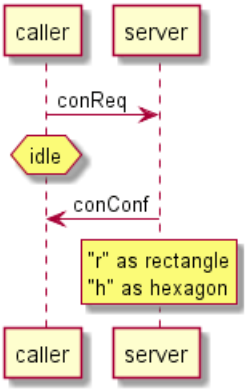


1.13 Changer l'aspect des notes

Vous pouvez préciser la forme géométrique des notes. \n (`rnote` : rectangulaire, ou `hnote` : hexagonale)

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

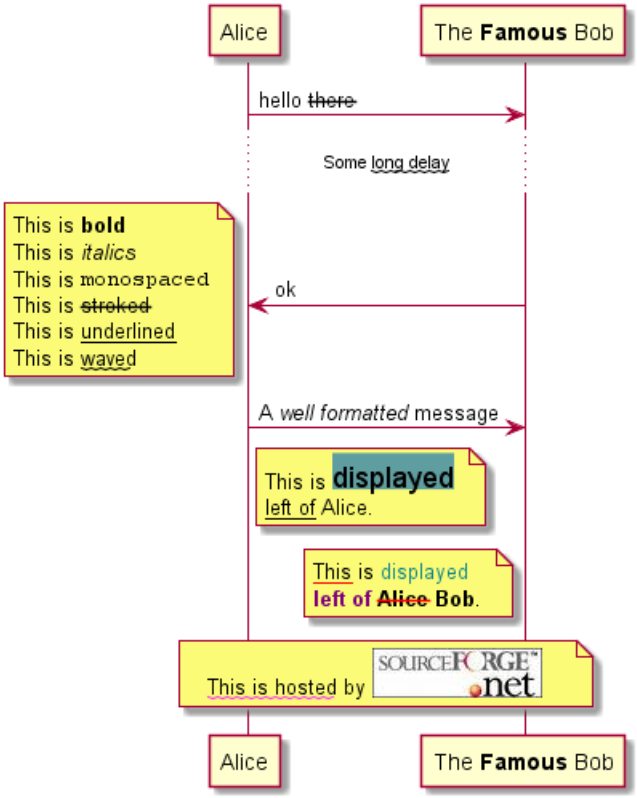
```
@enduml
```



1.14 Créole (langage de balisage léger) et HTML

Il est également possible d'utiliser le formatage créole (langage de balisage léger):

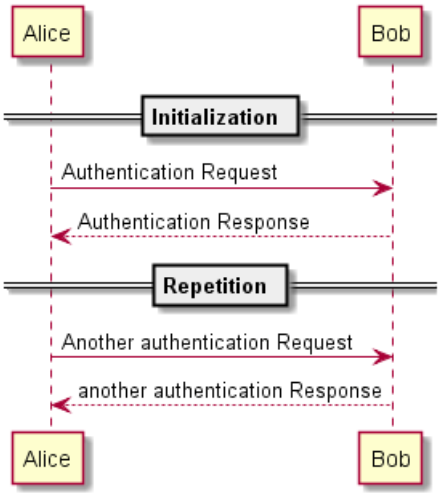
```
@startuml
participant Alice
participant "The **Famous** Bob" as Bob

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 Séparation

Si vous voulez, vous pouvez séparer le diagramme avec l'aide de "==" en étapes logiques.

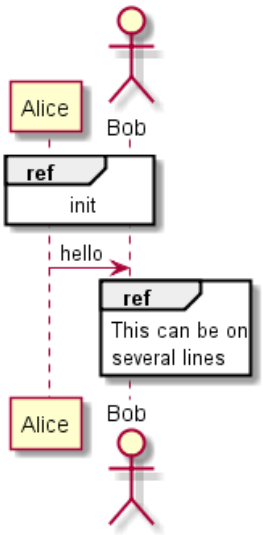
```
@startuml

== Initialization ==

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

== Repetition ==

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

@enduml
```



1.16 Référence

Vous pouvez ajouter des références dans un diagramme, en utilisant le mot-clé `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
```

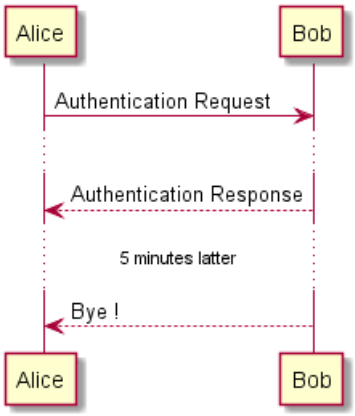


1.17 Retard

Utiliser `...` pour indiquer le passage de temps arbitraire dans le diagramme. Un message peut être associé à un retard.

```
@startuml
```

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

@enduml
```

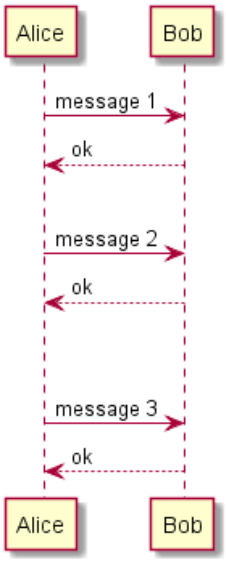


1.18 Séparation verticale

Utiliser `|||` pour créer un espace vertical dans le diagramme.

Il est également possible de spécifier un nombre de pixels pour la séparation verticale.

```
@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 Lignes de vie

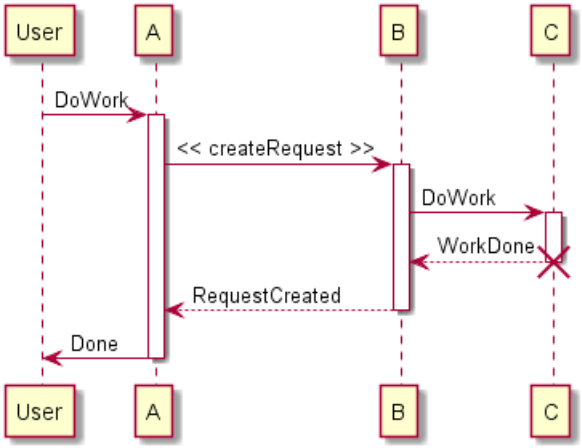
Vous pouvez utiliser `activate` et `deactivate` pour marquer l'activation des participants.

Une fois qu'un participant est activé, sa ligne de vie apparaît.

Les ordres `activate` et `deactivate` s'applique sur le message situé juste avant.

Le mot clé `destroy` sert à montrer la fin de vie d'un participant.

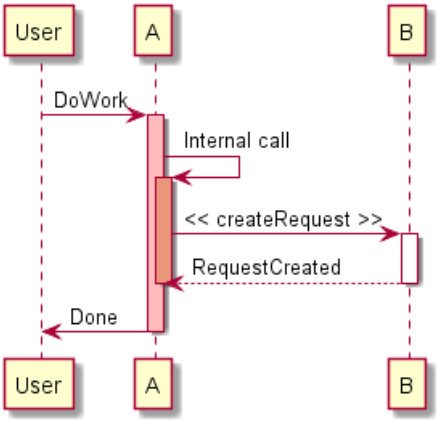
```
@startuml
participant User

User -> A: DoWork
activate A

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

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

B --> A: RequestCreated
deactivate B

A -> User: Done
deactivate A

@enduml
```



Les lignes de vie peuvent être imbriquées, et il est possible de les colorer.

```
@startuml
participant User

User -> A: DoWork
activate A #FFBBBB

A -> A: Internal call
activate A #DarkSalmon

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

B --> A: RequestCreated
```

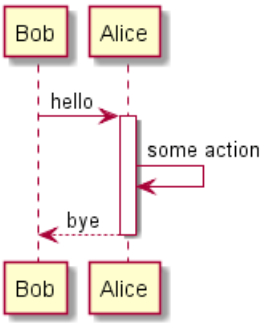
```
deactivate B
deactivate A
A -> User: Done
deactivate A

@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
```

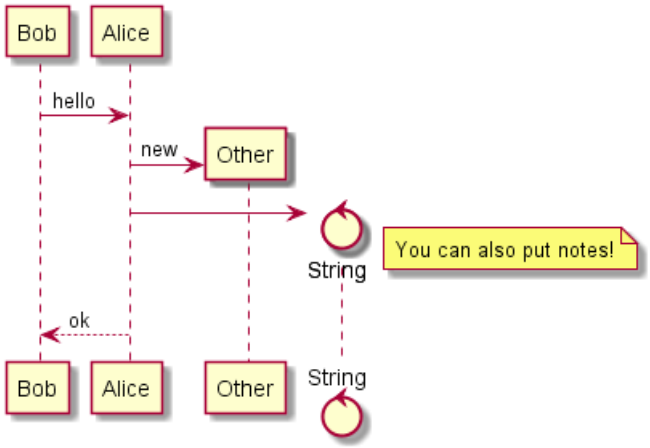


1.21 Création de participants.

Vous pouvez utiliser le mot clé `create` juste avant la première réception d'un message pour montrer que le message en question est une *création* d'un nouvelle objet.

```
@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
```

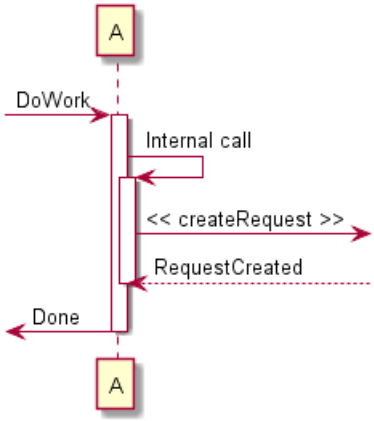


1.22 Messages entrant et sortant

Vous pouvez utiliser des flèches qui viennent de la droite ou de la gauche pour dessiner un sous-diagramme.

Il faut utiliser des crochets pour indiquer la gauche "[" ou la droite "]" du diagramme.

```
@startuml
[-> A: DoWork

activate A

A -> A: Internal call
activate A

A ->] : << createRequest >>

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



Vous pouvez aussi utiliser la syntaxe suivante:

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

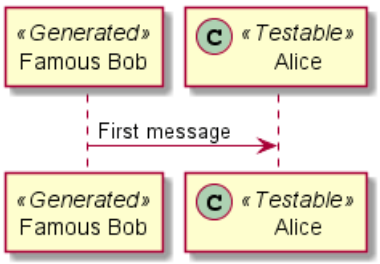
```
[o-> Bob
[o->o Bob
[x-> Bob

[<- Bob
[x<- Bob

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

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

1.23 Stéréotypes et décoration

Il est possible de rajouter un stéréotype aux participants en utilisant "<<" et ">>".

Dans le stéréotype, vous pouvez ajouter un caractère entouré d'un cercle coloré en utilisant la syntaxe `(X,couleur)`.

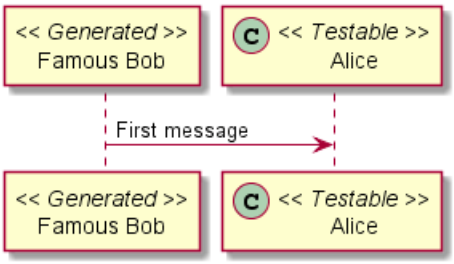
```
@startuml

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

Bob->Alice: First message

@enduml
```



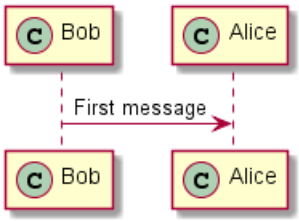
Par défaut, le caractère *guillemet* est utilisé pour afficher les stéréotypes. Vous pouvez changer ce comportement en utilisant la propriété skinparam `guillemet`:

```
@startuml

skinparam guillemet false
```

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

Bob->Alice: First message

@enduml
```



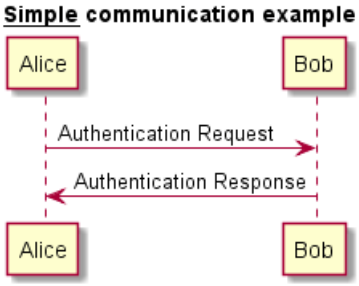

```
@startuml

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

Bob->Alice: First message

@enduml
```



1.24 Plus d'information sur les titres

Vous pouvez utiliser le formatage creole dans le titre.

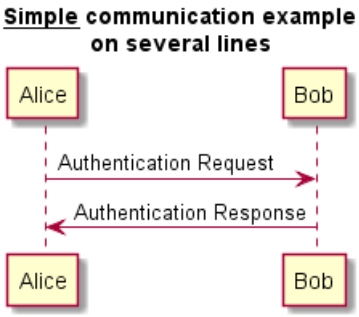
```
@startuml

title __Simple__ **communication** example

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

@enduml
```



Vous pouvez mettre des retours à la ligne en utilisant `\n` dans la description.

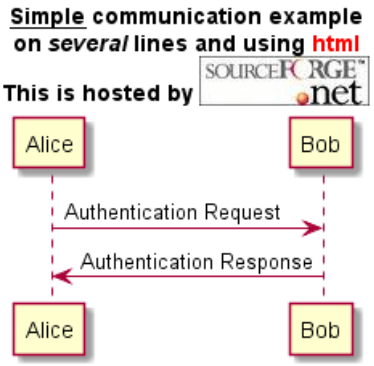
```
@startuml

title __Simple__ communication example\non several lines
```

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

@enduml
```



Vous pouvez aussi mettre un titre sur plusieurs lignes à l'aide des mots-clé `title` et `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
```



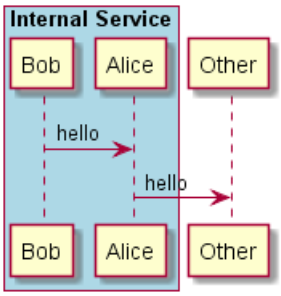
1.25 Cadre pour les participants

Il est possible de dessiner un cadre autour de certains participants, en utilisant les commandes `box` et `end box`.

Vous pouvez ajouter un titre ou bien une couleur de fond après le mot-clé `box`.

```
@startuml

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

Bob -> Alice : hello
```

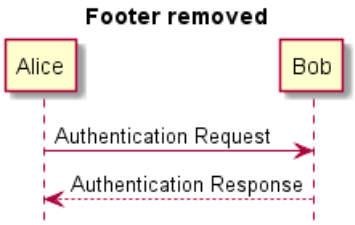
```
Alice -> Other : hello

@enduml
```



1.26 Supprimer les en-pieds

Vous pouvez utiliser le mot-clé `hide footbox` pour supprimer la partie basse du diagramme.

```
@startuml

hide footbox
title Footer removed

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

@enduml
```



1.27 Personnalisation

Utilisez la commande `skinparam` pour changer la couleur et la mise en forme du texte du schéma.

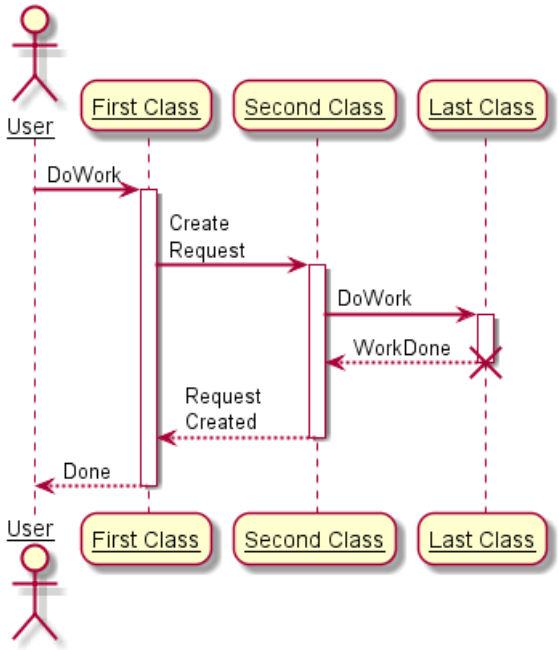
Vous pouvez utiliser cette commande :

- Dans la définition du diagramme, comme pour les autres commandes,
- Dans un fichier inclus,
- Dans un fichier de configuration, renseigné dans la ligne de commande ou la tâche ANT.

Vous pouvez aussi modifier d'autres paramètres pour le rendu, comme le montrent les exemples suivants:

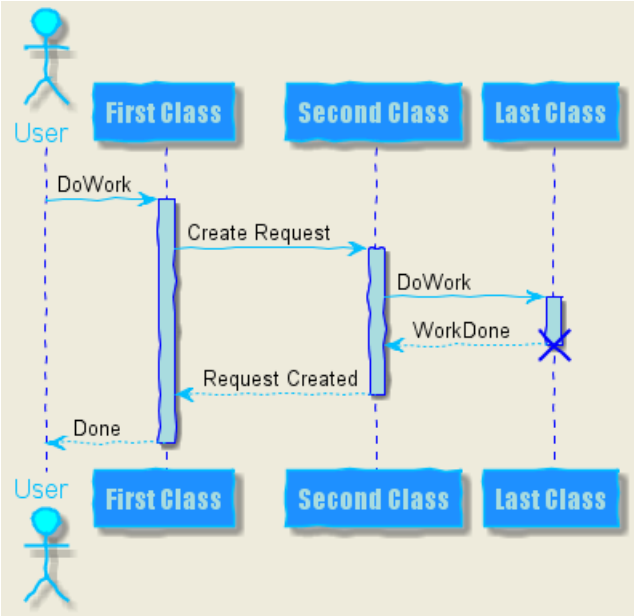
```
@startuml
skinparam sequenceArrowThickness 2
skinparam roundcorner 20
skinparam maxmessagesize 60
skinparam sequenceParticipant underline

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
```

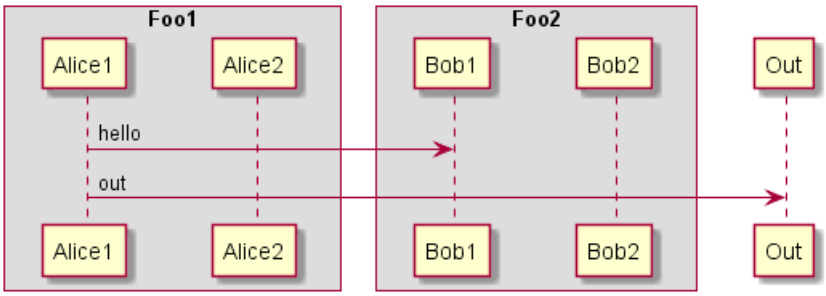


1.28 Changer le padding

Il est possible de changer certains paramètres du padding.

```
@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 Diagramme de cas d'utilisation

Let's have few examples :

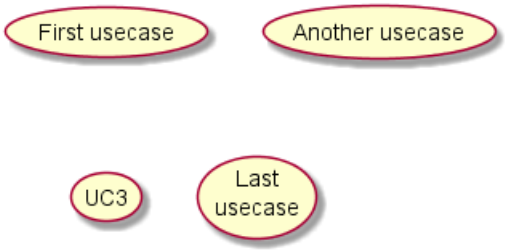
Note that you can disable the shadowing using the `skinparam shadowing false` command.

2.1 Cas d'utilisation

Les cas d'utilisation sont mis entre parenthèses (car deux parenthèses forment un ovale).

Vous pouvez aussi utiliser le mot-clé `usecase` pour définir un cas d'utilisation. Et vous pouvez définir un alias avec le mot-clé `as`. Cet alias sera ensuite utilisé lors de la définition des relations.

```
@startuml

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

@enduml
```



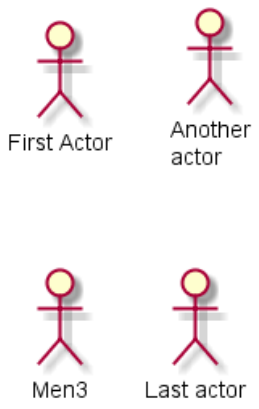
2.2 Acteurs

Un Acteur est encadré par des deux points.

Vous pouvez aussi utiliser le mot-clé `actor` pour définir un acteur. Et vous pouvez définir un alias avec le mot-clé `as`. Cet alias sera ensuite utilisé lors de la définition des relations.

Nous verrons que la définition des acteurs est optionnelle.

```
@startuml

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

@enduml
```

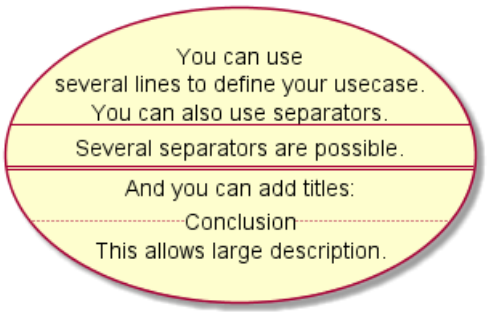


2.3 Description des cas d'utilisation

Si vous voulez une description sur plusieurs lignes, vous pouvez utiliser des guillemets.

Vous pouvez aussi utiliser les séparateurs suivants: `--` `..` `==` `__`. Et vous pouvez mettre un titre dans les séparateurs.

```
@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 Exemples très simples

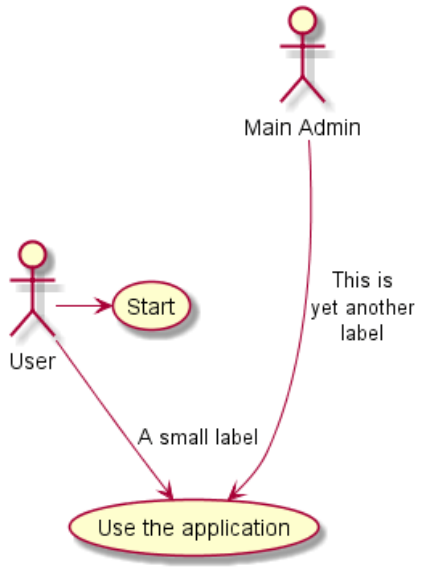
Pour lier les acteurs et les cas d'utilisation, la flèche `-->` est utilisée.

Plus il y a de tirets `-` dans la flèche, plus elle sera longue. Vous pouvez ajouter un libellé sur la flèche, en ajoutant un caractère `:` dans la définition de la flèche.

Dans cet exemple, vous voyez que *User* n'a pas été défini préalablement, et qu'il est implicitement reconnu comme acteur.

```
@startuml

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

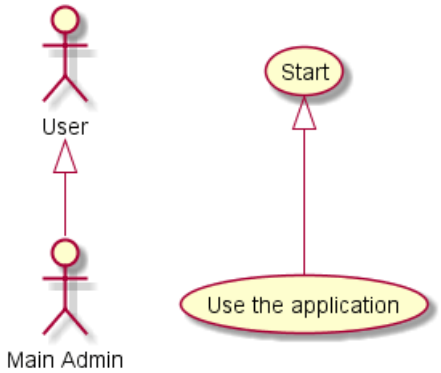
```
:Main Admin: ---> (Use the application) : This is\nyet another\nlabel

@enduml
```



2.5 Héritage

Si un acteur ou un cas d'utilisation en étend un autre, vous pouvez utiliser le symbole `<|--`.

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

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

@enduml
```



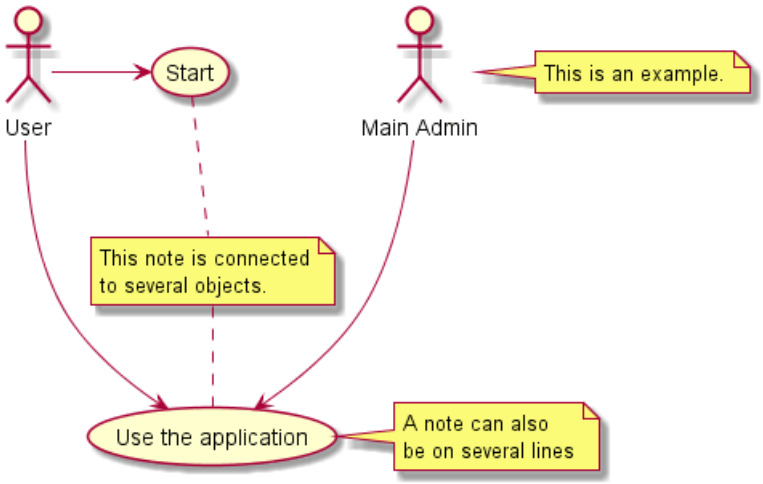
2.6 Notes

Vous pouvez utiliser les mots clés `note left of` , `note right of` , `note top of` , `note bottom of` pour définir les notes en relation avec un objet.

Une note peut également être définie seule avec des mots-clés, puis liée à d'autres objets en utilisant le symbole `..`.

```
@startuml
:Main Admin: as Admin
```

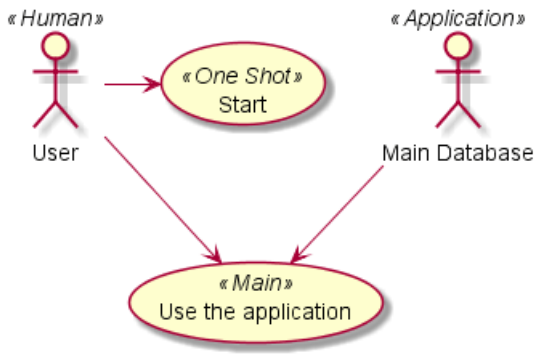
```
(Use the application) as (Use)

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 Stéréotypes

Vous pouvez ajouter des stéréotypes à la définition des acteurs et des cas d'utilisation avec << et >>.

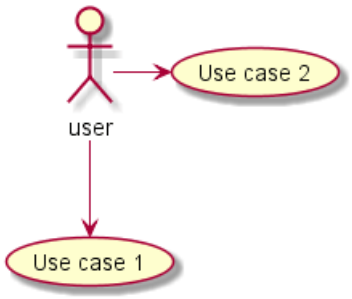
```
@startuml
User << Human >>
:Main Database: as MySql << Application >>
(Start) << One Shot >>
(Use the application) as (Use) << Main >>

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

MySql --> (Use)

@enduml
```



2.8 Changer les directions des flèches

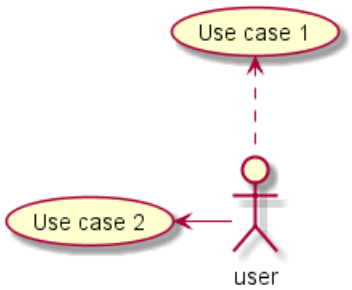
Par défaut, les liens entre les classes ont deux tirets `--` et sont orientés verticalement. Il est possible de mettre des liens horizontaux en mettant un seul tiret (ou un point) comme ceci:

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



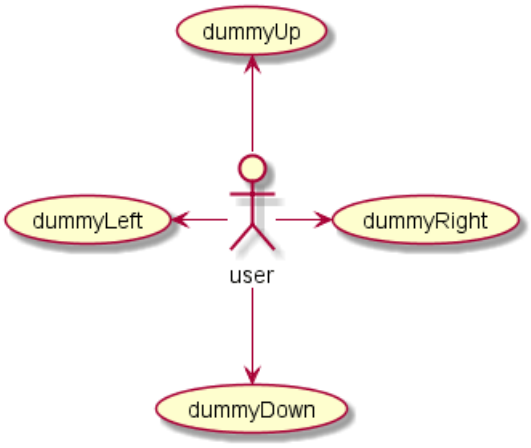
Vous pouvez aussi changer le sens en renversant le lien :

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



Il est possible de changer la direction d'une flèche en utilisant les mots-clé `left`, `right`, `up` ou `down` à l'intérieur de la flèche :

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



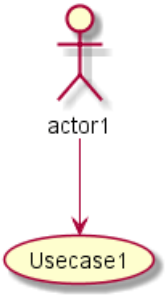
Vous pouvez abréger les noms des flèches en indiquant seulement le premier caractère de la direction (par exemple `-d-` pour `-down-`) ou les deux premiers caractères (`-do-`).

Il est conseillé de ne pas abuser de cette fonctionnalité : *Graphviz* qui donne d'assez bon résultats quoique non "garantis".

2.9 Découper les diagrames

Le mot-clé `newpage` est utilisé pour découper un diagrame en plusieurs images.

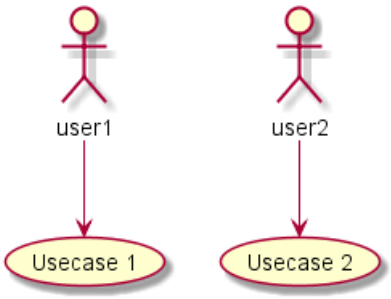
```
@startuml
:actor1: --> (Usecase1)
newpage
:actor2: --> (Usecase2)
@enduml
```



2.10 De droite à gauche

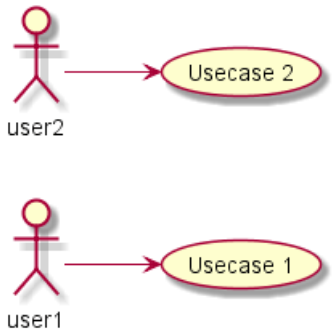
Le comportement général de construction des diagrammes est de haut en bas.

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

@enduml
```



Il est possible de changer pour aller plutôt de la droite vers la gauche avec la commande `left to right direction`. Le résultat est parfois meilleur dans ce cas.

```
@startuml

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

@enduml
```



2.11 La commande Skinparam

Utilisez la commande `skinparam` pour changer la couleur et la mise en forme du texte du schéma.

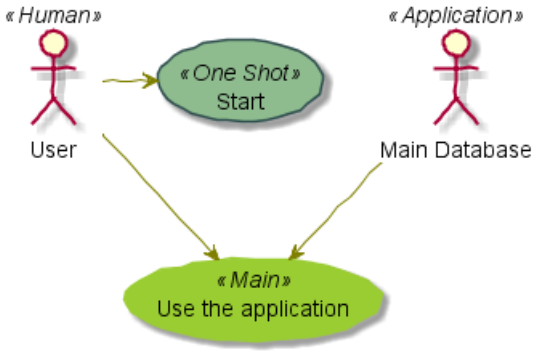
Vous pouvez utiliser cette commande :

- Dans la définition du diagramme, comme pour les autres commandes,
- Dans un fichier inclus,
- Dans un fichier de configuration, renseigné dans la ligne de commande ou la tâche ANT.

Vous pouvez aussi spécifier les polices et les couleurs pour les acteurs et cas d'utilisation avec des stéréotypes.

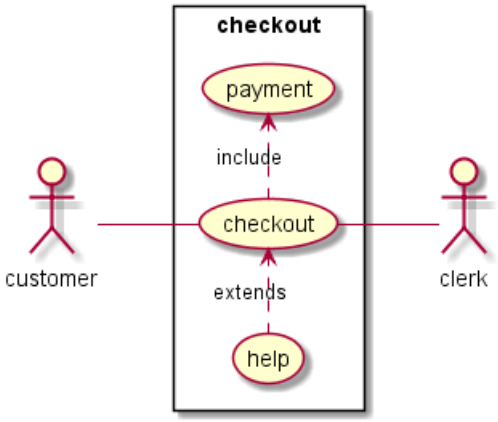
```
@startuml
skinparam handwritten true

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 Exemple complet

```
@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 Diagramme de classes

3.1 Relations entre classes

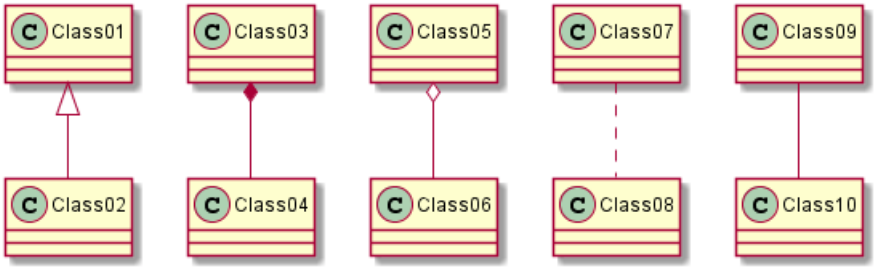
Les relations entre les classes sont définies en utilisant les symboles suivants :

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

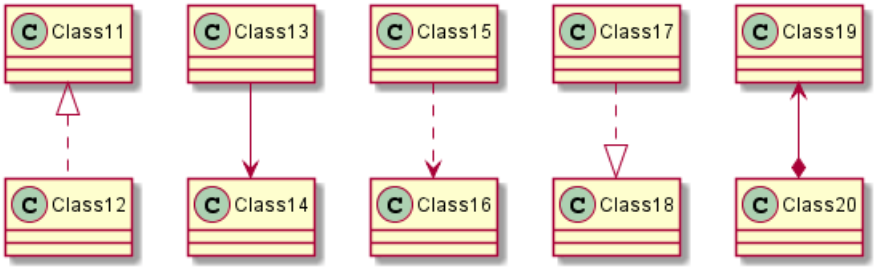
Il est possible de substituer `--` par `..` pour obtenir une ligne en pointillée.

Grâce à ces règles, il est possible de faire les diagrammes suivants :

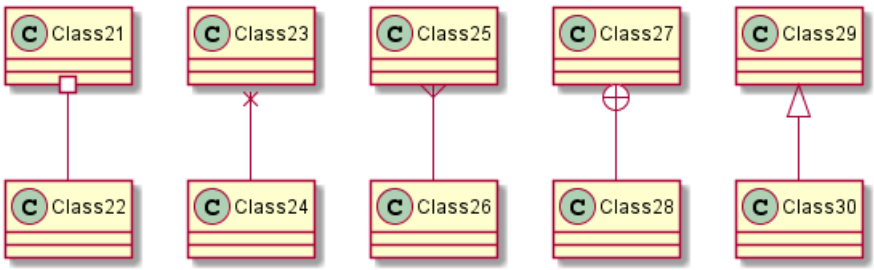
```
@startuml
Class01 <|-- Class02
Class03 *-- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10
@enduml
```




```
@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
```

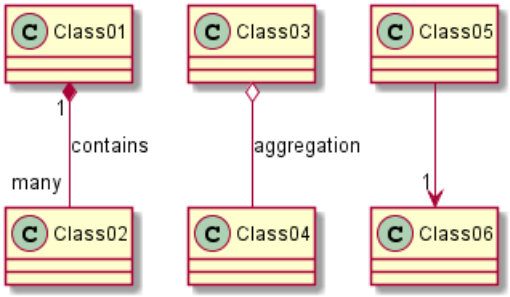


3.2 Libellés sur les relations

Il est possible de rajouter un libellé sur une relation, en utilisant les deux points `:`, suivi du texte du libellé.

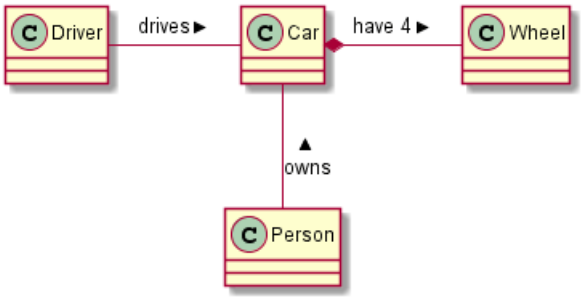
Pour les cardinalité, vous pouvez utiliser des guillemets `""` des deux cotés de la relation.

```
@startuml

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

Class03 o-- Class04 : aggregation

Class05 --> "1" Class06

@enduml
```



Vous pouvez ajouter une flèche désignant quel objet agit sur l'autre en utilisant < ou > au début ou à la fin du libellé.

```
@startuml
class Car

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

@enduml
```

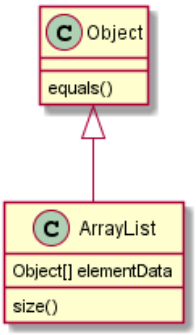


3.3 Définir les méthodes

Pour déclarer des méthodes ou des champs, vous pouvez utiliser le caractère : suivi de la méthode ou du champ.

Le système utilise la présence de parenthèses pour choisir entre méthodes et champs.

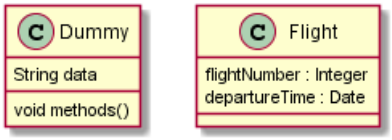
```
@startuml
Object <|-- ArrayList

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

@enduml
```



Il est possible de regrouper tous les champs et méthodes en utilisant des crochets {}.

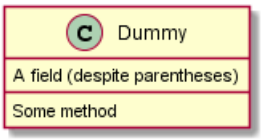
Notez que la syntaxe est très souple sur l'ordre des champs et des méthodes.

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

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



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

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

@enduml
```

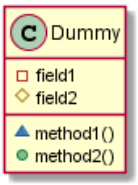


3.4 Définir les visibilités

Quand vous déclarez des champs ou des méthodes, vous pouvez utiliser certains caractères pour définir la visibilité des éléments :

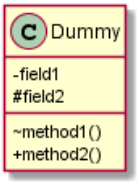
Character	Icon for field	Icon for method	Visibility
-	□	■	private
#	◇	◆	protected
~	△	▲	package private
+	○	●	public

```
@startuml

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

@enduml
```



Vous pouvez invalider cette fonctionnalité par la commande `skinparam classAttributeIconSize 0` :

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

@enduml
```



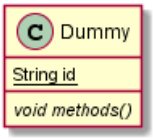
3.5 Abstrait et statique

Vous pouvez définir une méthode statique ou abstraite ou un champ utilisant `{static}` ou `{abstract}` modificateur.

Ce modificateur peut être utilisé au début ou à la fin de la ligne. Vous pouvez alors utiliser `{classifier}` plutôt que `{static}`.

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

```
@enduml
```

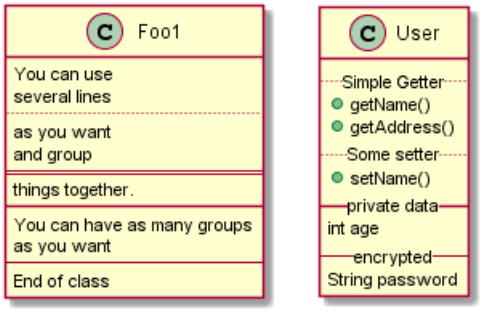


3.6 Corps de classe avancé

Par défaut, méthodes et champs sont automatiquement regroupés par PlantUML. Vous pouvez utiliser un séparateur pour définir votre propre manière d'ordonner les champs et les méthodes. Les séparateurs suivants sont possibles : `--` `..` `==` `__`.

Vous pouvez aussi utiliser les titres dans les séparateurs.

```
@startuml
class Foo1 {
  You can use
  several lines
  ..
  as you want
  and group
  ==
  things together.
  --
  You can have as many groups
  as you want
  --
  End of class
}

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

@enduml
```



3.7 Notes et stéréotypes

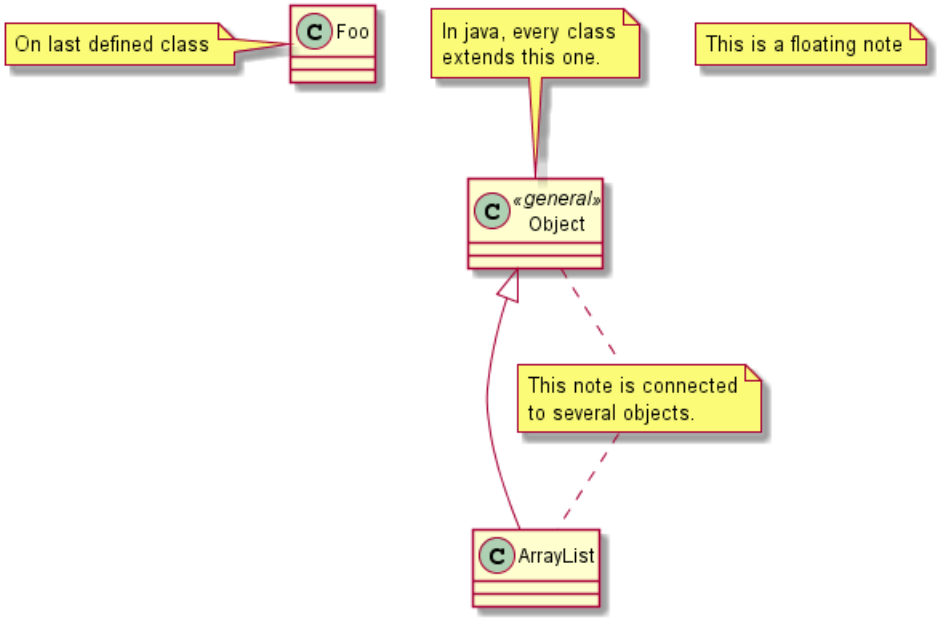
Stéréotypes sont définies avec le mot clé `class`, `<<` et `>>`.

Vous pouvez aussi définir une note en utilisant les mots clés `note left of` , `note right of` , `note top of` , `note bottom of`.

Vous pouvez aussi définir une note sur la dernière classe utilisant `note left`, `note right`, `note top`, `note bottom`.

Une note peut aussi être définie le mot clé `note`, puis être lié à un autre objet en utilisant le symbole `...`

```
@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 Encore des notes

Il est possible d'utiliser quelques tag HTML comme :

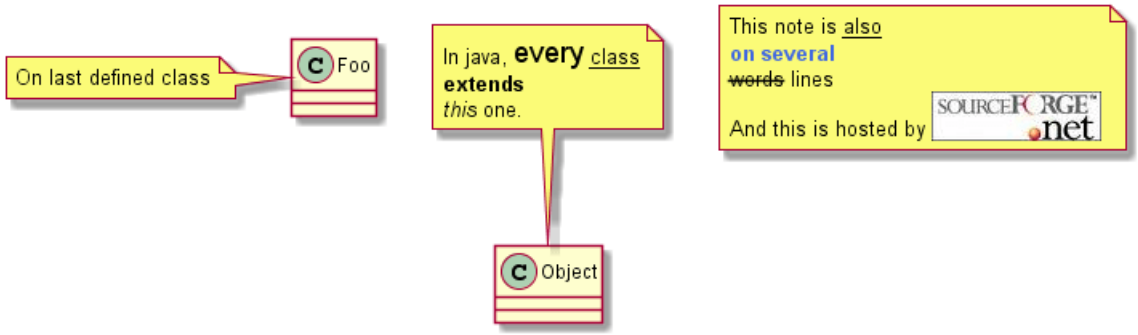
- `<b>`
- `<u>`
- `<i>`
- `<s>`, `<del>`, `<strike>`
- `<font color="#AAAAAA">` or `<font color="colorName">`
- `<color:#AAAAAA>` or `<color:colorName>`

- `<size:nn>` to change font size
- `<img src="file">` or `<img:file>`: the file must be accessible by the filesystem

Vous pouvez aussi définir des notes sur plusieurs lignes.

Vous pouvez également définir une note sur la dernière classe définie en utilisant `note left`, `note right`, `note top`, `note bottom`.

```
@startuml

class Foo
note left: On last defined class

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

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

@enduml
```

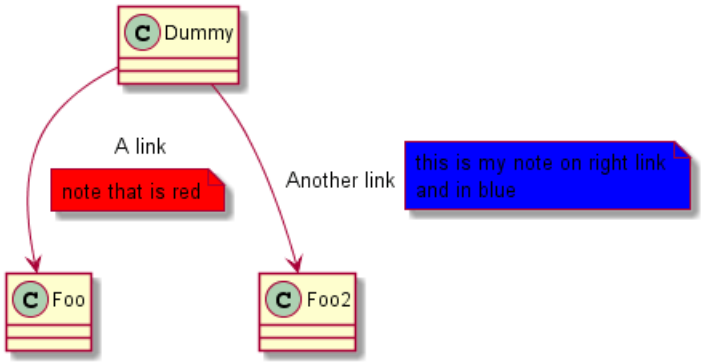


3.9 Note sur les liens

Il est possible d'ajouter une note sur un lien, juste après la définition d'un lien, utiliser `note on link`.

Vous pouvez aussi utiliser `note left on link`, `note right on link`, `note top on link`, `note bottom on link` si vous voulez changer la position relative de la note avec l'étiquette.

```
@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
```

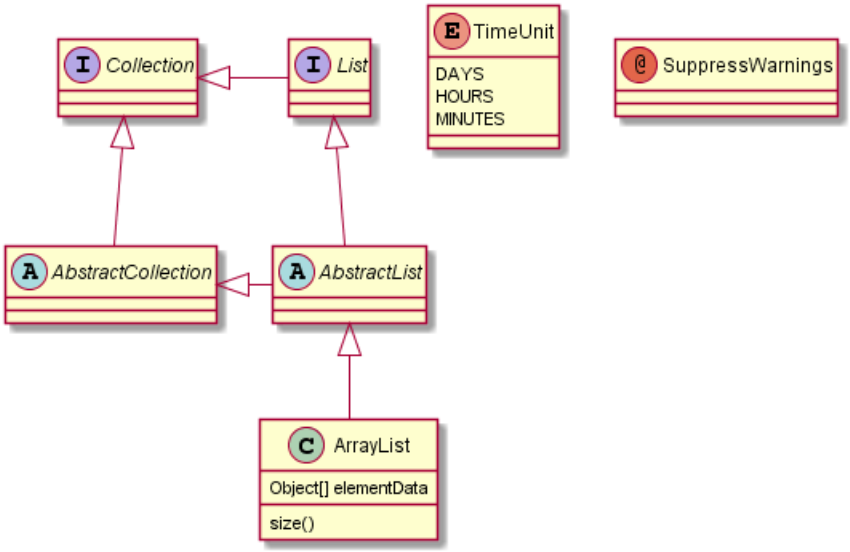


3.10 Classe abstraite et Interface

Vous pouvez déclarer un classe abstraite en utilisant `abstract` ou `abstract class`. La classe sera alors écrite en *italique*.

Vous pouvez aussi utiliser `interface`, `annotation` et `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
```

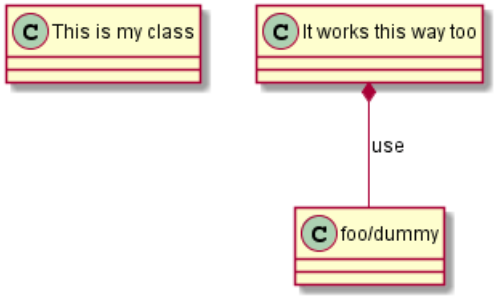


3.11 Caractères non alphabétiques

Si nous voulez utiliser autre chose que des lettres dans les classes (ou les enums...), vous pouvez:

- Utiliser le mot clé `as` dans la définition de la classe
- Metter des guillemets `""` autour du nom de la classe

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

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



3.12 Masquer les attributs et les méthodes

Vous pouvez paramétrer l'affichage des classes à l'aide de la commande `hide/show` .

La commande de base est: `hide empty members`. Cette commande va masquer la zone des champs ou des méthodes si celle-ci est vide.

A la place de `empty members`, vous pouvez utiliser:

- `empty fields` ou `empty attributes` pour des champs vides,
- `empty methods` pour des méthodes vides,
- `fields` or `attributes` qui masque les champs, même s'il y en a de définis,
- `methods` qui masque les méthodes, même s'il y en a de définies,
- `members` qui masque les méthodes ou les champs, même s'il y en a de définies,

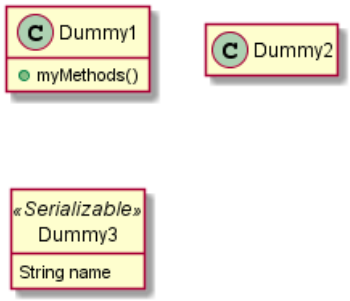
- `circle` pour le caractère entouré en face du nom de la classe,
- `stereotype` pour le stéréotype.

Vous pouvez aussi fournir, juste après le mot-clé `hide` ou `show` :

- `class` pour toutes les classes,
- `interface` pour toutes les interfaces,
- `enum` pour tous les enums,
- `<<foo1>>` pour les classes qui sont stéréotypée avec *foo1*,
- Un nom de classe existant

Vous pouvez utiliser plusieurs commandes `show/hide` pour définir des règles et des exceptions.

```
@startuml

class Dummy1 {
  +myMethods()
}

class Dummy2 {
  +hiddenMethod()
}

class Dummy3 <<Serializable>> {
String name
}

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

@enduml
```



3.13 Cacher des classes

Vous pouvez également utiliser la commande `show/hide` pour cacher une classe.

Cela peut être utile si vous définissez un fichier inclus de grande taille, et si vous voulez en cacher quelques classes après l'inclusion de ce fichier.

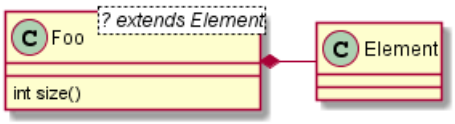
```
@startuml

class Foo1
class Foo2

Foo2 *-- Foo1

hide Foo2
```

```
@enduml
```

3.14 Utilisation de la généricité

Vous pouvez aussi utiliser les signes inférieur < et supérieur > pour définir l'utilisation de la généricité dans une classe.

```
@startuml

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

@enduml
```



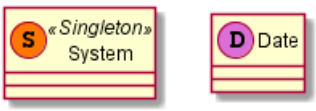
On peut désactiver ce comportement avec la commande `skinparam genericDisplay old`.

3.15 Caractère spécial

Normalement, un caractère (C, I, E ou A) est utilisé pour les classes, les interfaces ou les énum.

Vous pouvez aussi utiliser le caractère de votre choix, en définissant le stéréotype et en ajoutant une couleur, comme par exemple :

```
@startuml

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



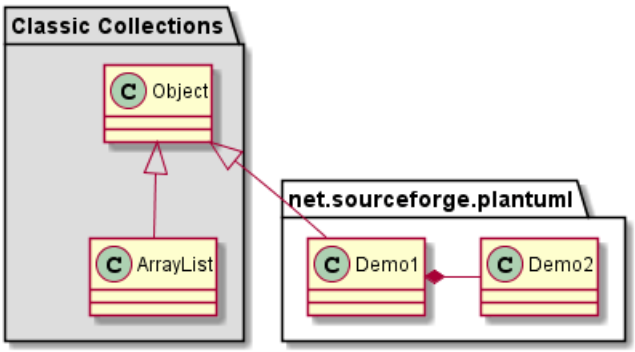
3.16 Packages

Vous pouvez définir un package en utilisant le mot-clé `package`, et optionnellement déclarer une couleur de fond pour votre package (en utilisant un code couleur HTML ou son nom).

Notez que les définitions de packages peuvent être imbriquées.

```
@startuml

package "Classic Collections" #DDDDDD {
```

```
  Object <|-- ArrayList
}

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

@enduml
```



3.17 Modèle de paquet

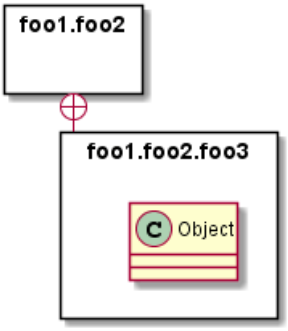
Il y a différents styles de paquets disponibles.

Vous pouvez les spécifier chacun par un réglage par défaut avec la commande : `skinparam packageStyle`, ou par l'utilisation d'un stéréotype sur le paquet:

```
@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
```

Vous pouvez aussi définir les liens entre les paquets, comme dans l'exemple suivant :

```
@startuml

skinparam packageStyle rectangle

package foo1.foo2 {
}

package foo1.foo2.foo3 {
  class Object
}

foo1.foo2 +-- foo1.foo2.foo3

@enduml
```



3.18 Les espaces de nommage

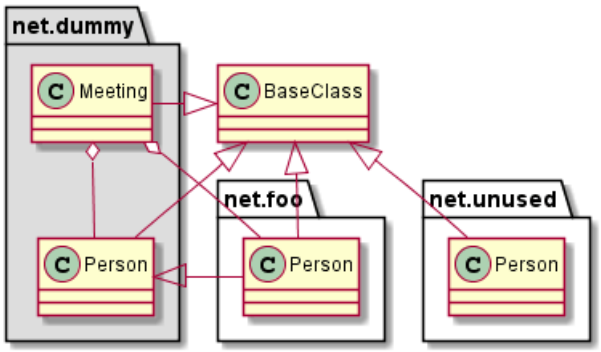
Avec les packages, le nom de la classe est l'identifiant unique de la classe. Cela signifie qu'on ne peux pas avoir deux classes avec le même nom dans deux packages différents. Pour ce faire, vous devez utiliser des espace de nommage (*namespace*) à la place des packages.

Vous pouvez faire référence à des classes d'autres espace de nommage en les nommant complétement. Les classes de l'espace de nommage par défaut (racine) sont nommées en commençant par un point.

Il n'est pas obligatoire de créer les espaces de nom. Un classe avec son nom complet sera automatiquement ajoutée au bon espace de nommage.

```
@startuml

class BaseClass

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

.BaseClass <|- Meeting
}

namespace net.foo {
```

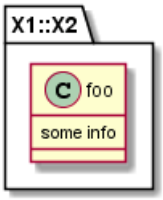
```
  net.dummy.Person  <|- Person
  .BaseClass <|-- Person

  net.dummy.Meeting o-- Person
}

BaseClass <|-- net.unused.Person

@enduml
```



3.19 Creation automatique d'espace de nommage

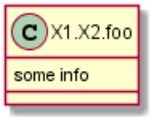
Vous pouvez définir une autre séparateur (autre que le point) en utilisant la commande : `set namespaceSeparator ???`.

```
@startuml

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

@enduml
```



Vous pouvez désactiver la création automatique de package en utilisant la commande `set namespaceSeparator none`.

```
@startuml

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

@enduml
```

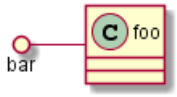


3.20 Interface boucle

Vous pouvez aussi rajouter des interfaces sur les classes avec la syntaxe suivante:

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

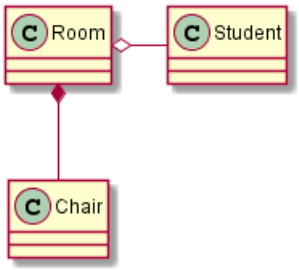
```
@startuml
class foo
bar ()- foo
@enduml
```



3.21 Changer la direction

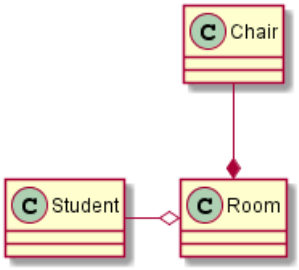
Par défaut, les liens entre les classe ont deux tirets `--` et sont orientés verticalement. Il est possible d'utiliser une ligne horizontal en mettant un simple tiret (Ou un point) comme ceci:

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



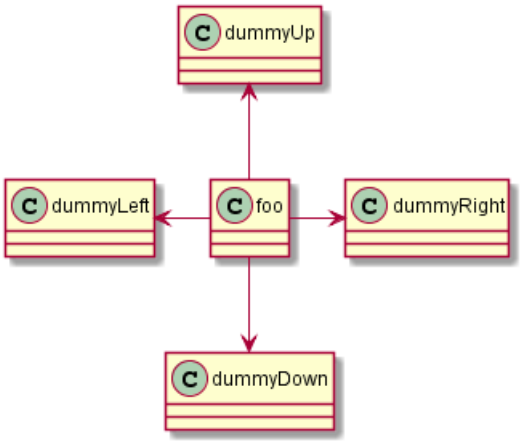
Vous pouvez aussi changer le sens en renversant le lien :

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



Il est aussi possible de changer la direction d'une flèche en ajoutant les mots clés `left`, `right`, `up` ou `down` à l'intérieur de la flèche:

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



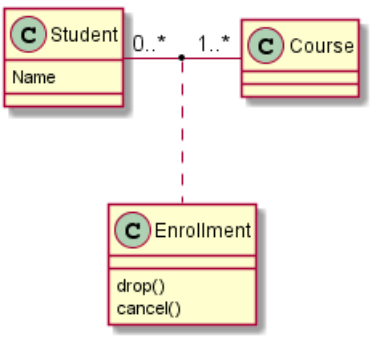
Il est possible de raccourcir la flèche en n'utilisant que la première lettre de la direction (par exemple, `-d-` au lieu de `-down-`) ou les deux premières lettres (`-do-`)

Attention à ne pas abuser de cette fonctionnalité : *GraphViz* donne généralement de bons résultats sans trop de raffistolages.

3.22 Classes d'association

Vous pouvez définir une *classe d'association* après qu'une relation ait été définie entre deux classes, comme dans l'exemple suivant:

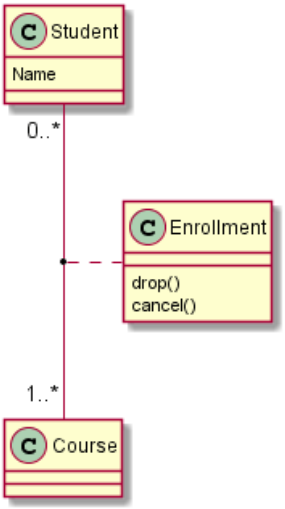
```
@startuml
class Student {
  Name
}
Student "0..*" - "1..*" Course
(Student, Course) .. Enrollment

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



Vous pouvez la définir dans une autre direction :

```
@startuml
class Student {
```

```
  Name
}
Student "0..*" -- "1..*" Course
(Student, Course) . Enrollment

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



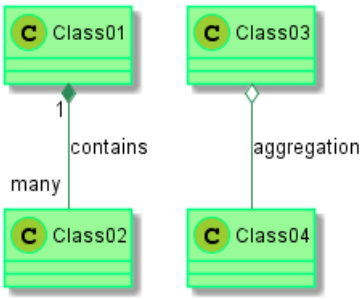
3.23 Personnalisation

La commande `skinparam` permet de changer la couleur et les polices de caractères.

Vous pouvez utiliser cette commande :

- Dans le diagramme, comme toutes les autre commandes,
- Dans un fichier inclus,
- Dans un fichier de configuration précisé par la ligne de commande ou la tâche ANT.

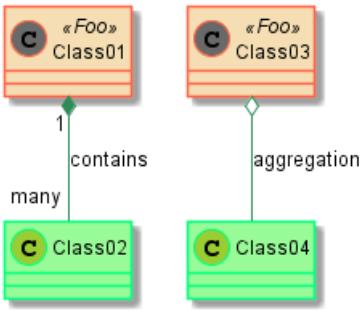
```
@startuml

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

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

Class03 o-- Class04 : aggregation

@enduml
```



3.24 Stéréotypes Personnalisés

Vous pouvez définir des couleurs et des fontes de caractères spécifiques pour les classes stéréotypées.

```
@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 Dégradé de couleur

Il est possible de déclarer individuellement une couleur pour des classes ou une note en utilisant la notation #.

Vous pouvez utiliser un nom de couleur standard ou un code RGB.

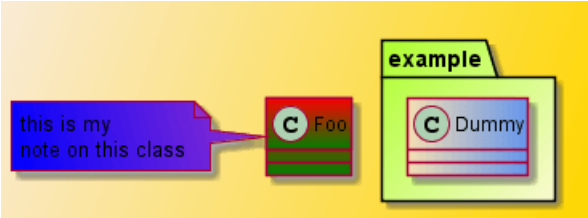
Vous pouvez aussi utiliser un dégradé de couleur en fond, avec la syntaxe suivante : deux noms de couleurs séparés par :

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

en fonction de la direction du dégradé

Par exemple, vous pouvez avoir :

```
@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
```



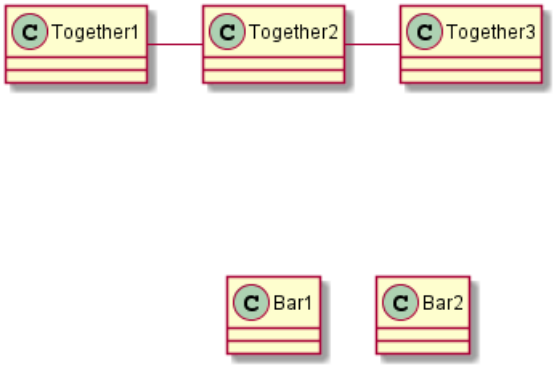
3.26 Aide pour la mise en page

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 Découper les grands diagrammes

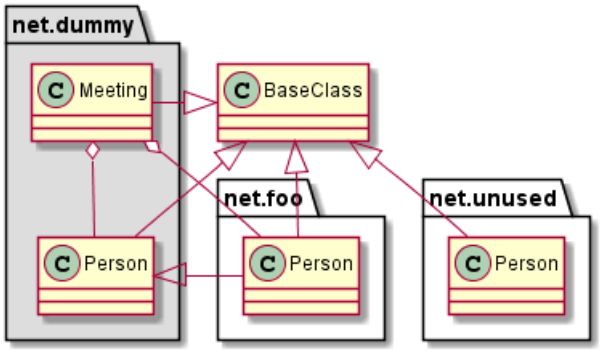
Parfois, vous obtiendrez des images de taille importante.

Vous pouvez utiliser la commande `page (hpages)x(vpages)` pour découper l'image en plusieurs fichiers:

`hpages` est le nombre de pages horizontales et `vpages` indique le nombre de pages verticales.

Vous pouvez aussi utiliser des paramètres spécifiques pour rajouter des bords sur les pages découpées (voir l'exemple).

```
@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
```


net.dummy
Meeting
BaseClass
net.foo
net.unused
Person
Person
Person

4 Diagrammes d'activité

4.1 Exemple de base

Vous devez utiliser `(*)` pour le début et la fin du diagramme d'activité.

Dans certaines occasions, vous pouriez vouloir utiliser `(*top)` pour forcer le début à être en haut du diagramme.

Utiliser `-->` pour les flèches.

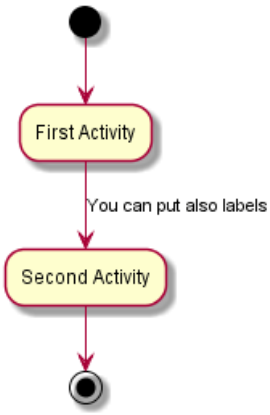
```
@startuml

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

@enduml
```

4.2 Texte sur les flèches.

Par défaut, une flèche commence à partir de la dernière activité définie.

Vous pouvez rajouter un libellé sur une flèche en mettant des crochets `[` et `]` juste après la définition de la flèche.

```
@startuml

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

@enduml
```



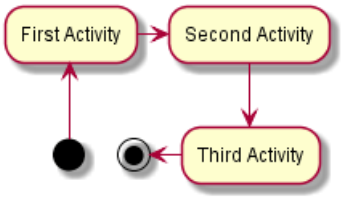
4.3 Changer la direction des flèches

Vous pouvez utiliser `->` pour les flèches horizontales. Il est aussi possible de forcer la direction d'une flèche en utilisant la syntaxe suivante :

- `-down->` (default arrow)

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

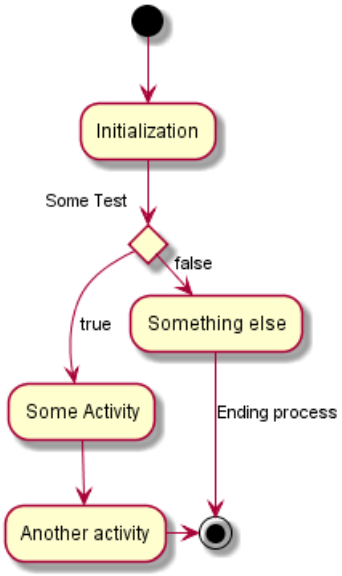
```
@startuml

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

@enduml
```



4.4 Branches

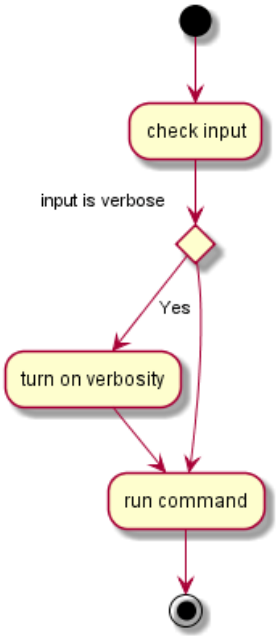
Vous pouvez utiliser le mot clé `if/then/else` pour définir une branche.

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

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

@enduml
```



Malheureusement, vous devez parfois avoir à répéter la même activité dans le diagramme de texte.

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



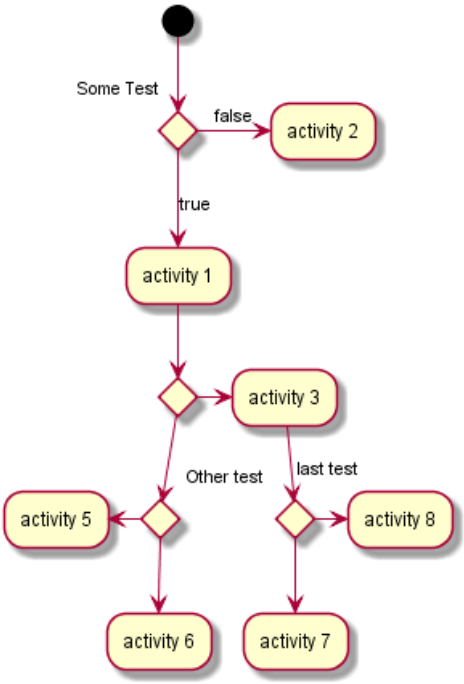
4.5 Encore des branches

Par défaut, une branche commence à la dernière activité définie, mais il est possible de passer outre et de définir un lien avec le mot clé `if`.

Il est aussi possible d'imbriquer les branches.

```
@startuml

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

  -->[true] "activity 1"

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

else

  ->[false] "activity 2"
```

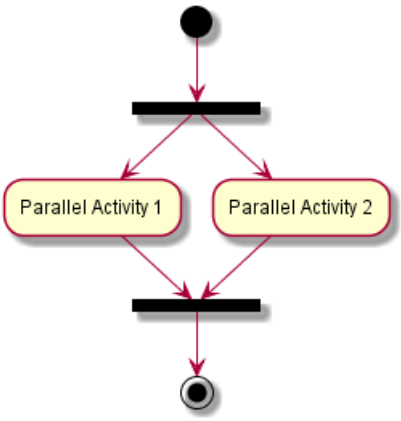
```
endif

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

@enduml
```



4.6 Synchronisation

Vous pouvez utiliser la syntaxe `===` `code` `===` pour afficher des barres de synchronisation.

```
@startuml

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

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

--> (*)

@enduml
```

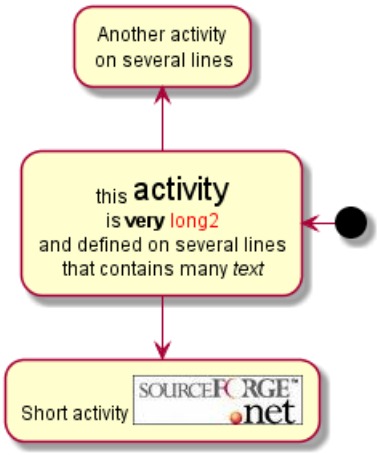


4.7 Description détaillée

Lorsque vous déclarez des activités, vous pouvez positionner sur plusieurs lignes le texte de description Vous pouvez également ajouter `\n` dans la description. Il est également possible d'utiliser quelques tags HTML tels que :

Vous pouvez aussi donner un court code à l'activité avec le mot clé `as`. Ce code peut être utilisé plus tard dans le diagramme de description.

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

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

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



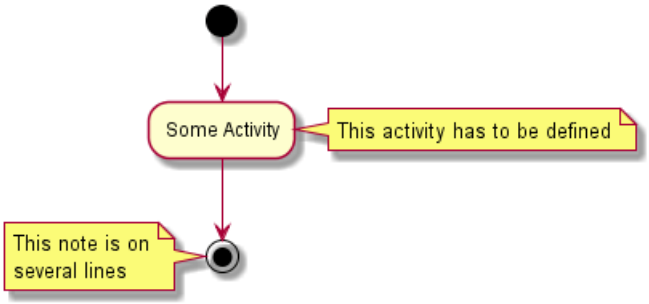
4.8 Notes

Vous pouvez rajouter des notes sur une activités en utilisant les commandes: `note left`, `note right`, `note top` ou `note bottom`, juste après la définition de l'activité concernée.

Si vous voulez mettre une note sur le démarrage du diagramme, définissez la note au tout début du diagramme.

Vous pouvez aussi avoir une note sur plusieurs lignes, en utilisant les mots clés `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
```



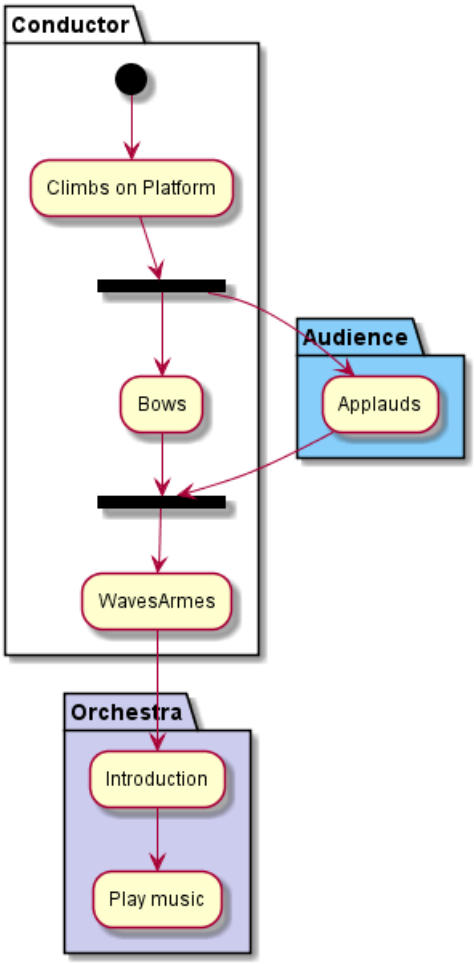
4.9 Partition

Vous pouvez définir une partition en utilisant le mot clé `partition`, et optionnellement déclarer un fond de couleur pour votre partition (En utilisant un code couleur html ou un nom)

Quand vous déclarez les activités, ils sont automatiquement mis dans la dernière partition utilisée.

Vous pouvez fermer la partition de définition en utilisant les crochets fermants `}`.

```
@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 Paramètre de thème

Vous pouvez utiliser la commande `skinparam` pour changer la couleur et la police d'écriture pour dessiner.

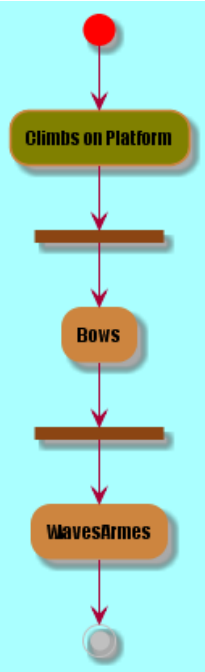
Vous pouvez utiliser cette commande :

- Dans le diagramme de définition, comme n'importe quelle autre commande,
- Dans un fichier inclus,
- Dans un fichier de configuration, à l'aide de la la ligne de commande ou la tâche ANT.

Vous pouvez spécifier une couleur et une police d'écriture dans les stéréotypes d'activités.

```
@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 Octogone

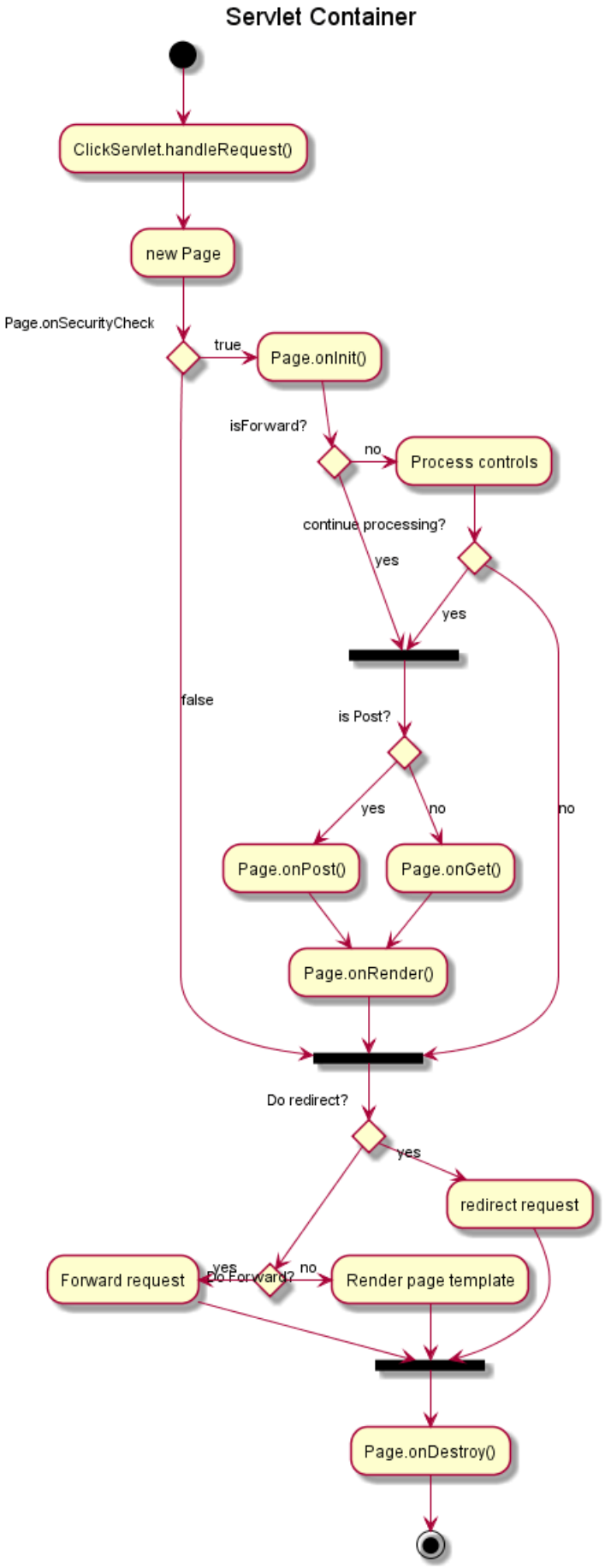
Vous pouvez changer la forme des activités en octogone en utilisant la commande `skinparam activityShape octagon`.

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

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

@enduml
```

4.12 Exemple complet

```
@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 Diagrammes d'activité (béta)

La syntaxe courante pour les diagrammes d'activité possède plusieurs limitations et inconvénients (par exemple, la difficulté à maintenir un diagramme lors de modifications).

Une complète nouvelle syntaxe et implémentation est proposée avec **beta version** aux utilisateurs (commence avec V7947), ainsi cela permet de définir une nouvelle et meilleure syntaxe.

Un autre avantage de cette nouvelle implémentation est qu'il n'y a pas besoin d'avoir Graphviz d'installé (comme pour les diagrammes de séquences).

La nouvelle syntaxe remplace l'ancienne. Cependant, pour des raisons de compatibilité, l'ancienne syntaxe reste reconnu, pour assurer *la compatibilité ascendante*.

Les utilisateurs sont simplement encouragés à migrer vers la nouvelle syntaxe.

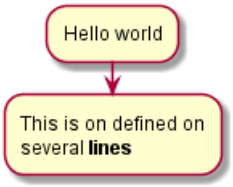
5.1 Activité simple

Les étiquettes d'activités commencent avec `:` et finissent avec `;`.

Le formatage de texte peut être fait en utilisant la syntaxe créole wiki.

Ils sont implicitement liés à leur ordre de définition.

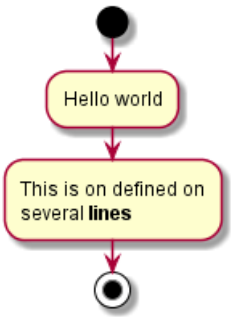
```
@startuml
:Hello world;
:This is on defined on
several **lines**;
@enduml
```



5.2 Départ/Arrêt

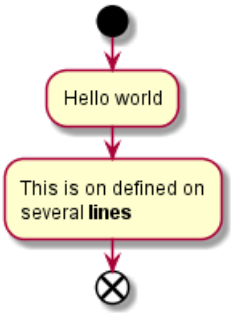
Vous pouvez utiliser les mots clés `start` et `stop` pour indiquer le début et la fin du diagramme.

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



Vous pouvez aussi utiliser le mot clés `end`.

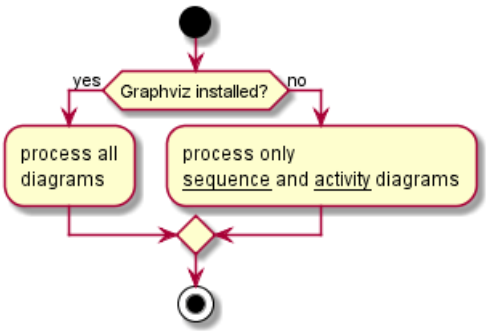
```
@startuml
start
:Hello world;
:This is on defined on
several **lines**;
end
@enduml
```



5.3 Conditionnel

Vous pouvez utiliser les mots clés `if`, `then` et `else` pour mettre des tests si votre diagramme. Les étiquettes peuvent être fournies entre parenthèse.

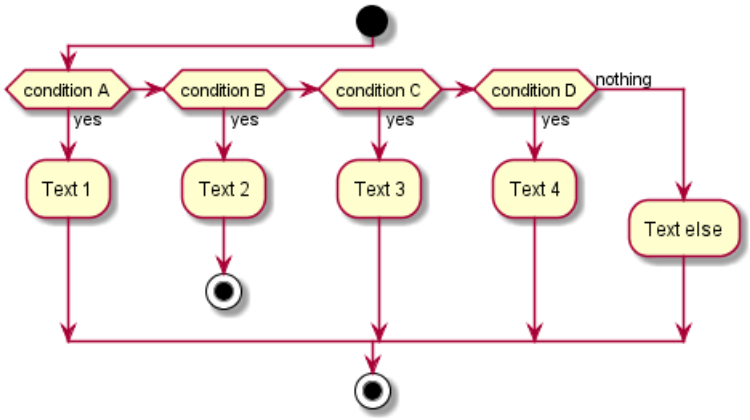
```
@startuml

start

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

stop

@enduml
```



Vous pouvez utiliser le mot clé `elseif` pour avoir plusieurs tests :

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

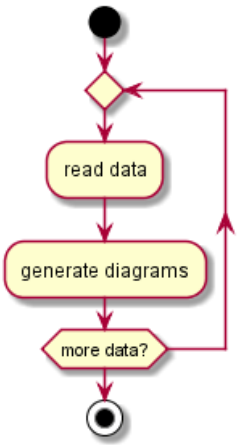
```
elseif (condition C) then (yes)
  :Text 3;
elseif (condition D) then (yes)
  :Text 4;
else (nothing)
  :Text else;
endif
stop
@enduml
```



5.4 Boucle de répétition

Vous pouvez utiliser les mots clés `repeat` et `repeatwhile` pour créer une boucle.

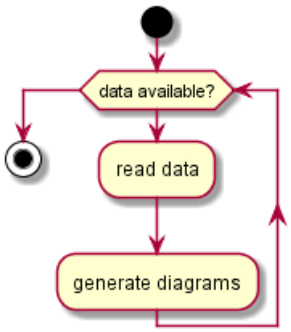
```
@startuml

start

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

stop

@enduml
```



5.5 Boucle While

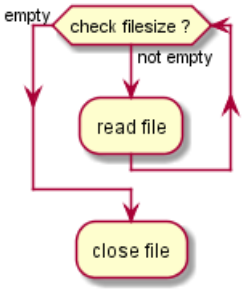
Vous pouvez utiliser les mots clés `while` et `end while` pour définir une boucle.

```
@startuml

start

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

stop

@enduml
```



Il est possible de mettre un libellé après le mot clé `endwhile` ou bien avec le mot clé `is`.

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

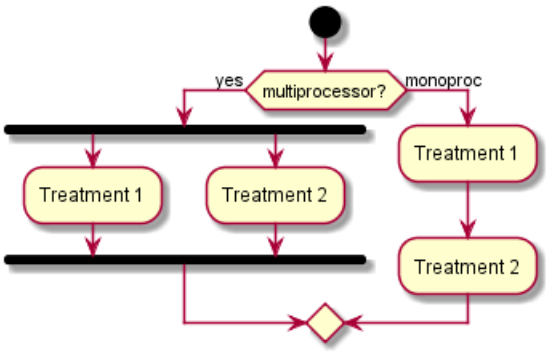


5.6 Processus parallèle

Vous pouvez utiliser les mots clés `fork`, `fork again` et `end fork` pour indiquer un processus parallèle.

```
@startuml

start

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

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

@enduml
```

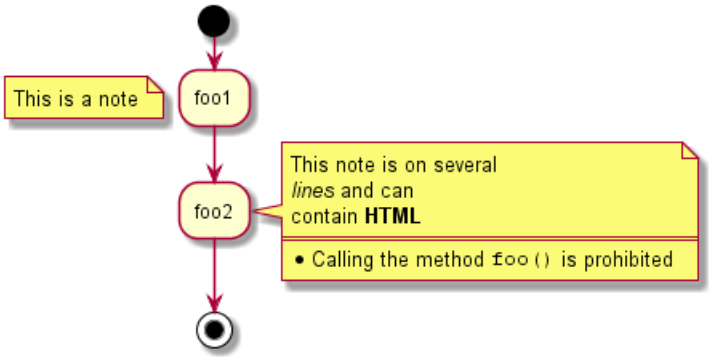


5.7 Notes

Le formattage de texte peut être fait en utilisant la syntaxe créole wiki.

Une note peut aussi être détachée, à l'aide du mot-clé `floating`.

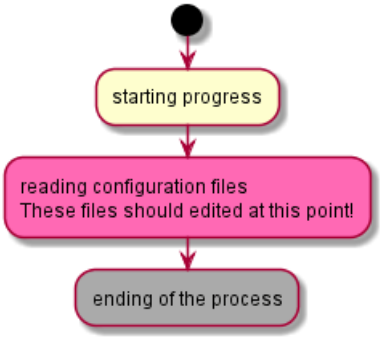
```
@startuml

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 Couleurs

Vous pouvez spécifier une couleur pour certaines activités.

```
@startuml

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

@enduml
```

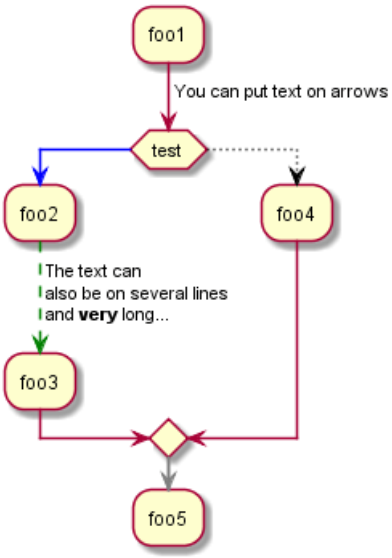


5.9 Flèches

Utiliser la notation `->`, vous pouvez ajouter le texte à la flèche, et changer leur couleur.

Il est aussi possible d'avoir des flèches en pointillé, en gras, avec des tirets ou bien complètement cachées.

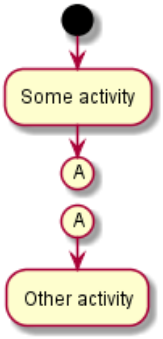
```
@startuml
:foo1;
-> You can put text on arrows;
if (test) then
  -[#blue]->
  :foo2;
  -[#green,dashed]-> The text can
  also be on several lines
  and **very** long...;
  :foo3;
else
  -[#black,dotted]->
  :foo4;
endif
-[#gray,bold]->
:foo5;
@enduml
```



5.10 Connector

You can use parentheses to denote connector.

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

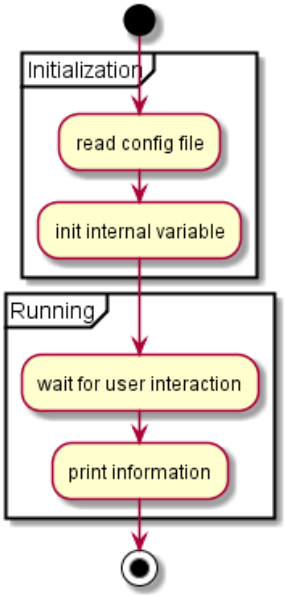


5.11 Groupement

Vous pouvez grouper les activités ensembles en définissant les partitions.

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

```
stop
@enduml
```

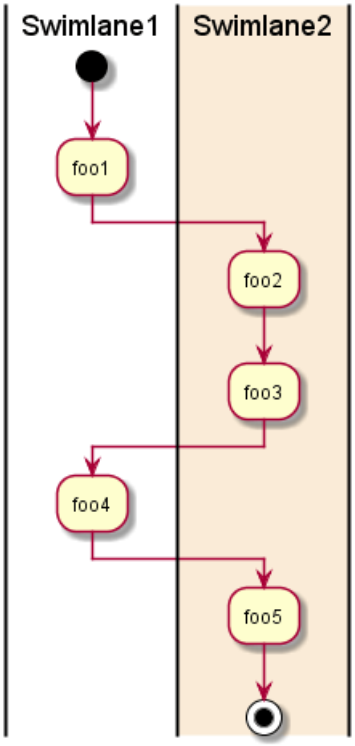


5.12 Couloirs

A l'aide du symbole `|`, il est possible de définir des couloirs d'éxecution.

Il est aussi possible de changer la couleur d'un couloir.

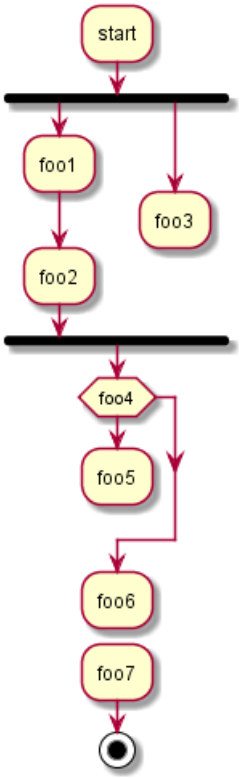
```
@startuml
|Swimlane1|
start
:foo1;
|#AntiqueWhite|Swimlane2|
:foo2;
:foo3;
|Swimlane1|
:foo4;
|Swimlane2|
:foo5;
stop
@enduml
```



5.13 Détacher

Il est possible de supprimer un utilisant le mot clé `detach`.

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



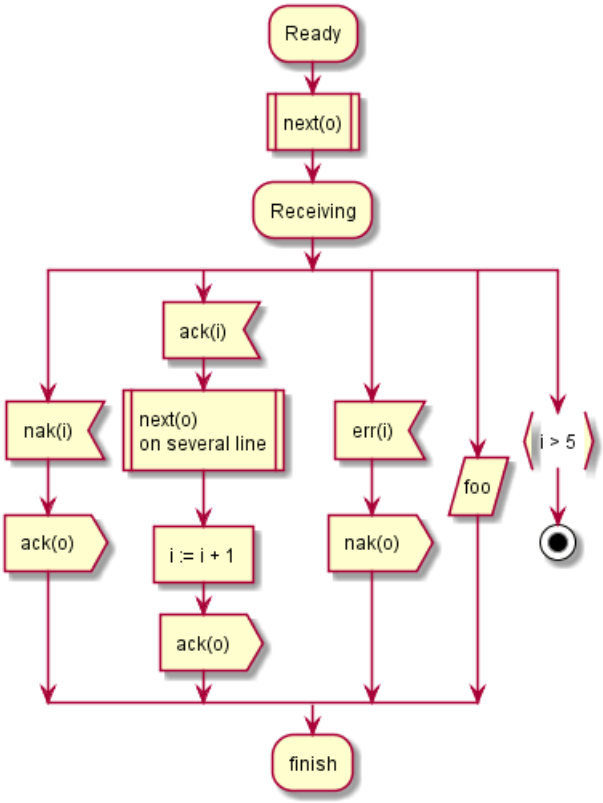
5.14 SDL

En changeant le séparateur final ;, vous pouvez déterminer différents rendu pour l' activité

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

```
@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}
```

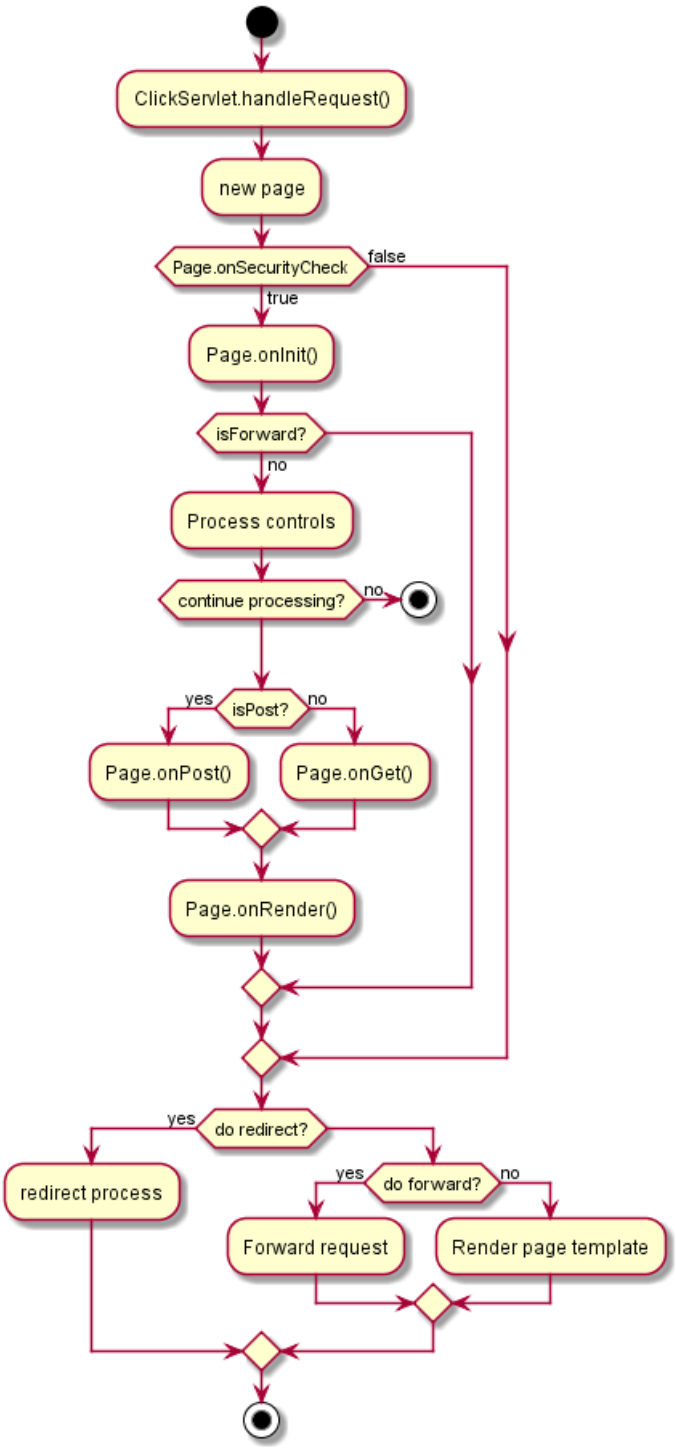
```
stop
end split
:finish;
@enduml
```



5.15 Exemple complet

```
@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 Diagrammes de composants

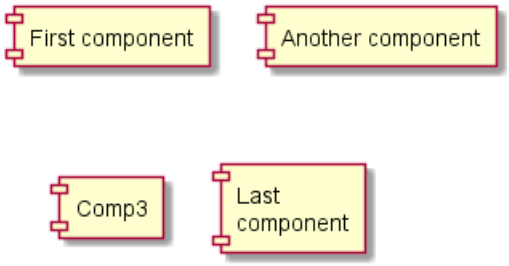
Prenons quelques exemples :

6.1 Composants

Les composants doivent être mis entre crochets.

Il est aussi possible d'utiliser le mot-clé `component` pour définir un composant. Et vous pouvez définir un alias, grâce au mot-clé `as`. Cet alias sera utile plus tard, pour définir des relations entre composants.

```
@startuml

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

@enduml
```



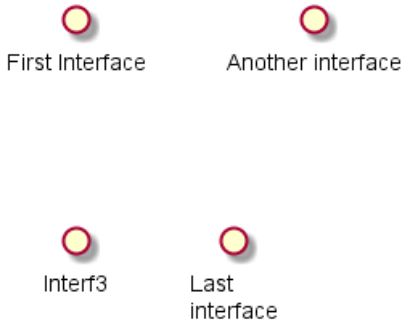
6.2 Interfaces

Les interfaces sont définies à l'aide du symbole `()` (parce que cela ressemble à un cercle).

Vous pouvez aussi utiliser le mot-clé `interface` pour définir une interface. Vous pouvez aussi définir un alias, à l'aide du mot-clé `as`. Cet alias pourrait être utilisé plus tard, lors de la définition des relations.

Nous verrons plus tard qu'il n'est pas obligatoire de définir les interfaces.

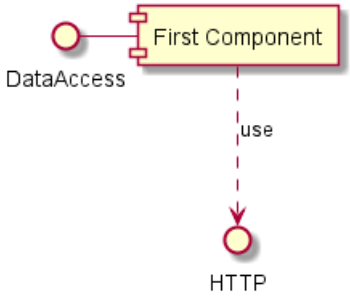
```
@startuml

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

@enduml
```



6.3 Exemple simple

Les liens entre les éléments sont à utiliser avec des combinaisons de lignes pointillés (. .), lignes droites(--), et de flèches (-->).

```
@startuml

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

@enduml
```

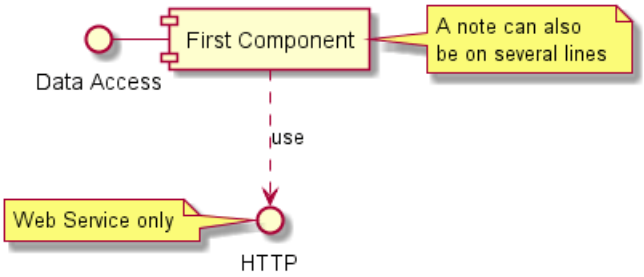


6.4 Mettre des notes

Vous pouvez utiliser les commandes suivantes : `note left of` , `note right of` , `note top of` , `note bottom of` suivi de l'identifiant du composant pour définir des notes reliées à un seul objet.

Une note peut aussi être créée seule avec le mot clé `note`, puis ensuite reliée à d'autres objets avec le symbole . . .

```
@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
```

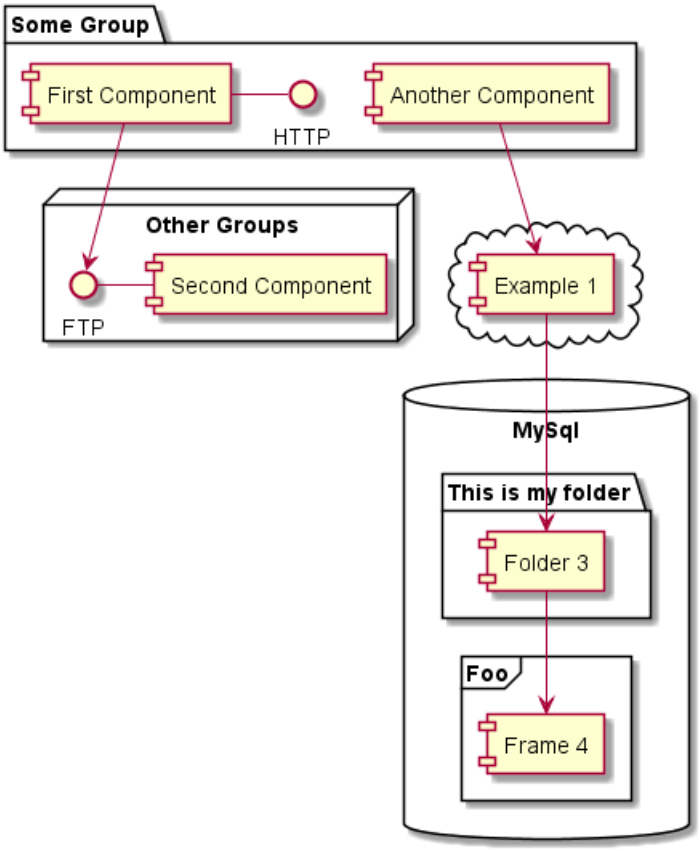


6.5 Regrouper des composants

Vous pouvez utiliser le mot-clé `package` pour regrouper des composants et des interfaces ensembles.

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

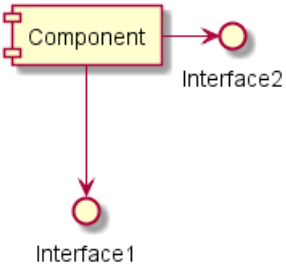
```
@startuml

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 Changer la direction des flèches

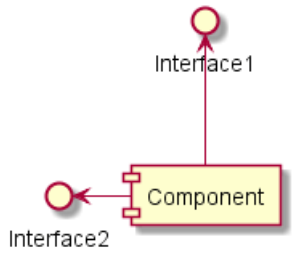
Par défaut, les liens entre classes ont deux tirets `--` et sont orientées verticalement. Il est possible d'utiliser un lien horizontal en mettant un simple tiret (ou point) comme ceci :

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



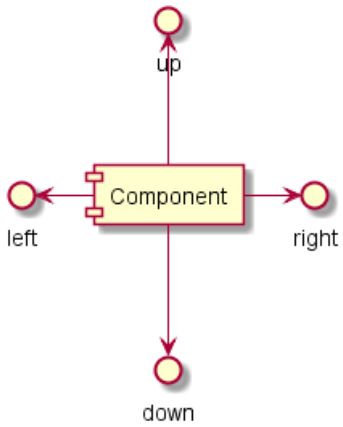
Vous pouvez aussi changer le sens en renversant le lien

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



Il est aussi possible de changer la direction des flèches en ajoutant les mots clés `left`, `right`, `up` ou `down` à l'intérieur des flèches :

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



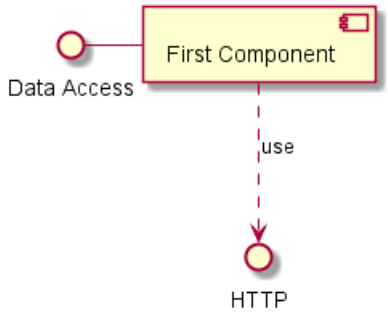
Vous pouvez raccourcir les flèches en utilisant seulement les premiers caractères de la direction (par exemple, `-d-` à la place de `-down-`) ou les deux premiers caractères (`-do-`).

Veuillez noter qu'il ne faut pas abuser de cette fonctionnalité : *Graphviz* donne généralement de bon résultat sans modification.

6.7 Utiliser la notation UML2

La commande `skinparam componentStyle uml2` est utilisée pour changer vers la notation UML2.

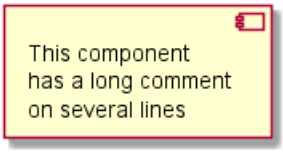
```
@startuml
skinparam componentStyle uml2

interface "Data Access" as DA

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

@enduml
```



6.8 Description longue

Il est possible de mettre un long texte sur plusieurs lignes en utilisant des crochets.

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



6.9 Couleurs individuelles

Il est possible de spécifier une couleur après la définition du composant.

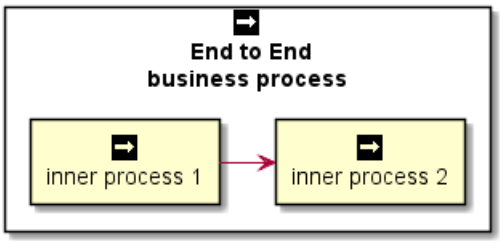
```
@startuml
component  [Web Server] #Yellow
@enduml
```

6.10 Sprites et stéréotypes

Vous pouvez utiliser des sprites dans les stéréotypes des composants.

```
@startuml
sprite $businessProcess [16x16/16] {
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFF0FFFFF
FFFFFFFFFF00FFFF
FF00000000000FFF
FF000000000000FF
FF00000000000FFF
FFFFFFFFFF00FFFF
FFFFFFFFFF0FFFFF
```

```
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
}


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



6.11 Skinparam

Utilisez la commande `skinparam` pour changer la couleur et la mise en forme du texte du schéma.

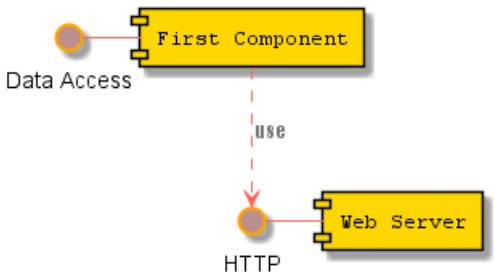
Vous pouvez utiliser cette commande :

- Dans la définition du diagramme, comme pour les autres commandes,
- Dans un fichier inclus,
- Dans un fichier de configuration, renseigné dans la ligne de commande ou la tâche ANT.

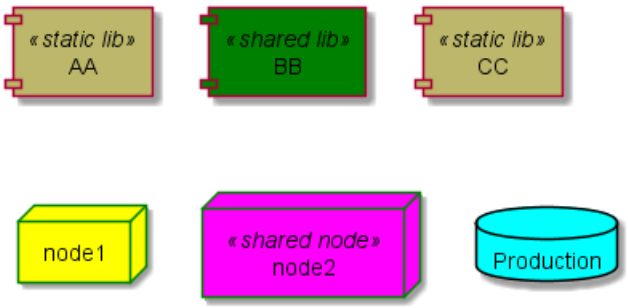
Vous pouvez définir des couleurs et des fontes spécifiques pour les composants et interfaces stéréotypés.

```
@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
```



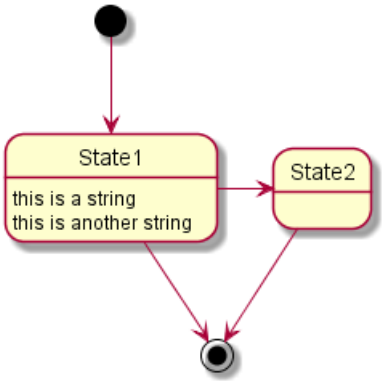
7 Diagrammes d'état

7.1 Exemple simple

Vous devez utiliser `[*]` pour le début et la fin du diagramme d'état.

Utilisez `-->` pour les flèches.

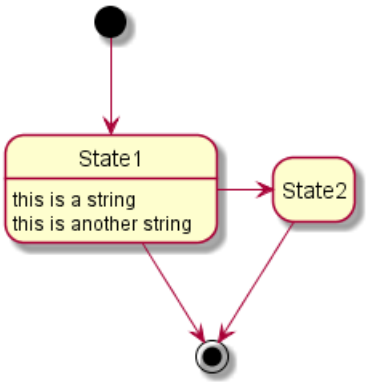
```
@startuml

[*] --> 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.

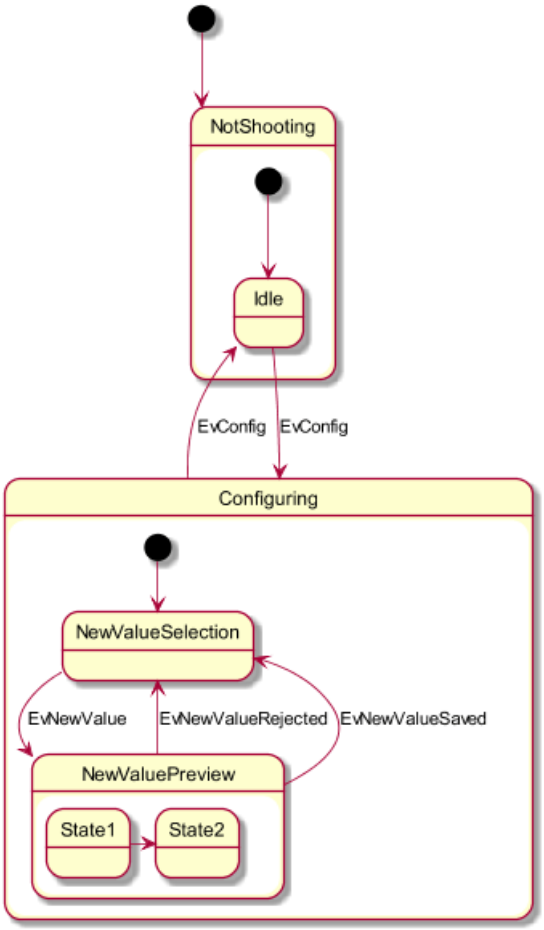
```
@startuml
hide empty description
[*] --> State1
State1 --> [*]
State1 : this is a string
State1 : this is another string

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



7.3 Etat composite

Un état peut également être composite. Vous devez alors le définir avec le mot-clé `state` et des accolades.

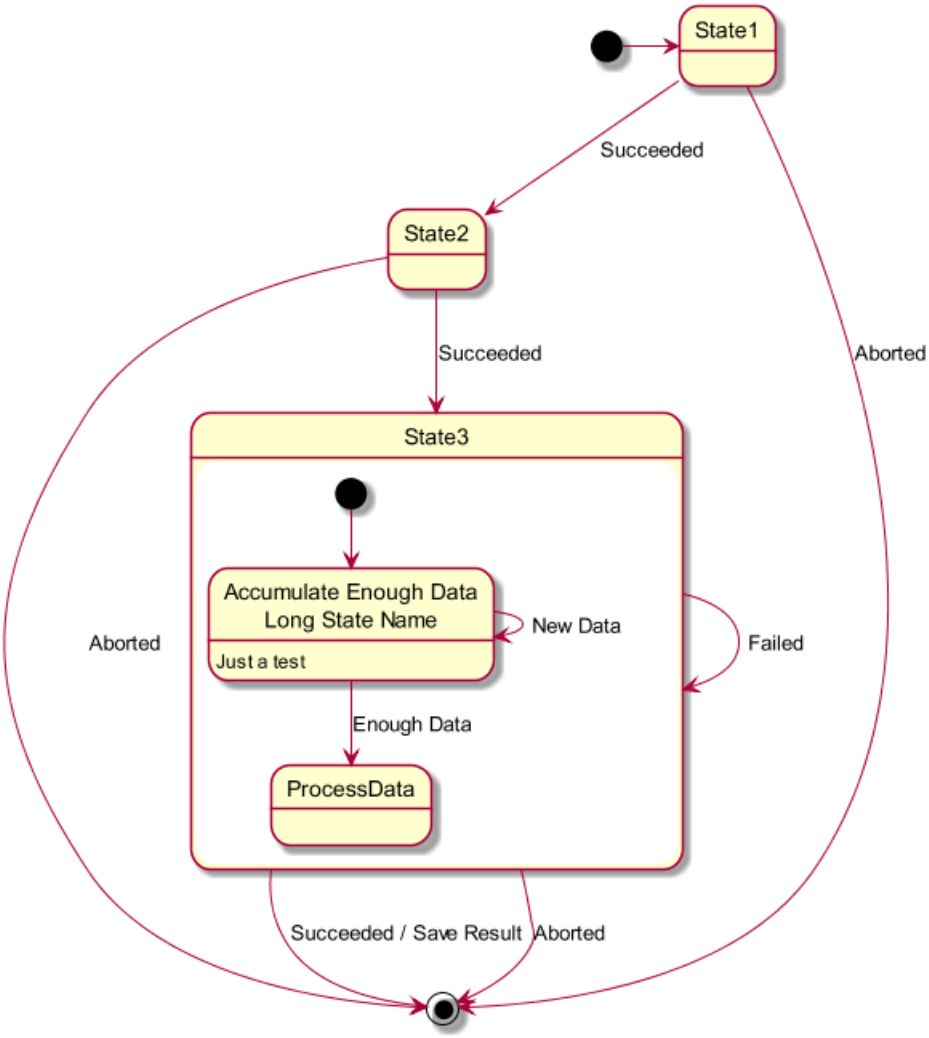
```
@startuml
scale 350 width
[*] --> NotShooting

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 Nom long

Vous pouvez aussi utiliser le mot-clé `state` pour donner un nom avec des espaces à un état.

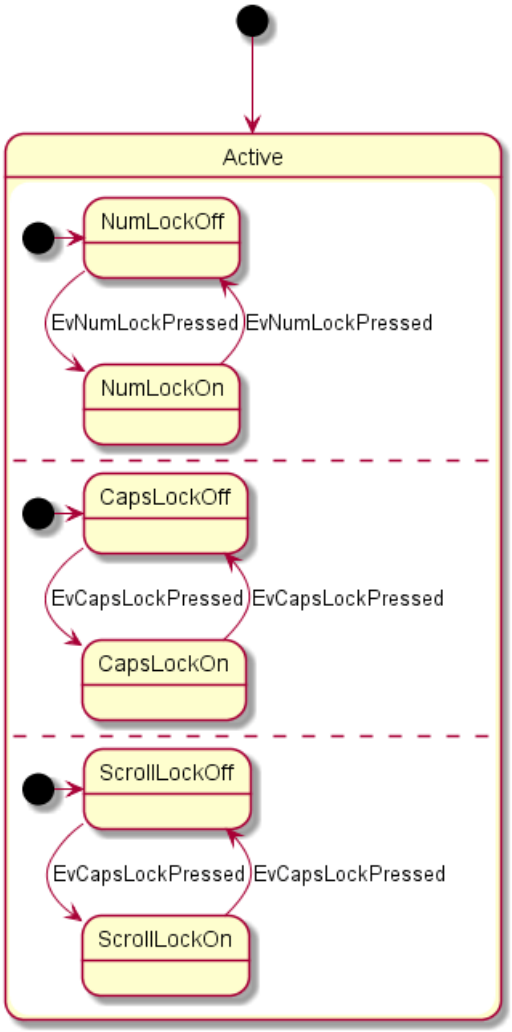
```
@startuml
scale 600 width

[*] -> 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 Etat concurrent

Vous pouvez définir un état concurrent dans un état composé en utilisant le symbole `--` ou `||` comme séparateur.

```
@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
```

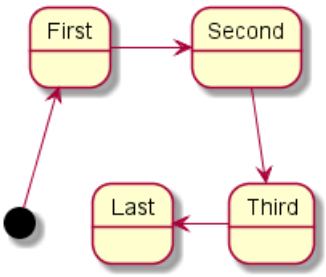


7.6 Direction des flèches

Vous pouvez utiliser `->` pour les flèches horizontales. Il est aussi possible de forcer la direction de la flèche avec la syntaxe suivante:

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

```
@startuml

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

@enduml
```



Vous pouvez aussi utiliser une notation abrégée, avec soit le premier caractère de la direction (par exemple `-d-` à la place de `-down-`) ou bien les deux premiers caractères (`-do-`).

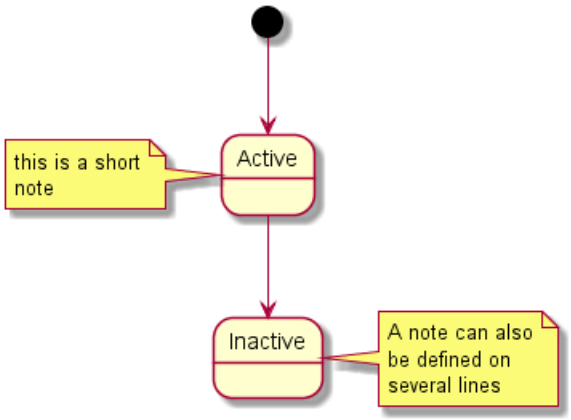
Veuillez noter qu'il ne faut pas abuser de cette fonction : *Graphviz* donne généralement de bons résultats sans peaufinage.

7.7 Note

Vous pouvez définir des notes avec les mots clés suivant: `note left of`, `note right of`, `note top of`, `note bottom of`

Vous pouvez aussi définir des notes sur plusieurs lignes.

```
@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
```



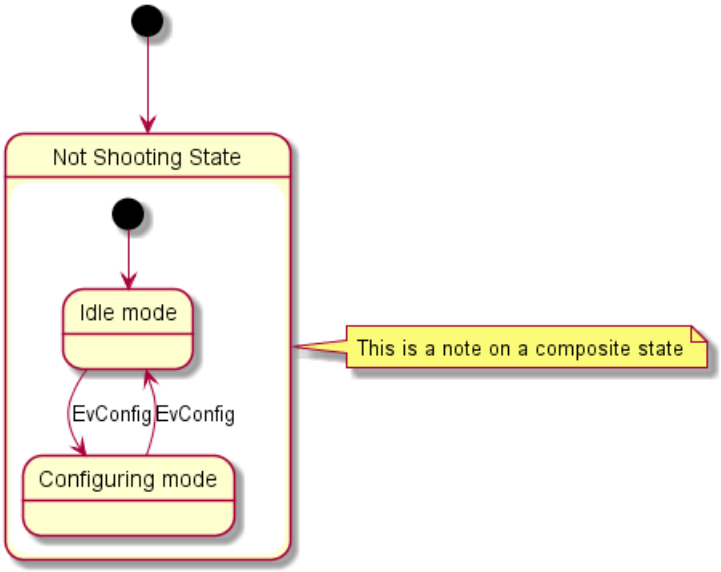
Vous pouvez aussi avoir des notes flottantes.

```
@startuml

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

@enduml
```

7.8 Plus de notes

Vous pouvez mettre des notes sur les états de composite

```
@startuml

[*] --> NotShooting

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

note right of NotShooting : This is a note on a composite state

@enduml
```



7.9 Skinparam

Utilisez la commande `skinparam` pour changer la couleur et la mise en forme du texte du schéma.

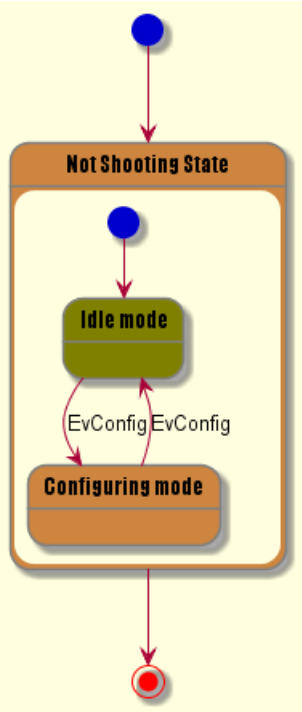
Vous pouvez utiliser cette commande :

- Dans la définition du diagramme, comme pour les autres commandes,
- Dans un fichier inclus,
- Dans un fichier de configuration, renseigné dans la ligne de commande ou la tâche ANT.

Vous pouvez définir une couleur spécifique et une police d'écriture pour les états stéréotypés.

```
@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 Diagrammes d'objets

8.1 Définition des objets

Les instances d'objets sont défnies avec le mot clé `object`.

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

8.2 Relations entre les objets

Les relations entre objets sont définies à l'aide des symboles suivants :

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

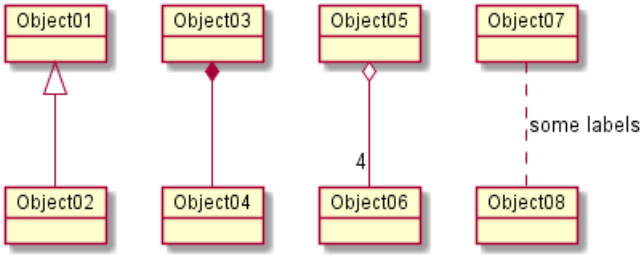
Il est possible de remplacer `--` par `..` pour avoir des pointillés.

Grâce à ces règles, on peut avoir les dessins suivants:

Il est possible d'ajouter une étiquette sur la relation, en utilisant `:` suivi par le texte de l'étiquette.

Pour les cardinalités, vous pouvez utiliser les doubles quotes `""` sur chaque côté de la relation.

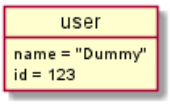
```
@startuml
object Object01
object Object02
object Object03
object Object04
object Object05
object Object06
object Object07
object Object08

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



8.3 Ajout de champs

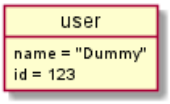
Pour déclarer un champ, vous pouvez utiliser le symbole `:` suivi par le nom du champs.

```
@startuml

object user

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

@enduml
```



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

```
@startuml

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

@enduml
```



8.4 Caractéristiques communes avec les diagrammes de classes

- Visibilité
- Ajout de notes
- Utilisation de packages
- Personnalisation de l'affichage

9 Diagramme de temps

Ceci n'est qu'une proposition qui est susceptible d'évoluer.

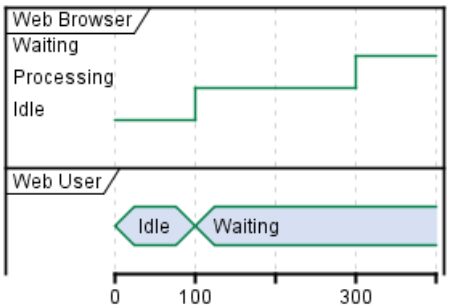
Vous êtes invités à créer des discussions sur cette future syntaxe. Vos retours, vos idées et vos suggestions nous aideront à trouver la meilleure solution.

9.1 Définitions des participants

Les participants sont déclarés à l'aide des mots-clé `consise` ou `robust`, en fonction de la façon dont vous souhaitez les dessiner.

Les changements d'état sont notifiés avec la notation `@` et le verbe `is`.

```
@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
```

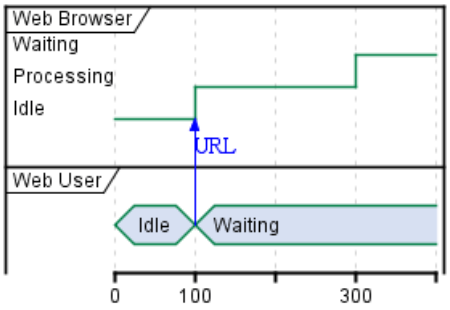


9.2 Ajout de messages

Vous pouvez rajouter des messages à l'aide de la syntaxe suivante.

```
@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
```

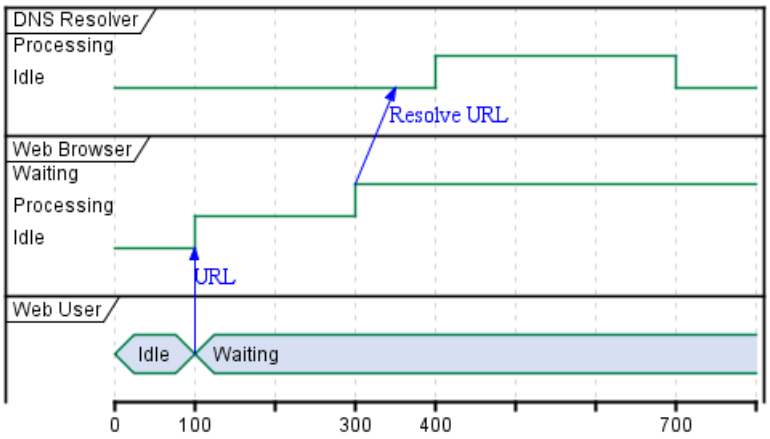
```
WB is Waiting

@enduml
```



9.3 Référence relative de temps

Avec la notation `@`, il est possible d'utiliser une notation relative du temps.

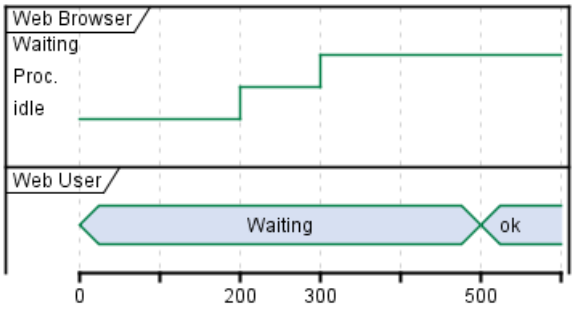
```
@startuml
robust "DNS Resolver" as DNS
robust "Web Browser" as WB
concise "Web User" as WU

@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 Définition participant par participant

Plutôt que de déclarer le diagramme dans l'ordre chronologique, il est possible de le définir participant par participant.

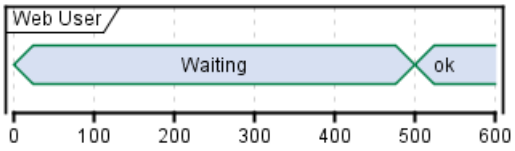
```
@startuml
robust "Web Browser" as WB
concise "Web User" as WU

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

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



9.5 Choix du zoom

Il est possible de choisir une échelle d'affichage précise.

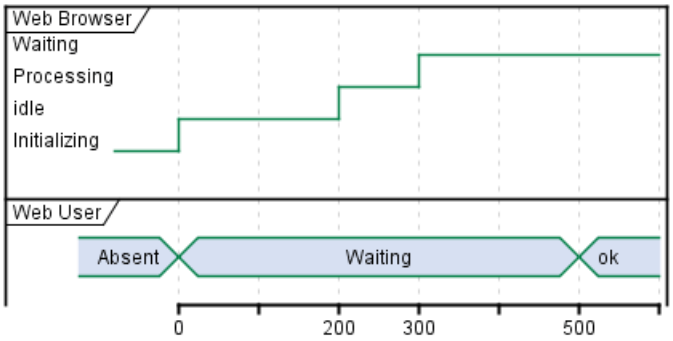
```
@startuml
concise "Web User" as WU
scale 100 as 50 pixels

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



9.6 État initial

Vous pouvez également définir un état initial.

```
@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
```

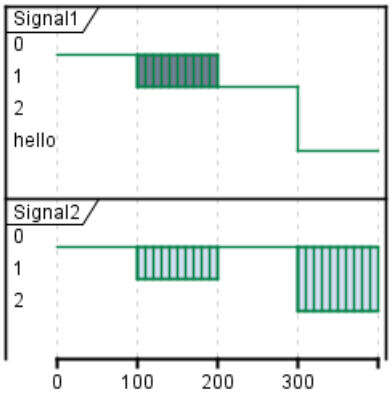


9.7 Intricated state

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
```

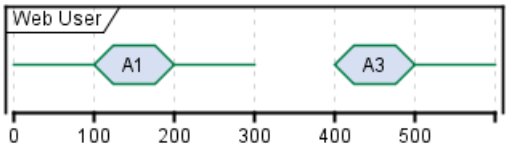
```
S1 is hello
S2 is {0,2}
@enduml
```



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
```

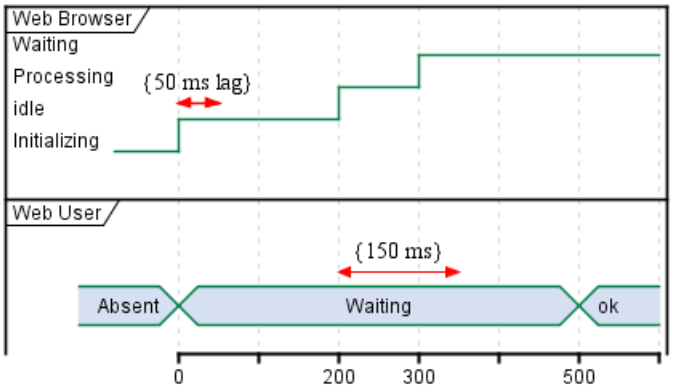


9.9 Ajout de contraintes

Il est possible d'afficher des contraintes de temps sur les diagrammes.

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

WB is Initializing
```

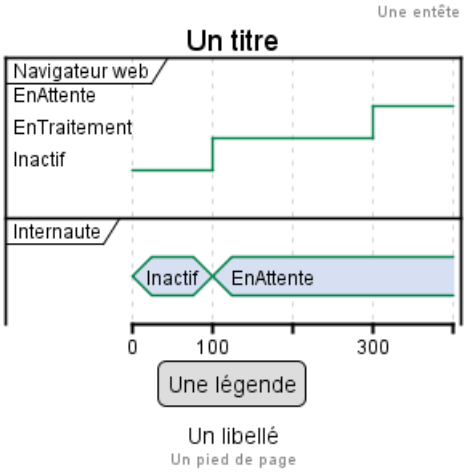
```
WU is Absent

@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 Ajout de textes

Vous pouvez ajouter éventuellement un titre, une entête, un pied de page, une légende ou un libellé :

```
@startuml
Title Un titre
header: Une entête
footer: Un pied de page
legend
Une légende
end legend
caption Un libellé

robust "Navigateur web" as WB
concise "Internaute" as WU

@0
WU is Inactif
WB is Inactif

@100
WU is EnAttente
WB is EnTraitement

@300
WB is EnAttente
@enduml
```


Une entête
Un titre
Navigateur web
EnAttente
EnTraitement
Inactif
Internaute
Inactif
EnAttente
0
100
300
Une légende
Un libellé
Un pied de page

10 Diagramme de Gantt

La syntaxe proposée n'est qu'une suggestion et est susceptible d'évoluer.

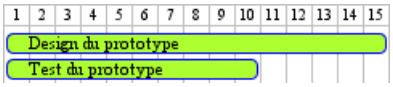
Vous êtes invités à créer des discussions sur cette future syntaxe. Vos retours, vos idées et vos suggestions nous aideront à trouver la meilleure solution.

Le Gantt doit être décrit en anglais, à l'aide de phrase très simple (sujet-verbe-complément).

10.1 Définir des tâches

Les tâches sont définies à l'aide des crochets. Leur durée est défini à l'aide du verbe `last`.

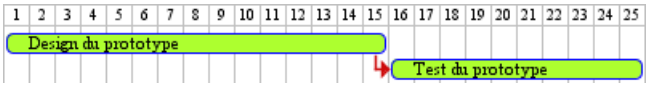
```
@startgantt
[Design du prototype] lasts 15 days
[Test du prototype] lasts 10 days
@endgantt
```



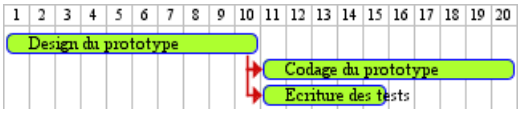
10.2 Ajout de contraintes

Il est possible de rajouter des contraintes entre les tâches.

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



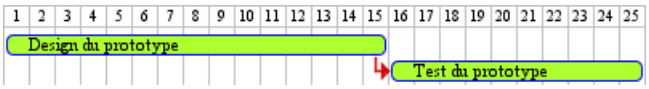

```
@startgantt
[Design du prototype] lasts 10 days
[Codage du prototype] lasts 10 days
[Ecriture des tests] lasts 5 days
[Codage du prototype] starts at [Design du prototype]'s end
[Ecriture des tests] starts at [Codage du prototype]'s start
@endgantt
```



10.3 Noms courts

Un nom court peut être utilisé pour les tâches à l'aide de l'instruction `as`.

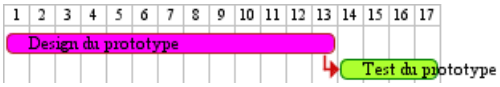
```
@startgantt
[Design du prototype] as [D] lasts 15 days
[Test du prototype] as [T] lasts 10 days
[T] starts at [D]'s end
@endgantt
```



10.4 Choix des couleurs

Il est possible de changer les couleurs des tâches.

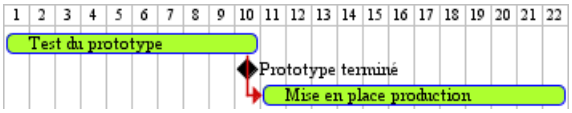
```
@startgantt
[Design du prototype] lasts 13 days
[Test du prototype] lasts 4 days
[Test du prototype] starts at [Design du prototype]'s end
[Design du prototype] is colored in Fuchsia/FireBrick
[Test du prototype] is colored in GreenYellow/Green
@endgantt
```



10.5 Jalon

Vous pouvez définir des jalons à l'aide du verb happens.

```
@startgantt
[Test du prototype] lasts 10 days
[Prototype terminé] happens at [Test du prototype]'s end
[Mise en place production] lasts 12 days
[Mise en place production] starts at [Test du prototype]'s end
@endgantt
```



10.6 Calendrier

Vous pouvez définir une date de début pour l'ensemble du projet. Par défault, la première tâche commence à cette date.

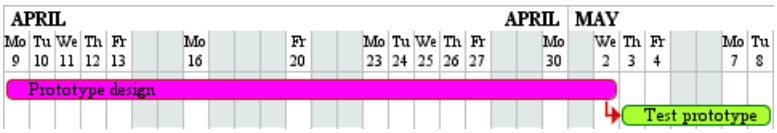
```
@startgantt
Project starts the 20th of september 2017
[Design du prototype] as [TASK1] lasts 13 days
[TASK1] is colored in Lavender/LightBlue
@endgantt
```

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
```

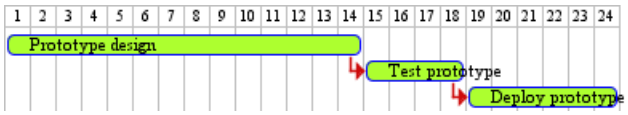
```
[Test prototype] is colored in GreenYellow/Green
@endgantt
```



10.8 Simplified task succession

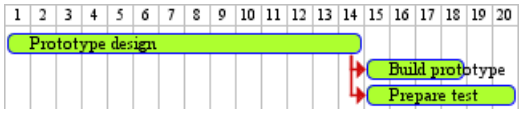
It's possible to use the `then` keyword to denote consecutive tasks.

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



You can also use arrow `->`

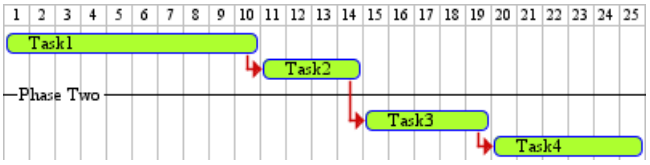
```
@startgantt
[Prototype design] lasts 14 days
[Build prototype] lasts 4 days
[Prepare test] lasts 6 days
[Prototype design] -> [Build prototype]
[Prototype design] -> [Prepare test]
@endgantt
```



10.9 Separator

You can use `--` to group tasks together.

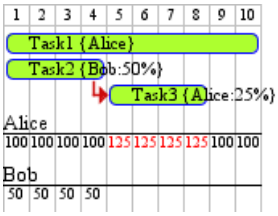
```
@startgantt
[Task1] lasts 10 days
then [Task2] lasts 4 days
-- Phase Two --
then [Task3] lasts 5 days
then [Task4] lasts 6 days
@endgantt
```



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
```

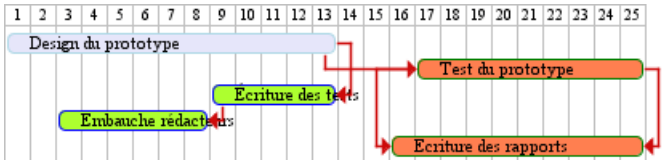


10.11 Exemple plus complexe

On peut se servir de la conjonction de coordination and.

Il est aussi possible de spécifier un delai dans les contraintes.

```
@startgantt
[Design du prototype] lasts 13 days and is colored in Lavender/LightBlue
[Test du prototype] lasts 9 days and is colored in Coral/Green and starts 3 days after [Design du prototype
[Écriture des tests] lasts 5 days and ends at [Design du prototype]'s end
[Embauche des rédacteurs] lasts 6 days and ends at [Écriture des tests]'s start
[Écriture des rapports] is colored in Coral/Green
[Écriture des rapports] starts 1 day before [Test du prototype]'s start and ends at [Test du prototype]'s
@endgantt
```



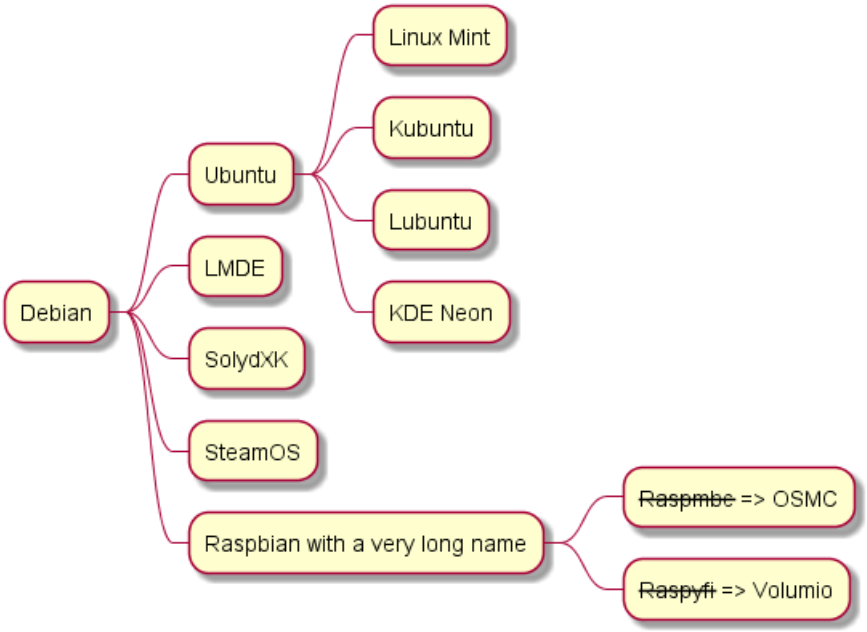
11 MindMap

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
```

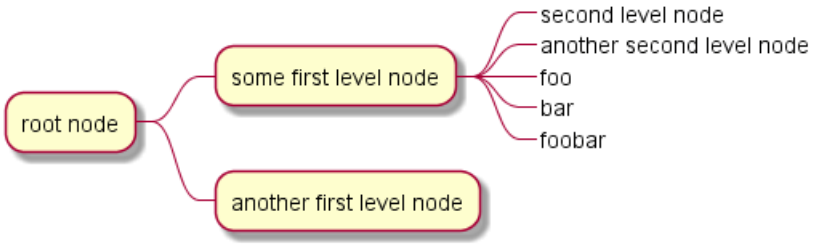


11.2 Removing box

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
```

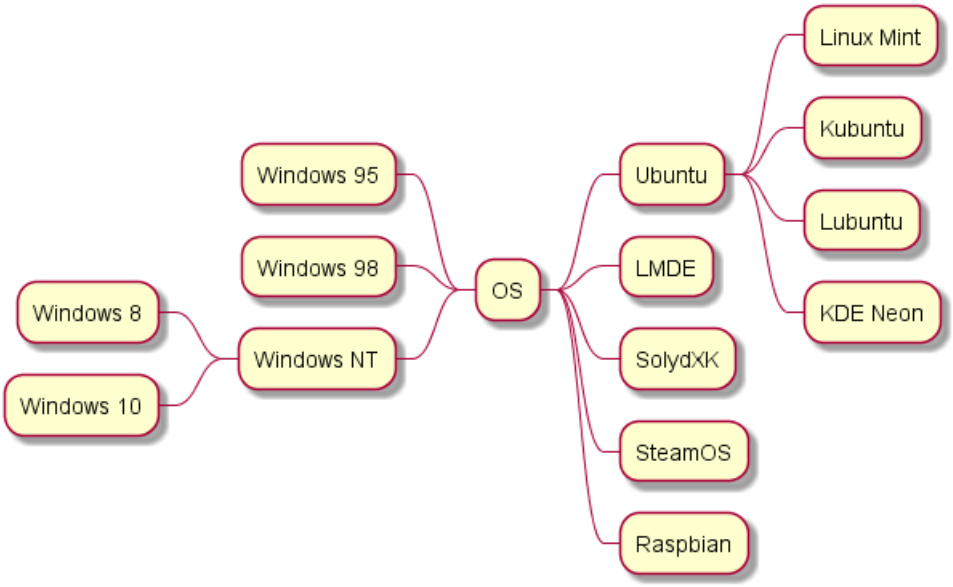
```
@endmindmap
```



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
```

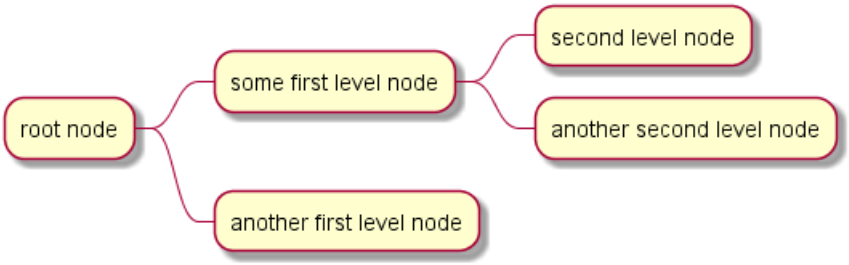


11.4 Markdown syntax

This syntax is compatible with Markdown

```
@startmindmap
```

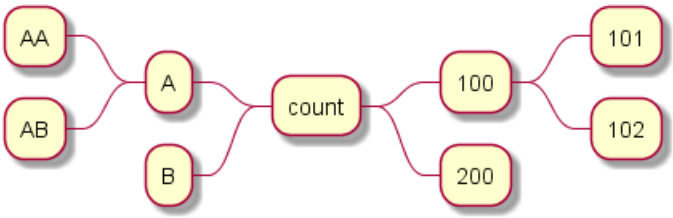
```
* root node
* some first level node
* second level node
* another second level node
* another first level node
@endmindmap
```



11.5 Changing diagram direction

It is possible to use both sides of the diagram.

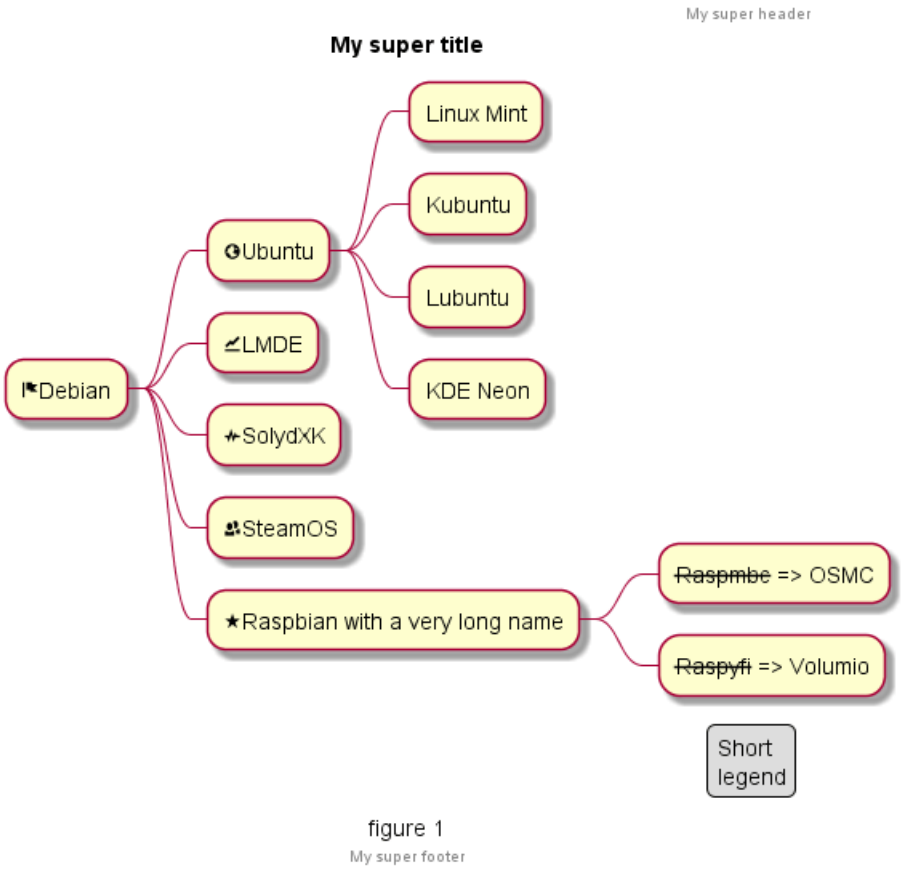
```
@startmindmap
* count
** 100
*** 101
*** 102
** 200

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
```



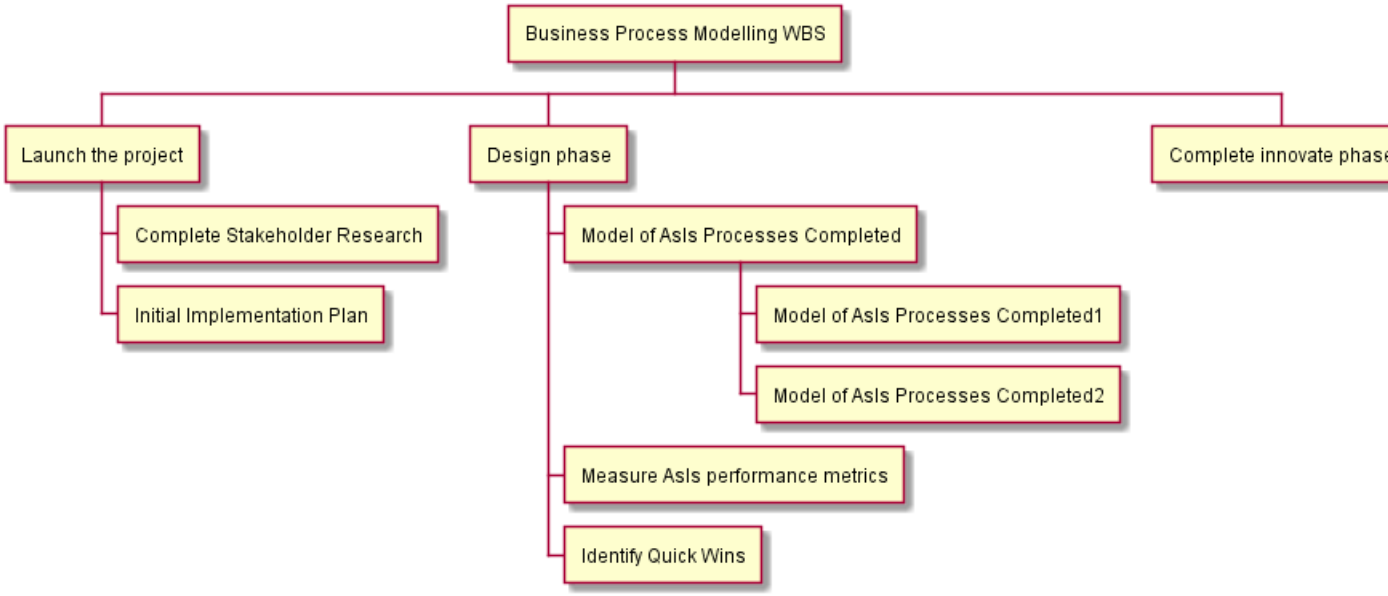
12 Work Breakdown Structure

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
```

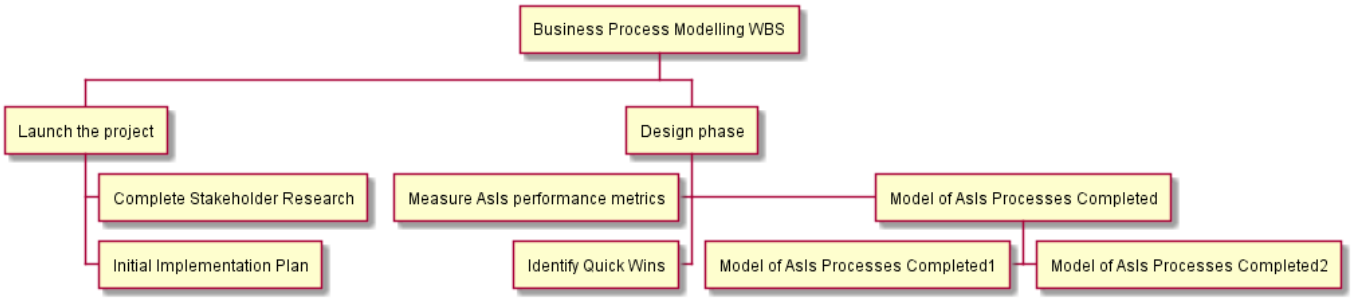


12.2 Change direction

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
```

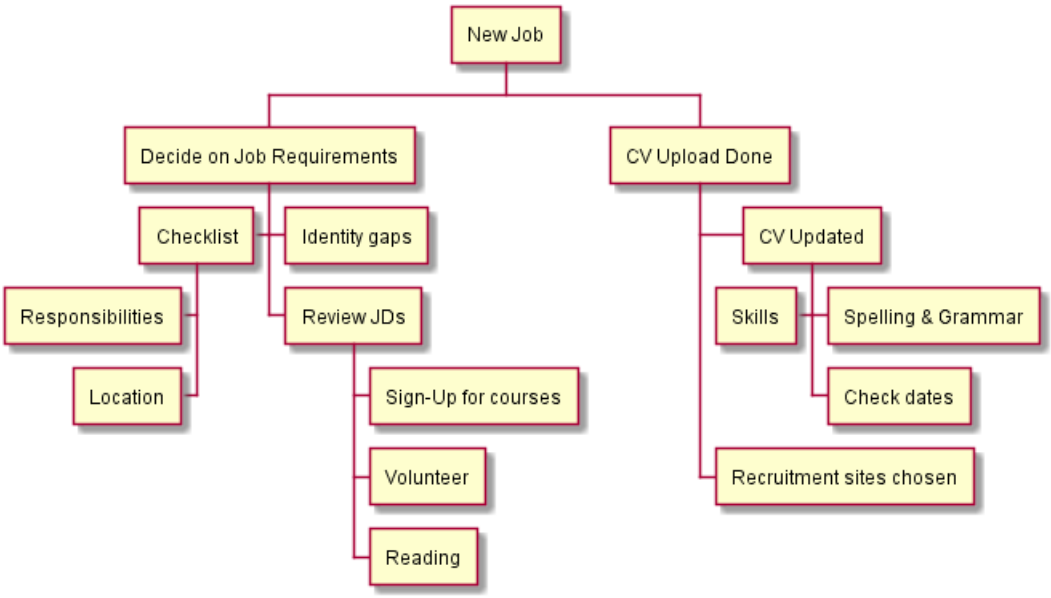
```
@endwbs
```



12.3 Arithmetic notation

You can use the following notation to choose diagram side.

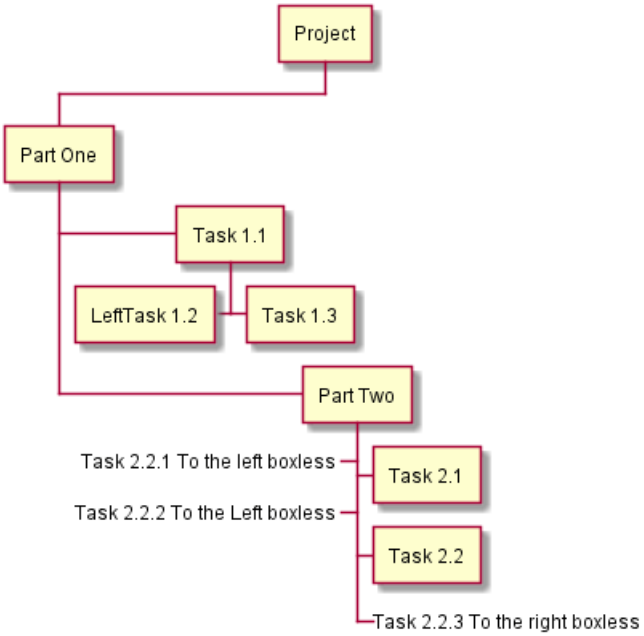
```
@startwbs
+ New Job
++ Decide on Job Requirements
+++ Identity gaps
+++ Review JDs
++++ Sign-Up for courses
++++ Volunteer
++++ Reading
++- Checklist
+++- Responsibilities
+++- Location
++ CV Upload Done
+++ CV Updated
++++ Spelling & Grammar
++++ Check dates
---- Skills
+++ Recruitment sites chosen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



You can use underscore _ to remove box drawing.

```
@startwbs
+ Project
```

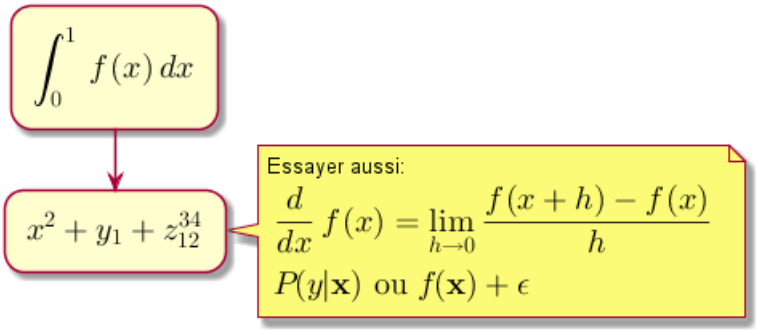
```
 + Part One
  + Task 1.1
   - LeftTask 1.2
   + Task 1.3
  + Part Two
   + Task 2.1
   + Task 2.2
   -_ Task 2.2.1 To the left boxless
   -_ Task 2.2.2 To the Left boxless
   +_ Task 2.2.3 To the right boxless
@endwbs
```



13 Mathématiques

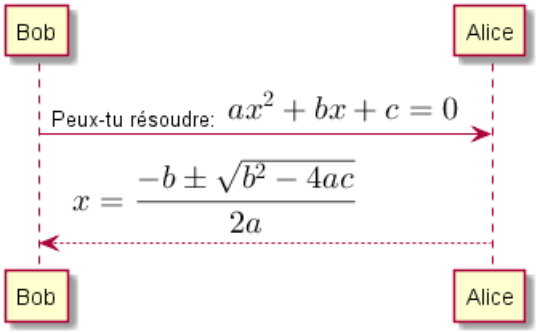
Vous pouvez utiliser les notations AsciiMath ou JLaTeXMath dans PlantUML:

```
@startuml
:<math>int_0^1f(x)dx</math>;
:<math>x^2+y_1+z_12^34</math>;
note right
Essayer aussi:
<math>d/dxf(x)=lim_(h->0)(f(x+h)-f(x))/h</math>
<latex>P(y|\mathbf{x}) \mbox{ ou } f(\mathbf{x})+\epsilon</latex>
end note
@enduml
```



ou encore:

```
@startuml
Bob -> Alice : Peux-tu résoudre: <math>ax^2+bx+c=0</math>
Alice --> Bob: <math>x = (-b+-sqrt(b^2-4ac))/(2a)</math>
@enduml
```



13.1 Diagramme indépendant

Il est possible d'utiliser `@startmath`/`@endmath` pour créer des formules AsciiMath.

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

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

Ou bien utiliser `@startlatex`/`@endlatex` pour créer des formules JLaTeXMath.

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

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

13.2 Comment cela fonctionne ?

Pour dessiner ces formules, PlantUML utilise deux projets OpenSource:

- AsciiMath qui convertit la notation AsciiMath vers une expression LaTeX.
- JLatexMath qui dessine une formule mathématique écrite en LaTeX. JLaTeXMath est le meilleur projet Java pour dessiner du code LaTeX.

ASCIIMathTeXImg.js est suffisamment petit pour être intégré dans la distribution standard de PlantUML.

Comme JLatexMath est plus gros, vous devez le télécharger séparément, puis extraire les 4 fichiers (*batik-all-1.7.jar*, *jlatexmath-minimal-1.0.3.jar*, *jlm_cyrillic.jar* et *jlm_greek.jar*) dans le même répertoire que 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

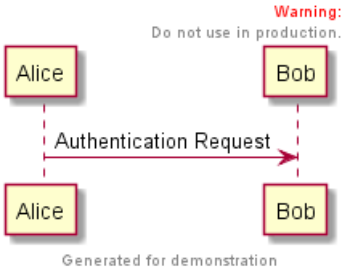
You can use the commands `header` or `footer` to add a footer or a header on any generated diagram.

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
```



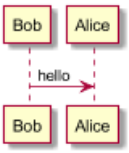
14.3 Zoom

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
```

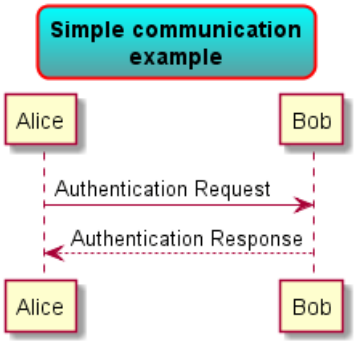


14.4 Title

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.

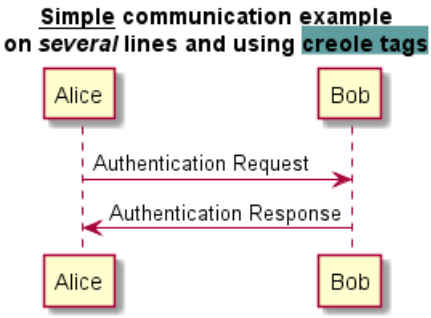
```
@startuml
skinparam titleBorderRoundCorner 15
skinparam titleBorderThickness 2
skinparam titleBorderColor red
skinparam titleBackgroundColor Aqua-CadetBlue

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.

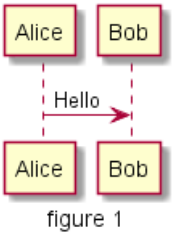
```
@startuml

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
```

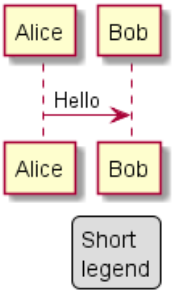


14.6 Legend the diagram

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

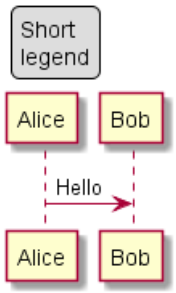
You can optionally specify to have `left`, `right`, `top`, `bottom` or `center` alignment for the legend.

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




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

```
  legend
endlegend
@enduml
```



15 Salt

Salt est un sous projet inclus dans PlantUML qui peut vous aider à modeler une interface graphique.

Vous pouvez utiliser sois le mot clé `@startsalt`, ou bien `@startuml` suivi par une ligne avec le mot clé `salt`.

15.1 widgets de base

Une fenêtre doit commencer et finir par une accolade. Vous pouvez ensuite définir:

- Un bouton en utilisant `[` et `]`.
- Un bouton radio en utilisant `(` et `)`.
- Une case à cocher en utilisant `[` et `]`.
- Zone de texte utilisateur utilisant `"`.

```
@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
```

Le but de cet outil est de discuter des échantillons de fenêtres simples.

15.2 Utilisation de grille

Un tableau est créé automatiquement en utilisant une accolade ouvrante `{`.

Il faut utiliser `|` pour séparer les colomnes.

Par exemple:

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

Tout de suite après l'accolade ouvrante, vous pouvez utiliser un caractère pour définir si vous voulez dessiner les lignes ou les colonnes de la grille :

Symbol	Result
#	Pour afficher toutes les lignes verticales et horizontales
!	Pour afficher toutes les lignes verticales
-	Pour afficher toutes les lignes horizontales
+	Pour afficher les lignes extérieurs

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

15.3 Group box

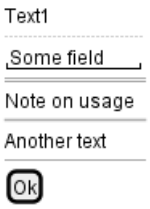
more info

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

15.4 Utilisation des séparateurs

Vous pouvez utiliser de nombreuses lignes horizontales en tant que séparateur.

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



15.5 Widget d'arbre

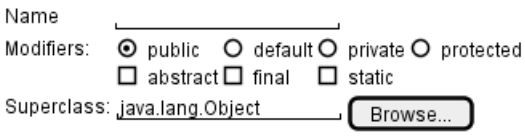
Pour faire un arbre vous devez commencer avec `{T` et utiliser `+` pour signaler la hiérarchie.

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

15.6 Accolades délimitantes

Vous pouvez définir des sous-éléments en créant une accolade ouvrante.

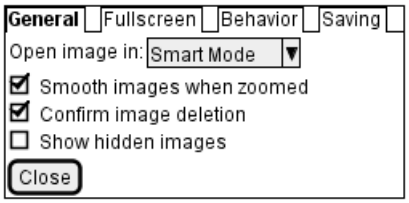
```
@startsalt
{
Name          | "                    "
Modifiers:    | { (X) public | () default | () private | () protected
             [] abstract | [] final   | [] static }
Superclass:   | { "java.lang.Object " | [Browse...] }
}
@endsalt
```



15.7 Ajout d'onglet

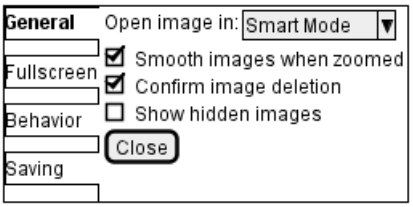
Vous pouvez ajouter des onglets avec la notation `{/`. Notez que vous pouvez utiliser du code HTML pour avoir un texte en gras.

```
@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
```



Les onglets peuvent également être orientés verticalement:

```
@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
```

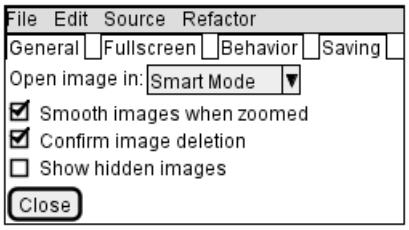


15.8 Utiliser les menus

Vous pouvez ajouter un menu en utilisant la notation `{*`.

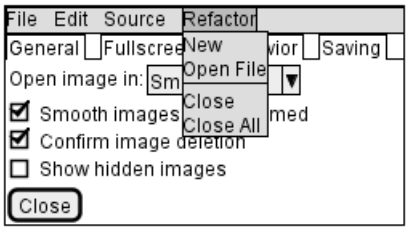
```
@startsalt
{+
{* File | Edit | Source | Refactor }
{/ General | Fullscreen | Behavior | Saving }
{
{ Open image in: | ^Smart Mode^ }
```

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



Il est également possible d'ouvrir un 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 Tableaux avancés

Vous pouvez utiliser deux notations spéciales pour les tableaux :

- `*` pour indiquer que la cellule de gauche peut s'étendre sur l'actuelle
- `.` pour indiquer une cellule vide

```
@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:

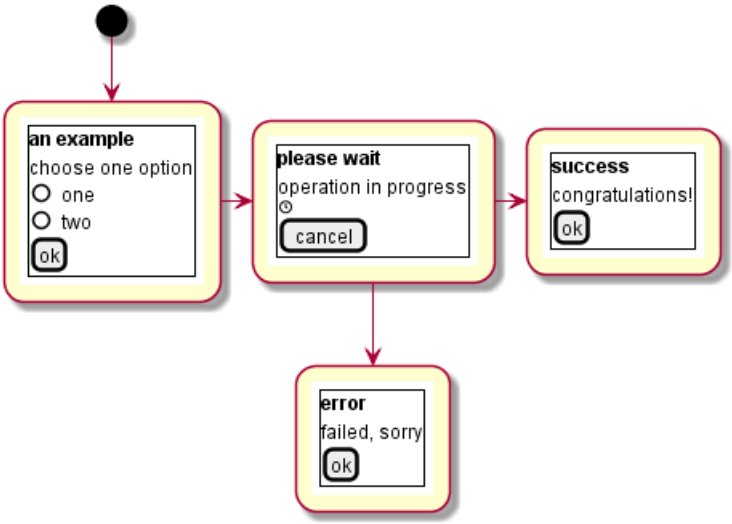
```
@startuml
listopeniconic
@enduml
```

15.11 Include Salt

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

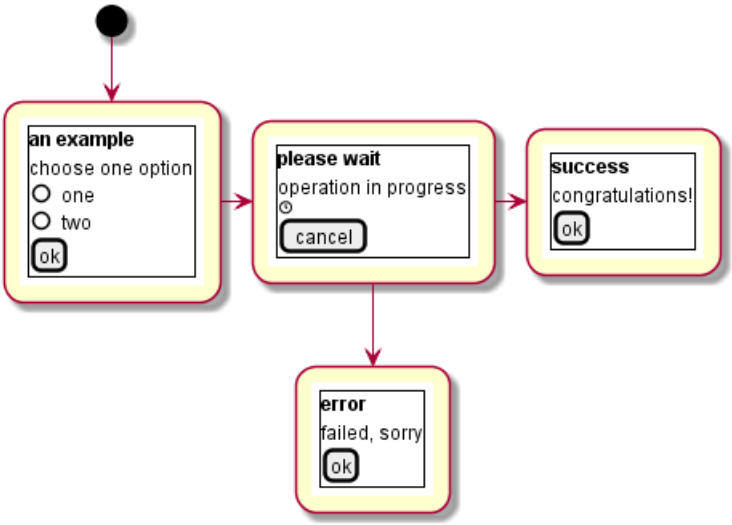
```
@startuml
(*) --> "
{{
salt
{+
<b>an example
choose one option
```

```
()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.

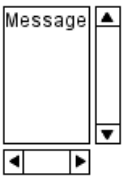
```
@startuml
!unquoted function SALT($x)
"{{
salt
%invoke_void_func("_"+$x)
}}" as $x
!endfunction

!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 Créole

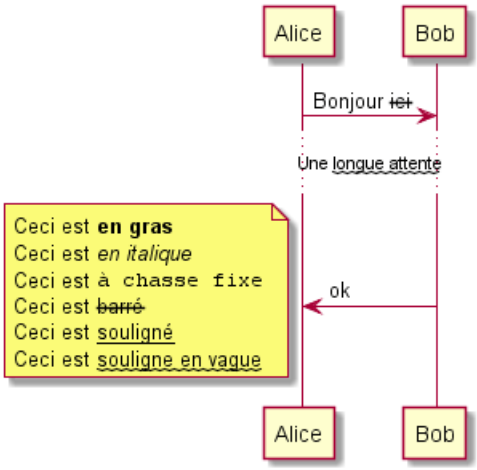
Un petit moteur Créole a été intégré à PlantUML pour pouvoir formater les textes de façon standardisé.

Tous les diagrammes intègrent cette syntaxe.

Notez qu'une compatibilité ascendante avec la syntaxe HTML a été conservée.

16.1 Formatage de texte

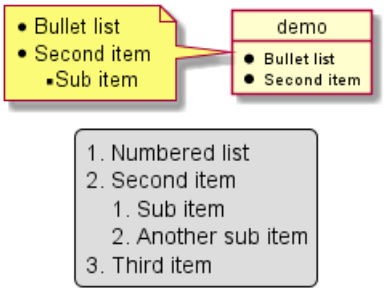
```
@startuml
Alice -> Bob : Bonjour --ici--
... Une ~~longue attente~~ ...
Bob -> Alice : ok
note left
  Ceci est **en gras**
  Ceci est //en italique//
  Ceci est ""à chasse fixe""
  Ceci est --barré--
  Ceci est __souligné__
  Ceci est ~~souligne en vague~~
end note
@enduml
```



16.2 Listes

```
@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
```

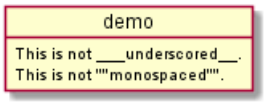
```
  # Third item
end legend
@enduml
```



16.3 Caractère d'échappement

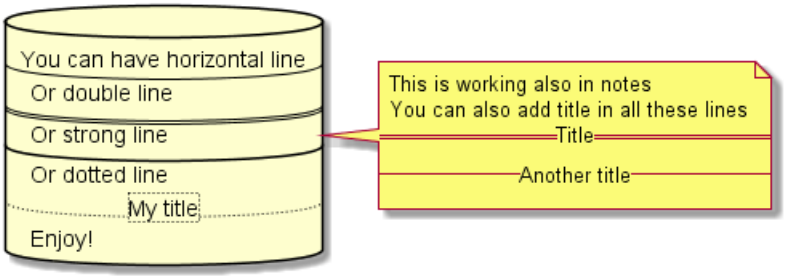
Vous pouvez utiliser le tilde ~ pour échapper les caractères Créoles spéciaux.

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



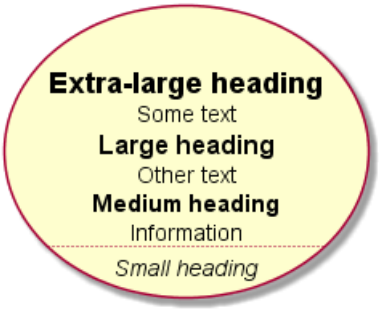
16.4 Lignes horizontales

```
@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 Entêtes

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



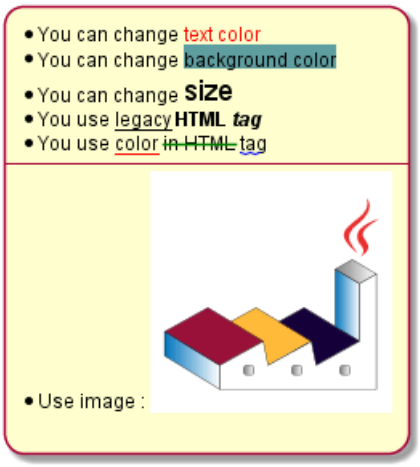
16.6 Tag HTML

Certains tag HTML sont encore fonctionnels:

- `<b>` pour du texte en gras
- `<u>` ou `<u:#AAAAAA>` ou `<u:colorName>` pour souligner
- `<i>` pour de l'italique
- `<s>` ou `<s:#AAAAAA>` ou `<s:colorName>` pour barrer du texte
- `<w>` ou `<w:#AAAAAA>` ou `<w:colorName>` pour souligner en vague
- `<color:#AAAAAA>` ou `<color:colorName>` pour la couleur
- `<back:#AAAAAA>` ou `<back:colorName>` pour la couleur de fond
- `<size:nn>` pour changer la taille des caractères
- `<img:file>` : le fichier doit être accessible sur le système de fichier
- `<img:http://plantuml.com/logo3.png>` : l'URL doit être accessible

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

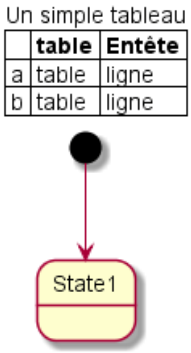
```
* You use <u>legacy</u> <b>HTML <i>tag</i></b>
* You use <u:red>color</u> <s:green>in HTML</s> <w:#0000FF>tag</w>
----
* Use image : <img:http://plantuml.com/logo3.png>
;
@enduml
```



16.7 Tableau

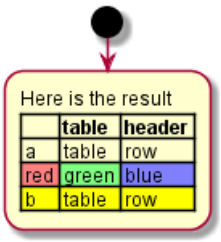
Il est possible de construire des tableaux.

```
@startuml
skinparam titleFontSize 14
title
  Un simple tableau
  |= |= table |= Entête |
  | a | table | ligne |
  | b | table | ligne |
end title
[*] --> State1
@enduml
```



Il est possible de changer la couleur de fond des cellules et des lignes.

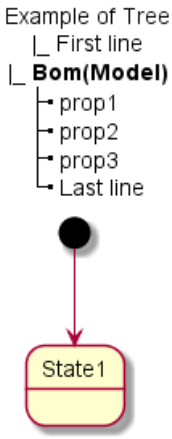
```
@startuml
start
:Here is the result
|= |= table |= header |
| a | table | row |
|<#FF8080> red |<#80FF80> green |<#8080FF> blue |
<#yellow>| b | table | row |;
@enduml
```



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>

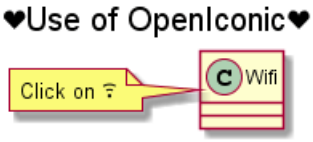
```
@startuml
usecase foo as "this is &#8734; long"
usecase bar as "this is also <U+221E> long"
@enduml
```

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
Credit to
https://useiconic.com/open

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

17 Defining and using sprites

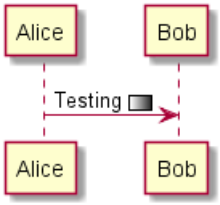
A *Sprite* is a small graphic element that can be used in diagrams.

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

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

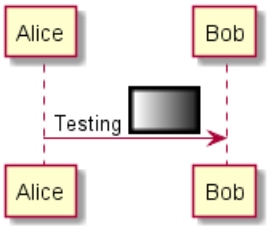
Then you can use the sprite using `<$XXX>` where XXX is the name of the sprite.

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



You can scale the sprite.

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



17.1 Encoding Sprite

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

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

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

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

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

17.2 Importing Sprite

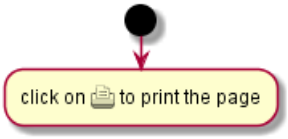
You can also launch the GUI to generate a sprite from an existing image.

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

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

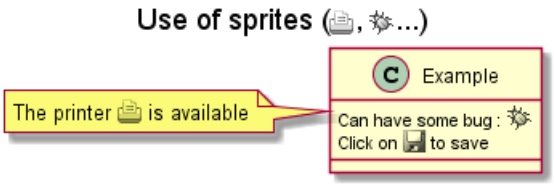
17.3 Examples

```
@startuml
sprite $printer [15x15/8z] NOtH3W0W208HxFz_kMAhj7lHWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvF
start
:click on <$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:

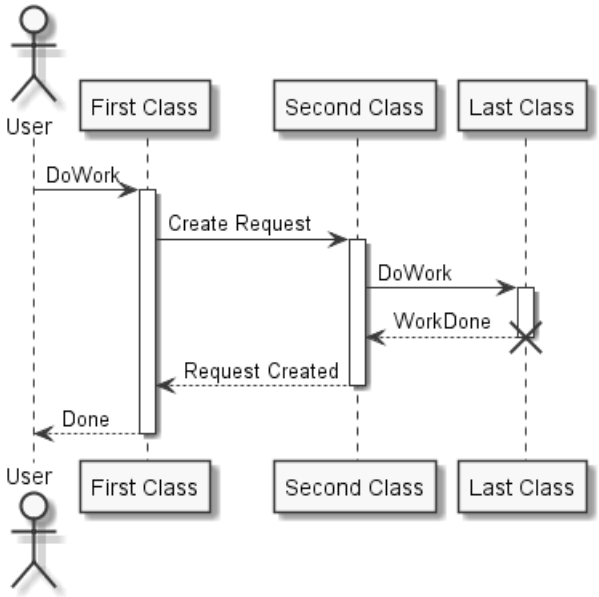
```
skinparam xxxx {
    Param1 value1
    Param2 value2
    Param3 value3
    Param4 value4
}
```

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
```

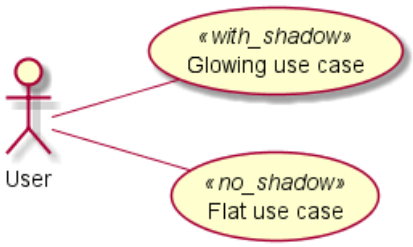
```
deactivate B

A --> User: Done
deactivate A

@enduml
```



18.4 Shadowing

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

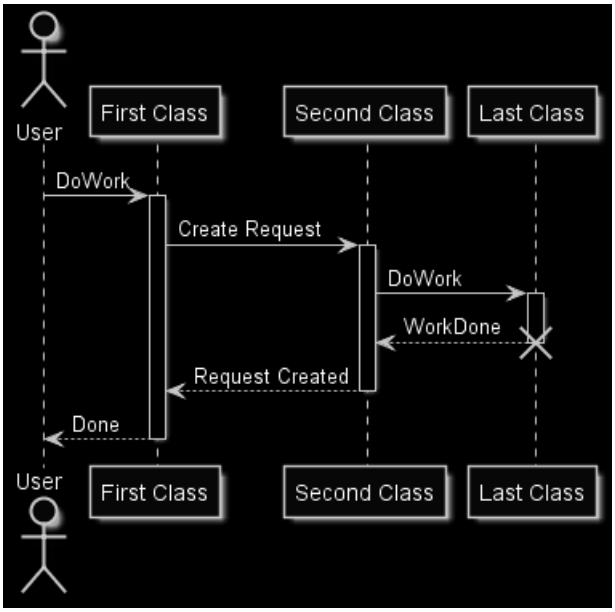
```
@startuml

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:

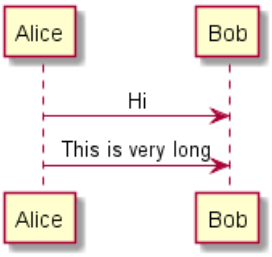
```
java -jar plantuml.jar -language
```

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.

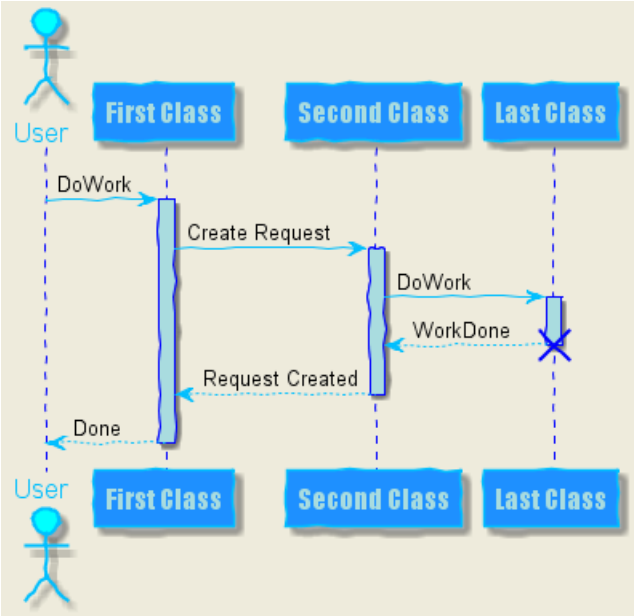
Param name	Default value	Comment
sequenceMessageAlign	left	Used for messages in sequence diagrams
sequenceReferenceAlign	center	Used for `ref over` in sequence diagrams

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

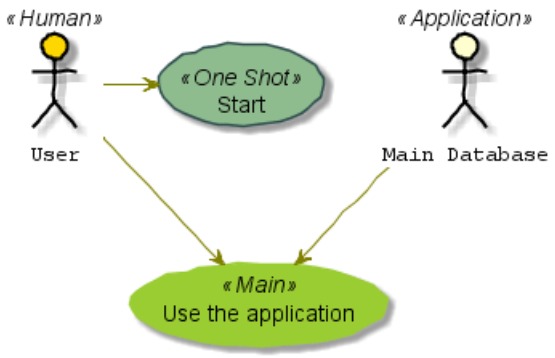


18.9 Examples

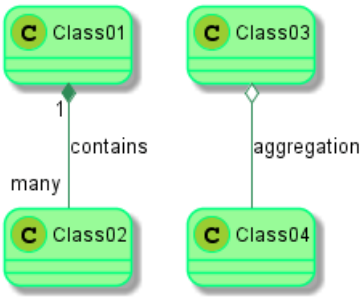
```
@startuml
skinparam backgroundColor #EEEBDC
skinparam handwritten true

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
```

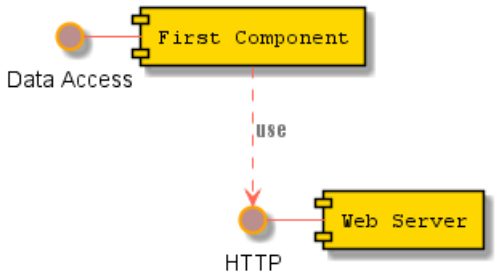



```
@startuml
skinparam roundcorner 20
skinparam class {
BackgroundColor PaleGreen
ArrowColor SeaGreen
BorderColor SpringGreen
}
skinparam stereotypeCBackgroundColor YellowGreen

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

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

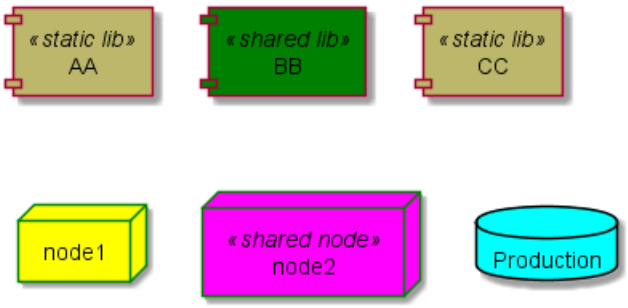



```
@startuml

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

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

Please contact us if you have any issues.

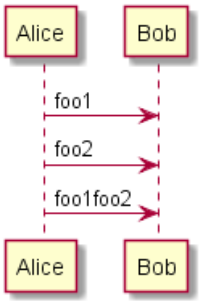
19.2 Variable definition

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

- Integer number
- String - these must be surrounded by single quote or double quote.

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

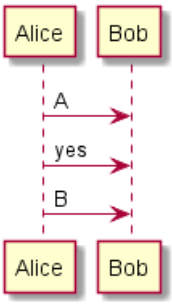
```
@startuml
!$ab = "foo1"
!$cd = "foo2"
!global $ef = $ab + $cd

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



19.3 Conditions

- You can use expression in condition.
- *else* is also implemented

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

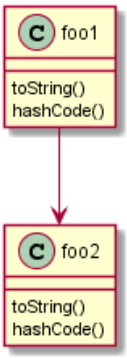


19.4 Void function

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

Example:

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

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


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


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



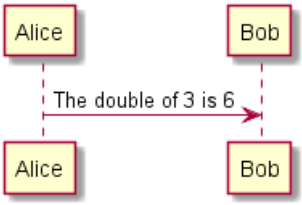
Variables defined in functions are **local**. It means that the variable is destroyed when the function ends.

19.5 Return function

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

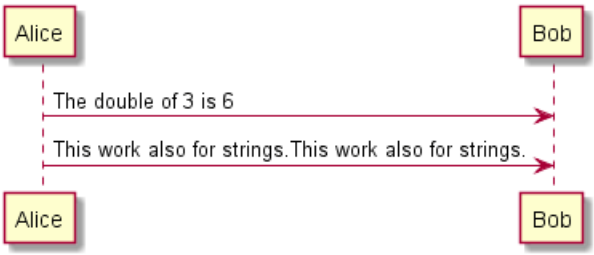
- directly in variable definition or in diagram text
- from other return function
- from other void function
- Function name *should* start by a $
- Argument names *should* start by a $

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

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



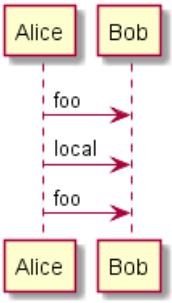
It is possible to shorten simple function definition in one line:

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

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



As in void function, 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.

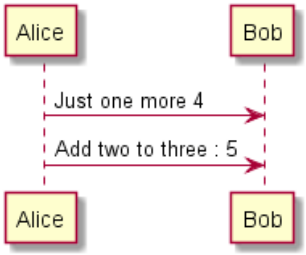
```
@startuml
!function $dummy()
!local $ijk = "local"
Alice -> Bob : $ijk
!endfunction

!global $ijk = "foo"

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



19.6 Default argument value

In both return and void functions, you can define default values for arguments.

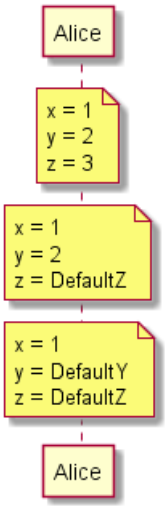
```
@startuml
!function $inc($value, $step=1)
!return $value + $step
!endfunction

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



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

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

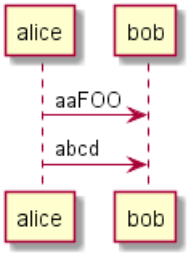
```
defaulttest(1, 2, 3)
defaulttest(1, 2)
defaulttest(1)
@enduml
```



19.7 Unquoted function

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

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

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

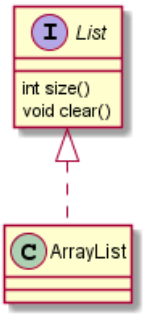


19.8 Including files or URL

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

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

```
@startuml

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



File List.iuml

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

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

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

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

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

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

19.9 Including Subpart

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

file1.puml:

```
@startuml

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

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

```
@startuml

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

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

file2.puml

```
@startuml

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

This file would be rendered exactly as if:

```
@startuml

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

19.10 Builtin functions

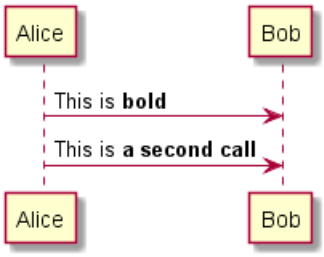
Some functions are defined by default. Their name starts by %

Name	Description	
`%strlen`	Calculate the length of a String	%
`%substr`	Extract a substring. Takes 2 or 3 arguments| `%substr("abcdef", 3, 2)`	"d
`%strpos`	Search a substring in a string	%strpo
`%intval`	Convert a String to Int	%
`%file_exists`	Check if a file exists on the local filesystem	%file_exis
`%function_exists`	Check if a function exists	%function_e
`%variable_exists`	Check if a variable exists	%variable_
`%set_variable_value`	Set a global variable	%set_variable_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.

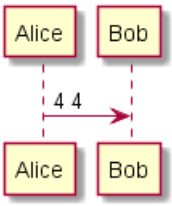
```
@startuml
!function bold($text)
!$result = "<b>"+ $text +"</b>"
!log Calling bold function with $text. The result is $result
!return $result
!endfunction

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.

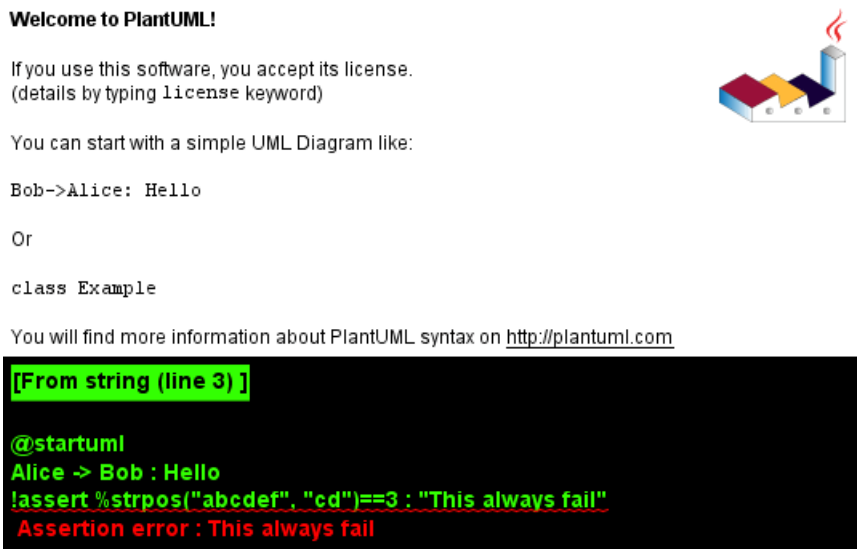
```
@startuml
!function $inc($string)
!$val = %intval($string)
!log value is $val
!dump_memory
!return $val+1
!endfunction

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
```

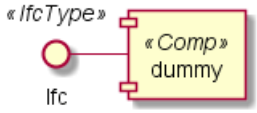
Note the this -D option has to put before the -jar option. -D options after the -jar option will be used to define constants within plantuml preprocessor.

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
```

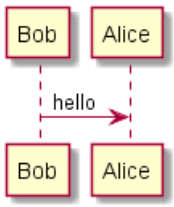


19.17 Dynamic function invocation

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.

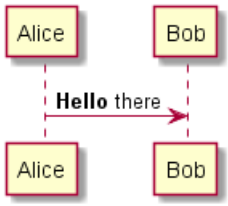
For example, you can have:

```
@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
```



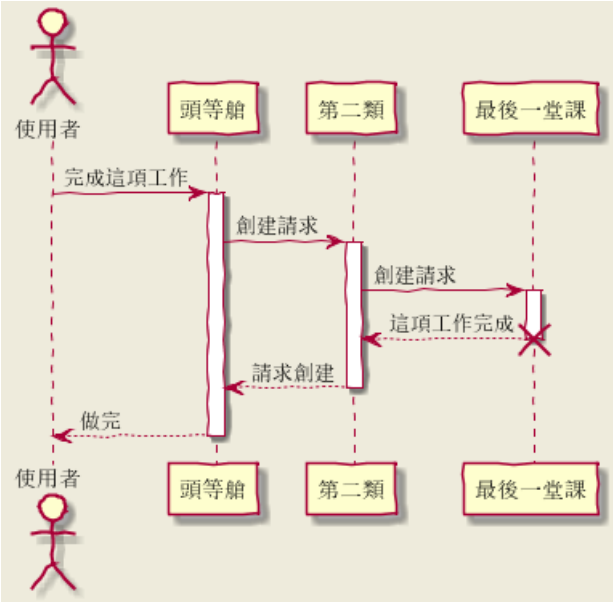
20 Unicode

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*.

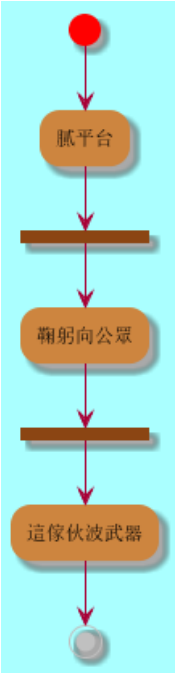
20.1 Examples

```
@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
```

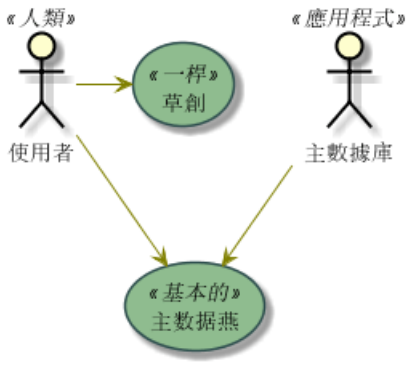



```
@startuml

(*) --> "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
```



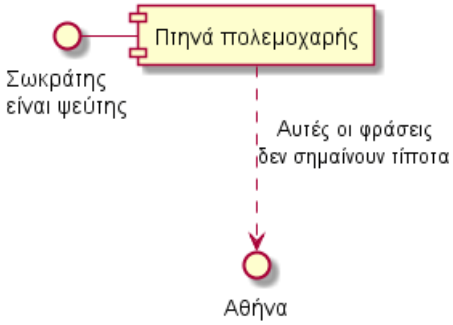
@startuml

() "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

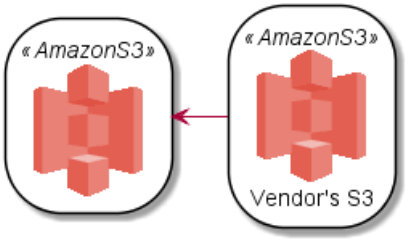
The AWS library consists of Amazon AWS icons, it provides icons of two different sizes.

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.

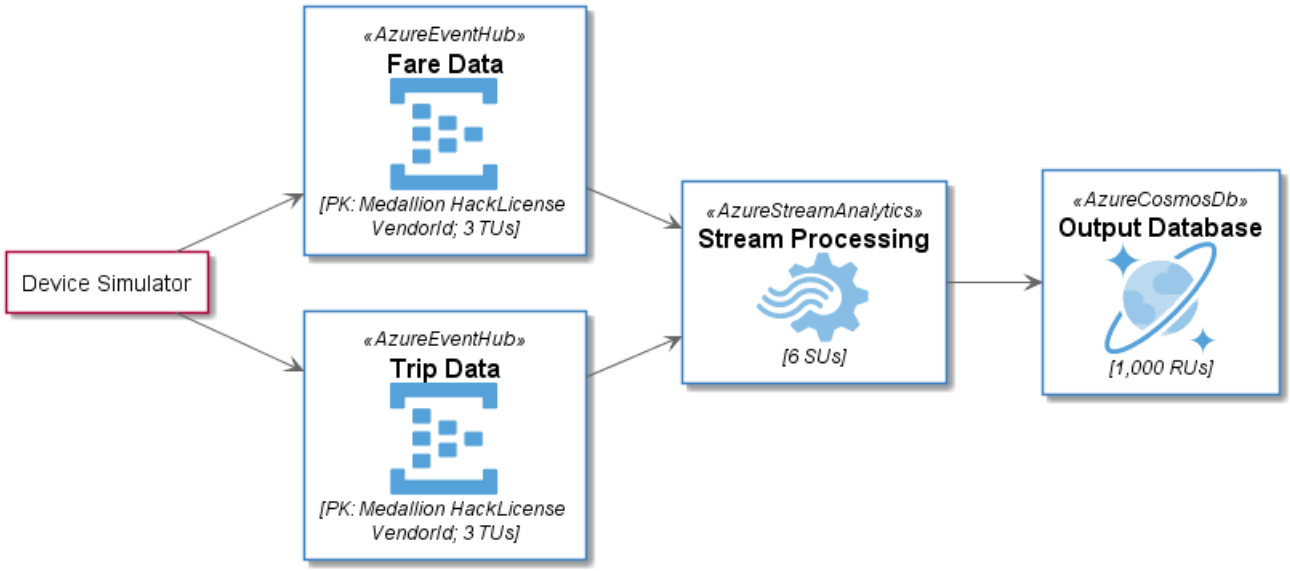
Use it by including the file that contains the sprite, eg: `!include <azure/Analytics/AzureEventHub.puml>`. When imported, you can use the sprite as normally you would, using `<$sprite_name>`.

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
```



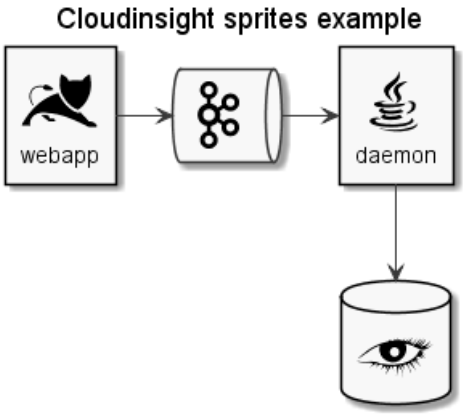
21.3 Cloud Insight

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

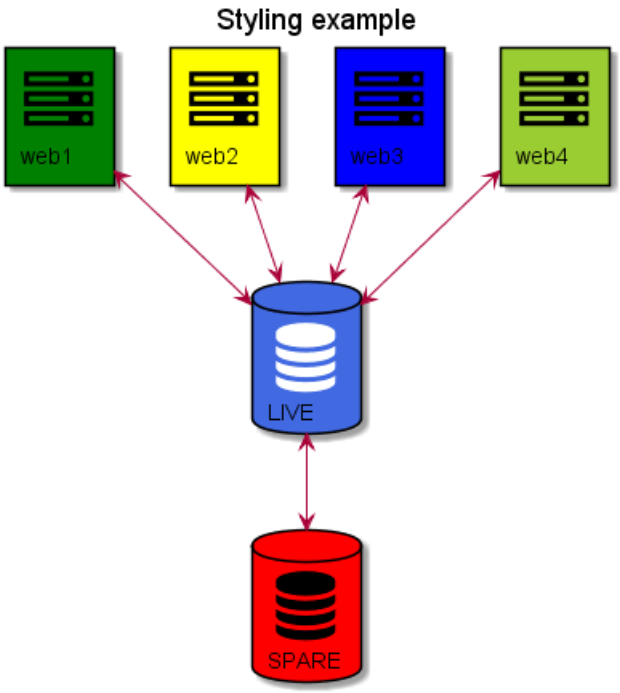
These two library consists respectively of Devicons and Font Awesome libraries of icons.

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>`.

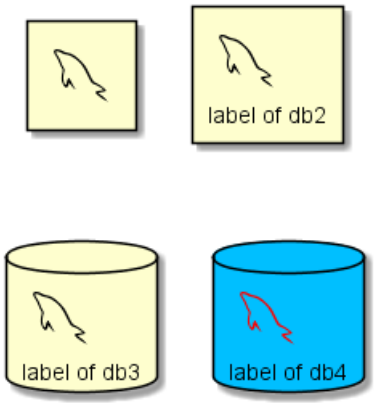
You may also include the `common.puml` file, eg: `!include <font-awesome/common>`, which contains helper macros defined. With the `common.puml` imported, you can use the `NAME_OF_SPRITE(parameters...)` macro.

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.

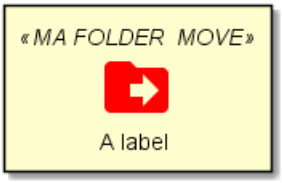
Use it by including the file that contains the sprite, eg: `!include <material/ma_folder_move>`. When imported, you can use the sprite as normally you would, using `<$ma_sprite_name>`. Notice that this library requires an `ma_` prefix on sprites names, this is to avoid clash of names if multiple sprites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

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

again the use of the prefix `MA_`.

Example of usage:

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

MA_FOLDER_MOVE(Red, 1, dir, rectangle, "A label")
@enduml
```

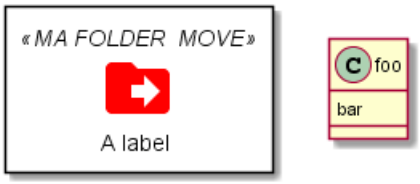


Notes

When mixing sprites macros with other elements you may get a syntax error if, for example, trying to add a rectangle along with classes. In those cases, add `{` and `}` after the macro to create the empty rectangle.

Example of usage:

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

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

class foo {
bar
}
@enduml
```



21.6 Office

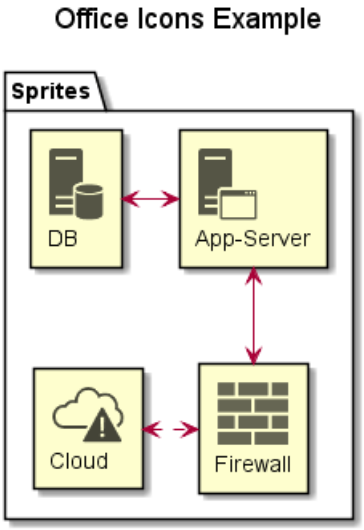
https://github.com/Roemer/plantuml-office

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

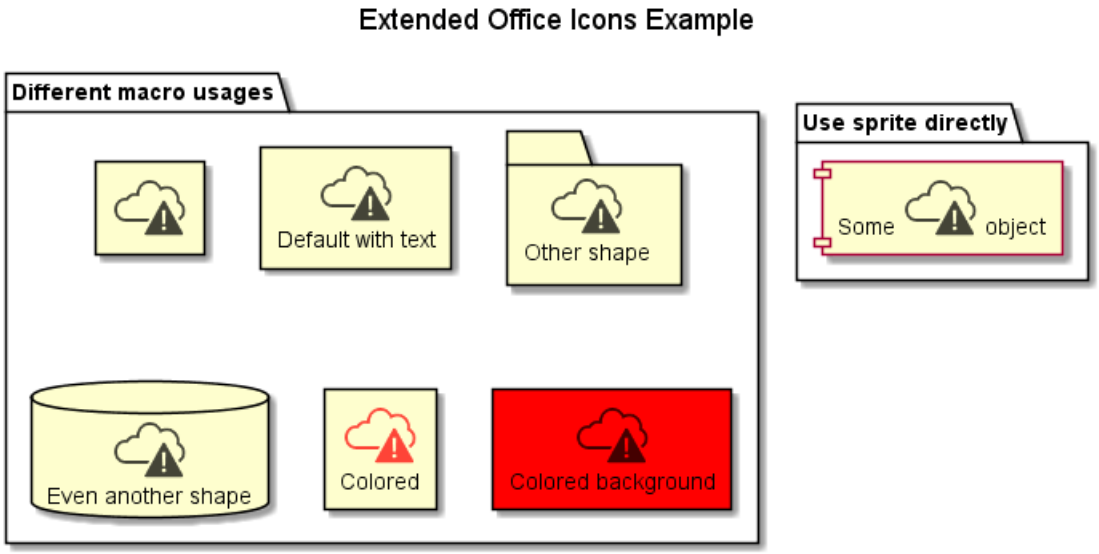
Example of usage:

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

!include <office/Servers/database_server>
!include <office/Servers/application_server>
!include <office/Concepts/firewall_orange>
!include <office/Clouds/cloud_disaster_red>
```

```
title Office Icons Example

package "Sprites" {
OFF_DATABASE_SERVER(db,DB)
OFF_APPLICATION_SERVER(app,App-Server)
OFF_FIREWALL_ORANGE(fw,Firewall)
OFF_CLOUD_DISASTER_RED(cloud,Cloud)
db <-> app
app <--> fw
fw <.left.> cloud
}

@enduml
```




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

!include <office/servers/database_server>
!include <office/servers/application_server>
!include <office/Concepts/firewall_orange>
!include <office/Clouds/cloud_disaster_red>

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

title Extended Office Icons Example

package "Use sprite directly" {
[Some <$cloud_disaster_red> object]
}

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



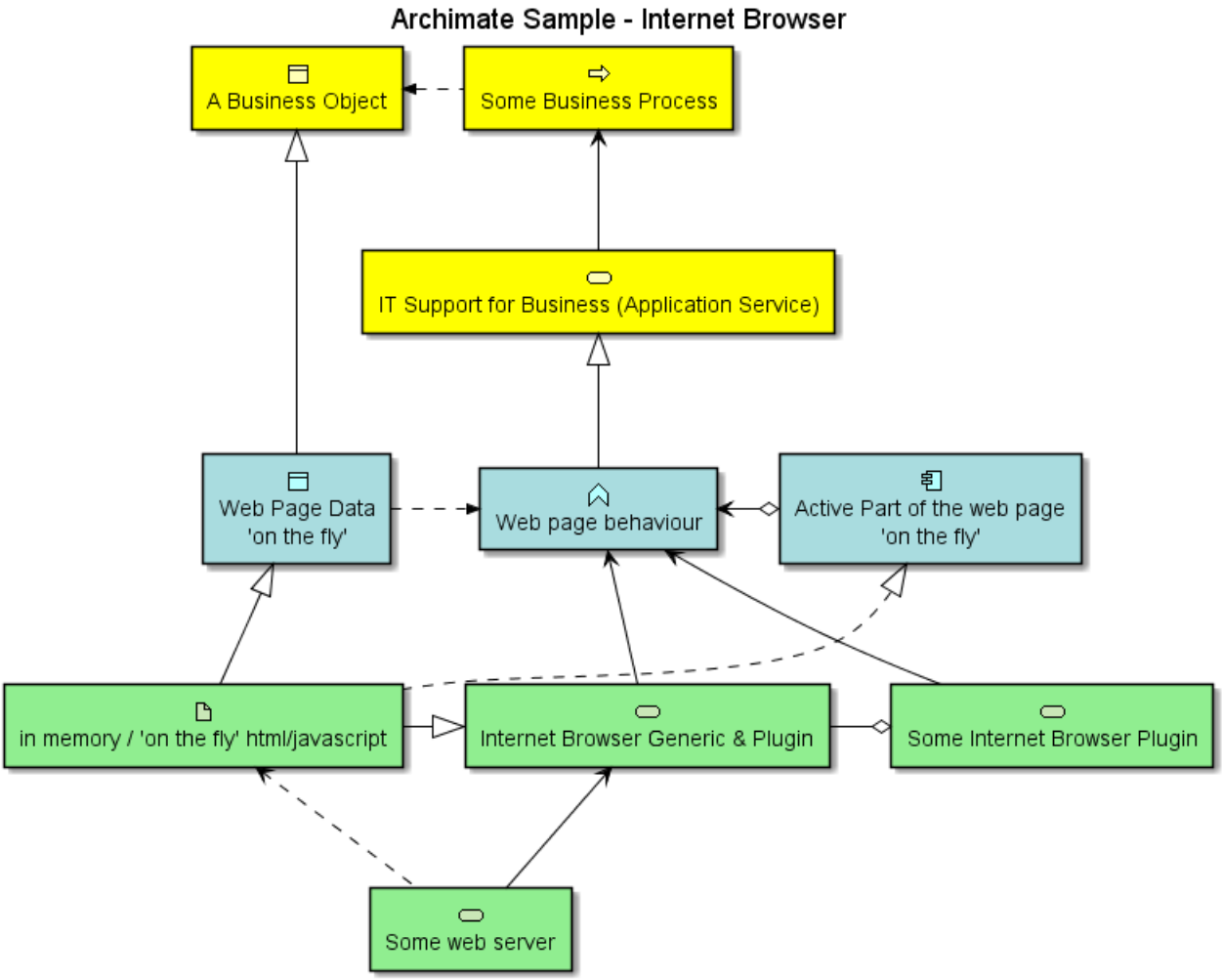
21.7 ArchiMate

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

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

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

title Archimate Sample - Internet Browser

' Elements
Business_Object(businessObject, "A Business Object")
Business_Process(someBusinessProcess,"Some Business Process")
Business_Service(itSupportService, "IT Support for Business (Application Service)")

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

Technology_Artifact(inMemoryItem,"in memory / 'on the fly' html/javascript")
Technology_Service(internetBrowser, "Internet Browser Generic & Plugin")
Technology_Service(internetBrowserPlugin, "Some Internet Browser Plugin")
Technology_Service(webServer, "Some web server")

'Relationships
Rel_Flow_Left(someBusinessProcess, businessObject, "")
Rel_Serving_Up(itSupportService, someBusinessProcess, "")
Rel_Specialization_Up(webpageBehaviour, itSupportService, "")
Rel_Flow_Right(dataObject, webpageBehaviour, "")
Rel_Specialization_Up(dataObject, businessObject, "")
Rel_Assignment_Left(ActivePartWebPage, webpageBehaviour, "")
Rel_Specialization_Up(inMemoryItem, dataObject, "")
Rel_Realization_Up(inMemoryItem, ActivePartWebPage, "")
Rel_Specialization_Right(inMemoryItem,internetBrowser, "")
Rel_Serving_Up(internetBrowser, webpageBehaviour, "")
Rel_Serving_Up(internetBrowserPlugin, webpageBehaviour, "")
Rel_Aggregation_Right(internetBrowser, internetBrowserPlugin, "")
Rel_Access_Up(webServer, inMemoryItem, "")
```

```
Rel_Serving_Up(webServer, internetBrowser, "")
@enduml
```



21.8 Miscellaneous

You can list standard library folders using the special diagram:

```
@startuml
stdlib
@enduml
```

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

awslib
Version 3.0.0
Delivered by https://github.com/awslabs/aws-icons-for-plantuml

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

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

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

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

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

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

osa
Version 0.0.1
Delivered by https://github.com/Crashedmind/PlantUML-opensecurityarchitecture-icons

tupadr3
Version 2.0.0
Delivered by https://github.com/tupadr3/plantuml-icon-font-sprites

It is also possible to use the command line `java -jar plantuml.jar -stdlib` to display the same list.

Finally, you can extract the full standard library sources using `java -jar plantuml.jar -extractstdlib`. All files will be extracted in the folder `stdlib`.

Sources used to build official PlantUML releases are hosted here https://github.com/plantuml/plantuml-stdlib.You can create Pull Request to update or add some library if you find it relevant.

Contents

1 Diagramme de séquence **1**
1.1 Exemples de base . . . 1
1.2 Déclaration de participants . . . 1
1.3 Caractères non alphanumérique dans les participants . . . 3
1.4 Message à soi-même . . . 3
1.5 Autre style de flèches . . . 3
1.6 Changer la couleur des flèches . . . 4
1.7 Numérotation automatique des messages . . . 4
1.8 Page Title, Header and Footer . . . 6
1.9 Découper un diagramme . . . 7
1.10 Regrouper les messages (cadres UML) . . . 8
1.11 Note sur les messages . . . 9
1.12 Encore plus de notes . . . 10
1.13 Changer l'aspect des notes . . . 10
1.14 Créole (langage de balisage léger) et HTML . . . 11
1.15 Séparation . . . 12
1.16 Référence . . . 13
1.17 Retard . . . 13
1.18 Séparation verticale . . . 14
1.19 Lignes de vie . . . 15
1.20 Return . . . 16
1.21 Création de participants. . . . 16
1.22 Messages entrant et sortant . . . 17
1.23 Stéréotypes et décoration . . . 18
1.24 Plus d'information sur les titres . . . 19
1.25 Cadre pour les participants . . . 20
1.26 Supprimer les en-pieds . . . 21
1.27 Personnalisation . . . 21
1.28 Changer le padding . . . 23

2 Diagramme de cas d'utilisation **25**
2.1 Cas d'utilisation . . . 25
2.2 Acteurs . . . 25
2.3 Description des cas d'utilisation . . . 26
2.4 Exemples très simples . . . 26
2.5 Héritage . . . 27
2.6 Notes . . . 27
2.7 Stéréotypes . . . 28
2.8 Changer les directions des flèches . . . 29
2.9 Découper les diagrames . . . 30
2.10 De droite à gauche . . . 30
2.11 La commande Skinparam . . . 31
2.12 Exemple complet . . . 32

3 Diagramme de classes **33**
3.1 Relations entre classes . . . 33
3.2 Libellés sur les relations . . . 34
3.3 Définir les méthodes . . . 35
3.4 Définir les visibilités . . . 36
3.5 Abstrait et statique . . . 36
3.6 Corps de classe avancé . . . 37
3.7 Notes et stéréotypes . . . 38
3.8 Encore des notes . . . 38
3.9 Note sur les liens . . . 39
3.10 Classe abstraite et Interface . . . 40
3.11 Caractères non alphabétiques . . . 41
3.12 Masquer les attributs et les méthodes . . . 41

3.13 Cacher des classes . . . 42
3.14 Utilisation de la généricité . . . 43
3.15 Caractère spécial . . . 43
3.16 Packages . . . 43
3.17 Modèle de paquet . . . 44
3.18 Les espaces de nommage . . . 45
3.19 Creation automatique d'espace de nommage . . . 46
3.20 Interface boucle . . . 47
3.21 Changer la direction . . . 47
3.22 Classes d'association . . . 48
3.23 Personnalisation . . . 49
3.24 Stéréotypes Personnalisés . . . 50
3.25 Dégradé de couleur . . . 50
3.26 Aide pour la mise en page . . . 51
3.27 Découper les grands diagrammes . . . 52

4 Diagrammes d'activité 54
4.1 Exemple de base . . . 54
4.2 Texte sur les flèches. . . . 54
4.3 Changer la direction des flèches . . . 54
4.4 Branches . . . 55
4.5 Encore des branches . . . 56
4.6 Synchronisation . . . 57
4.7 Description détaillée . . . 58
4.8 Notes . . . 58
4.9 Partition . . . 59
4.10 Paramètre de thème . . . 60
4.11 Octogone . . . 61
4.12 Exemple complet . . . 61

5 Diagrammes d'activité (béta) 64
5.1 Activité simple . . . 64
5.2 Départ/Arrêt . . . 64
5.3 Conditionnel . . . 65
5.4 Boucle de répétition . . . 66
5.5 Boucle While . . . 67
5.6 Processus parallèle . . . 67
5.7 Notes . . . 68
5.8 Couleurs . . . 69
5.9 Flèches . . . 69
5.10 Connector . . . 70
5.11 Groupement . . . 70
5.12 Couloirs . . . 71
5.13 Détacher . . . 72
5.14 SDL . . . 73
5.15 Exemple complet . . . 74

6 Diagrammes de composants 76
6.1 Composants . . . 76
6.2 Interfaces . . . 76
6.3 Exemple simple . . . 77
6.4 Mettre des notes . . . 77
6.5 Regrouper des composants . . . 77
6.6 Changer la direction des flèches . . . 79
6.7 Utiliser la notation UML2 . . . 80
6.8 Description longue . . . 81
6.9 Couleurs individuelles . . . 81
6.10 Sprites et stéréotypes . . . 81
6.11 Skinparam . . . 82

7 Diagrammes d'état **84**
7.1 Exemple simple . . . 84
7.2 Change state rendering . . . 84
7.3 Etat composite . . . 85
7.4 Nom long . . . 86
7.5 Etat concurrent . . . 87
7.6 Direction des flèches . . . 88
7.7 Note . . . 89
7.8 Plus de notes . . . 90
7.9 Skinparam . . . 90

8 Diagrammes d'objets **92**
8.1 Définition des objets . . . 92
8.2 Relations entre les objets . . . 92
8.3 Ajout de champs . . . 92
8.4 Caractéristiques communes avec les diagrammes de classes . . . 93

9 Diagramme de temps **94**
9.1 Définitions des participants . . . 94
9.2 Ajout de messages . . . 94
9.3 Référence relative de temps . . . 95
9.4 Définition participant par participant . . . 96
9.5 Choix du zoom . . . 96
9.6 État initial . . . 97
9.7 Intricated state . . . 97
9.8 Hidden state . . . 98
9.9 Ajout de contraintes . . . 98
9.10 Ajout de textes . . . 99

10 Diagramme de Gantt **101**
10.1 Définir des tâches . . . 101
10.2 Ajout de contraintes . . . 101
10.3 Noms courts . . . 101
10.4 Choix des couleurs . . . 102
10.5 Jalon . . . 102
10.6 Calendrier . . . 102
10.7 Close day . . . 102
10.8 Simplified task succession . . . 103
10.9 Separator . . . 103
10.10 Working with resources . . . 103
10.11 Exemple plus complexe . . . 104

11 MindMap **105**
11.1 OrgMode syntax . . . 105
11.2 Removing box . . . 105
11.3 Arithmetic notation . . . 106
11.4 Markdown syntax . . . 106
11.5 Changing diagram direction . . . 107
11.6 Complete example . . . 107

12 Work Breakdown Structure **109**
12.1 OrgMode syntax . . . 109
12.2 Change direction . . . 109
12.3 Arithmetic notation . . . 110

13 Mathématiques **112**
13.1 Diagramme indépendant . . . 112
13.2 Comment cela fonctionne ? . . . 113

14 Common commands **114**
14.1 Comments . . . 114
14.2 Footer and header . . . 114
14.3 Zoom . . . 114
14.4 Title . . . 115
14.5 Caption . . . 116
14.6 Legend the diagram . . . 116

15 Salt **118**
15.1 widgets de base . . . 118
15.2 Utilisation de grille . . . 118
15.3 Group box . . . 119
15.4 Utilisation des séparateurs . . . 119
15.5 Widget d'arbre . . . 120
15.6 Accolades délimitantes . . . 120
15.7 Ajout d'onglet . . . 121
15.8 Utiliser les menus . . . 121
15.9 Tableaux avancés . . . 122
15.10 OpenIconic . . . 123
15.11 Include Salt . . . 123
15.12 Scroll Bars . . . 126

16 Créole **128**
16.1 Formatage de texte . . . 128
16.2 Listes . . . 128
16.3 Caractère d'échappement . . . 129
16.4 Lignes horizontales . . . 129
16.5 Entêtes . . . 130
16.6 Tag HTML . . . 130
16.7 Tableau . . . 131
16.8 Tree . . . 132
16.9 Special characters . . . 132
16.10 OpenIconic . . . 132

17 Defining and using sprites **134**
17.1 Encoding Sprite . . . 135
17.2 Importing Sprite . . . 135
17.3 Examples . . . 135

18 Skinparam command **137**
18.1 Usage . . . 137
18.2 Nested . . . 137
18.3 Black and White . . . 137
18.4 Shadowing . . . 138
18.5 Reverse colors . . . 139
18.6 Colors . . . 139
18.7 Font color, name and size . . . 140
18.8 Text Alignment . . . 140
18.9 Examples . . . 141
18.10 List of all skinparam parameters . . . 144

19 Preprocessing **147**
19.1 Migration notes . . . 147
19.2 Variable definition . . . 147
19.3 Conditions . . . 148
19.4 Void function . . . 148
19.5 Return function . . . 149
19.6 Default argument value . . . 150
19.7 Unquoted function . . . 151

19.8 Including files or URL . . . 151
19.9 Including Subpart . . . 152
19.10 Builtin functions . . . 153
19.11 Logging . . . 153
19.12 Memory dump . . . 154
19.13 Assertion . . . 154
19.14 Building custom library . . . 155
19.15 Search path . . . 155
19.16 Argument concatenation . . . 155
19.17 Dynamic function invocation . . . 156

20 Unicode **157**
20.1 Examples . . . 157
20.2 Charset . . . 159

21 Standard Library **160**
21.1 AWS library . . . 160
21.2 Azure library . . . 160
21.3 Cloud Insight . . . 161
21.4 Devicons and Font Awesome library . . . 162
21.5 Google Material Icons . . . 163
21.6 Office . . . 164
21.7 ArchiMate . . . 166
21.8 Miscellaneous . . . 167