PlantUML を使った UML の描き方

PlantUML 言語リファレンスガイド

(Version 1.2021.1)

PlantUML は、以下のようなダイアグラムを素早く作成するためのコンポーネントです。

- シーケンス図
- ユースケース図
- クラス図
- オブジェクト図
- アクティビティ図
- コンポーネント図
- 配置図
- 状態遷移図（ステートマシン図）
- タイミング図

以下のような、UML 以外の図もサポートしてます。

- JSON Data
- Network diagram (nwdiag)
- ワイヤーフレーム
- アーキテクチャ図
- 仕様及び記述言語 (SDL)
- Ditaa
- ガントチャート
- マインドマップ
- WBS 図 (作業分解図)
- AsciiMath や JLaTeXMath による、数学的記法
- ER 図

各ダイアグラムは、シンプルで直感的に書くことができます。

1 シーケンス図

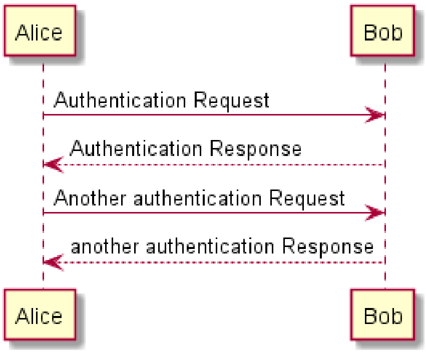
1.1 基本的な例

シーケンス -> を、2 つの分類子間のメッセージを描画するために使います。分類子を、明示的に宣言する必要はありません。

点線の矢印を使う場合は、--> とします。

また、<- や <-- を使うこともできます。これらによって図の見た目が変わることはありませんが、可読性を高めることができます。ただし、以上の方法はシーケンス図だけに当てはまります。ほかの種類の図には当てはまりません。

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

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



1.2 分類子の宣言

キーワード participant を使って分類子を宣言すると、分類子の表示を調整することができます。

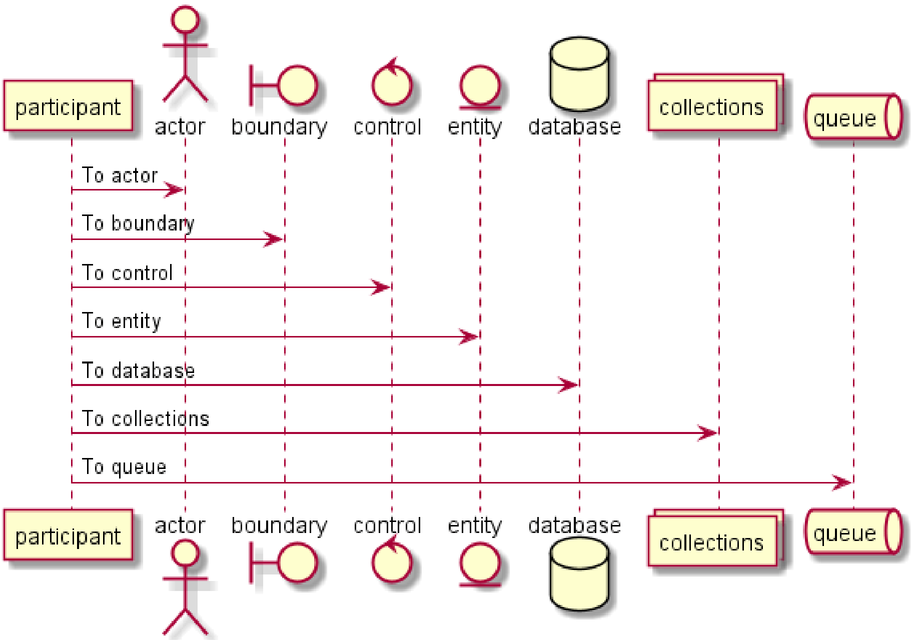
宣言した順序が、デフォルトの表示順になります。

分類子の宣言に別のキーワードを使用すると、分類子の形を変えることができます：

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

```
@startuml
participant participant as Foo
actor       actor       as Foo1
boundary    boundary    as Foo2
control     control     as Foo3
entity      entity      as Foo4
database    database    as Foo5
collections collections as Foo6
queue       queue       as Foo7
Foo -> Foo1 : To actor
```

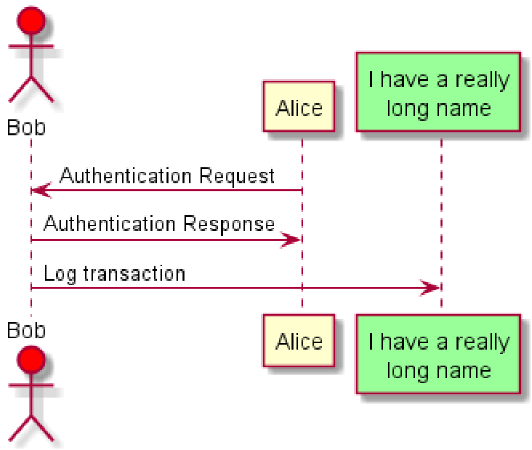
```
Foo -> Foo2 : To boundary
Foo -> Foo3 : To control
Foo -> Foo4 : To entity
Foo -> Foo5 : To database
Foo -> Foo6 : To collections
Foo -> Foo7: To queue
@enduml
```



キーワード as を使って分類子の名前を変更することができます。

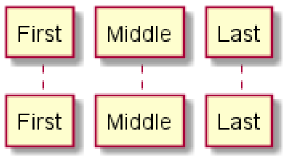
アクターや分類子の背景色を、HTML コードや色名を使って変更することもできます。

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

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



order キーワードを使って、分類子が表示される順序を変更することもできます。

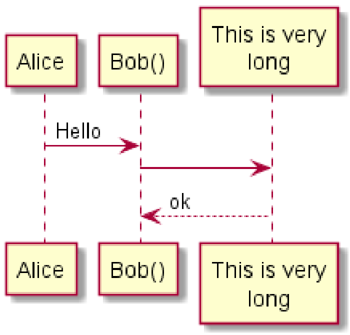
```
@startuml
participant Last order 30
participant Middle order 20
participant First order 10
@enduml
```



1.3 分類子名にアルファベット以外を使う

分類子を定義するときに引用符を使用することができます。そして、分類子にエイリアスを与えるためにキーワード as を使用することができます。

```
@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
```



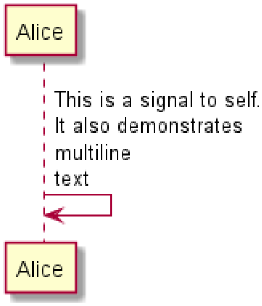
1.4 自分自身へのメッセージ

分類子は自分自身へメッセージを送信できます。

を使用して、複数行のテキストを扱えます。

```
@startuml
```

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

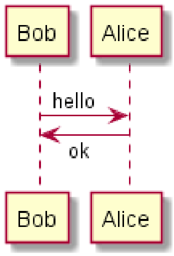


1.5 Text alignment

1.5.1 応答メッセージの矢印の下の文字

`skinparam responseMessageBelowArrow true` コマンドを使うことで、応答メッセージの矢印の下に文字を配置することができます。

```
@startuml
skinparam responseMessageBelowArrow true
Bob -> Alice : hello
Alice -> Bob : ok
@enduml
```



TODO: TODO Link to Text Alignment on skinparam page.

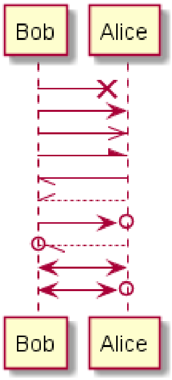
1.6 矢印の見た目を変える

矢印の見た目をいくつかの方法によって変更できます。

- メッセージの消失を示す最後の `x` を追加
- `\` や `/` を `<` や `>` の代わりに使うと
- 矢印の先端が上側だけまたは下側だけになります。
- 矢印の先端を繰り返す (たとえば `>>` や `//`) と、矢印の先端が細くなります。
- `--` を `-` の代わりに使うと、矢印が点線になります。
- 矢じりに最後の"O" を追加
- 双方向の矢印を使用する

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

Bob ->o Alice
```

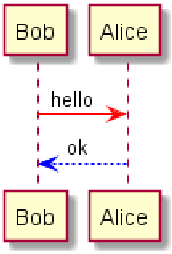
```
Bob o\\-- Alice

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



1.7 矢印の色を替える

以下の表記を使って、個々の矢印の色を変えることができます。

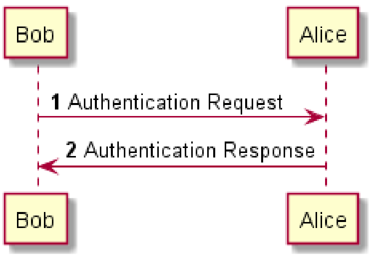
```
@startuml
Bob -[#red]> Alice : hello
Alice -[#0000FF]->Bob : ok
@enduml
```



1.8 メッセージシーケンスの番号付け

メッセージへ自動で番号を振るために、キーワード autonumber を使います。

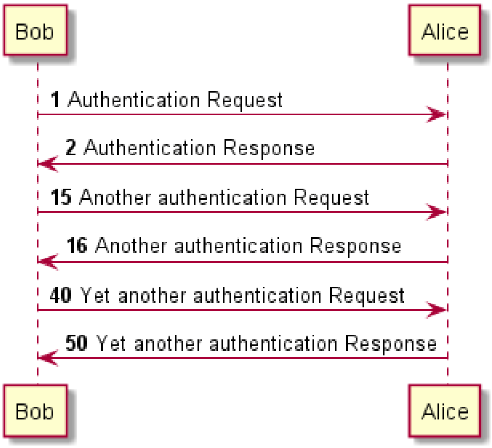
```
@startuml
autonumber
Bob -> Alice : Authentication Request
Bob <- Alice : Authentication Response
@enduml
```



autonumber //開始// で開始番号を、また、autonumber //開始// //増分// で増分も指定することができます。

```
@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
```

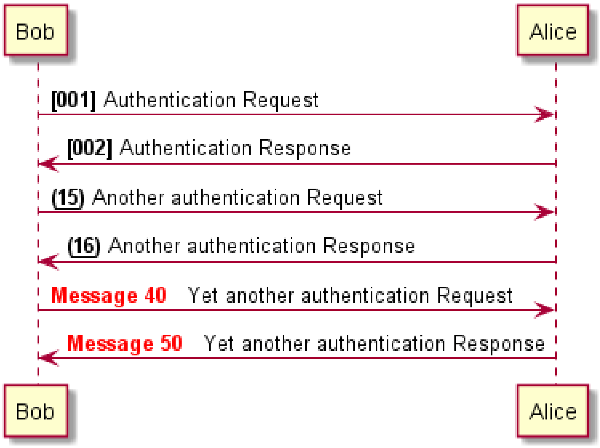


二重引用符で囲って番号の書式を指定することができます。

その書式指定は Java の `DecimalFormat` 方式で行う (`0` は桁を表し, `#` は存在しない場合は 0 で埋める桁を意味する)。

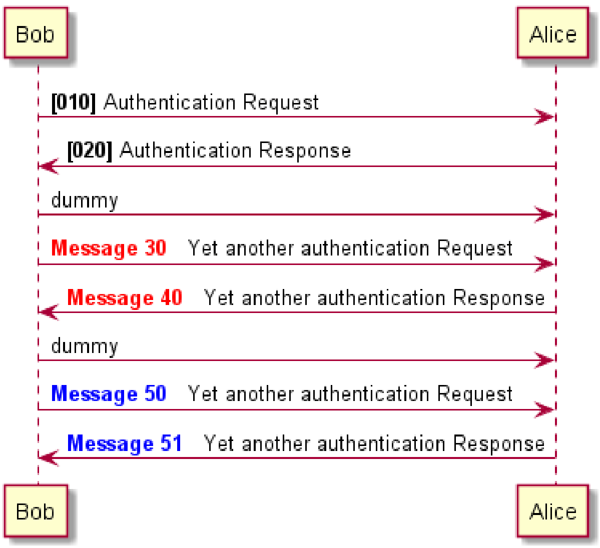
HTML タグを書式に使うこともできます。

```
@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
```



autonumber stop と autonumber resume //増分// //書式// を自動採番の一時停止と再開にそれぞれを使用することができます。

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

autonumber stop
Bob -> Alice : dummy

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

autonumber stop
Bob -> Alice : dummy

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

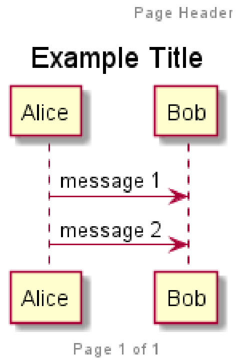


1.9 タイトル、ヘッダー、フッター

title キーワードはページにタイトルをつけるのに使われます。

header や footer を使うことにより、ページにヘッダーやフッターをつけて表示することができます。

```
@startuml

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

title Example Title

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

@enduml
```



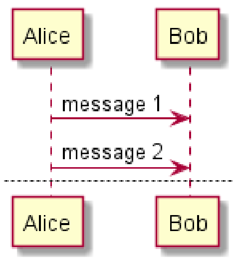
1.10　図の分割

図を複数の画像に分けるためにキーワード newpage を使います。

新しいページのタイトルをキーワード newpage の直後に書くことができます。

これは、複数ページにわたる長い図を書くときに便利な機能です。

```
@startuml

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

newpage

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

newpage A title for the\nlast page

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



1.11 メッセージのグループ化

次のキーワードを使えば、メッセージをまとめてグループ化できます。

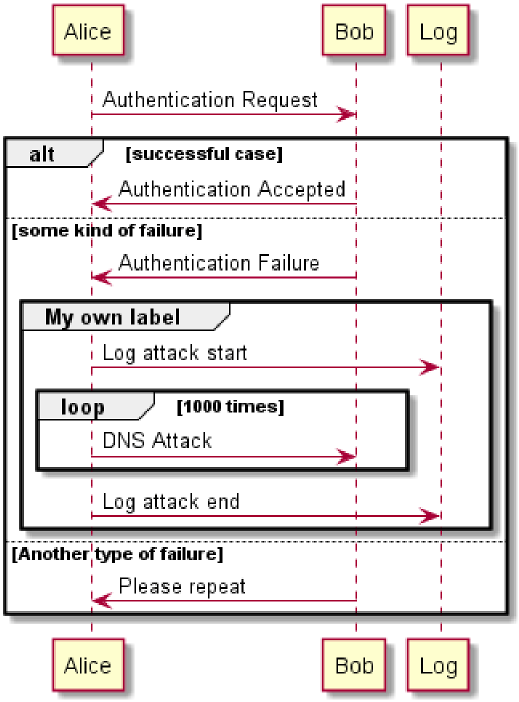
- alt/else
- opt
- loop
- par
- break
- critical
- group 表示するテキスト

ヘッダ部分に文字列を追加することが可能です。(group については、後述の「group の 2 つ目のラベル」を参照)

グループを閉じるにはキーワード end を使用します。

注：グループはネスト可能です。

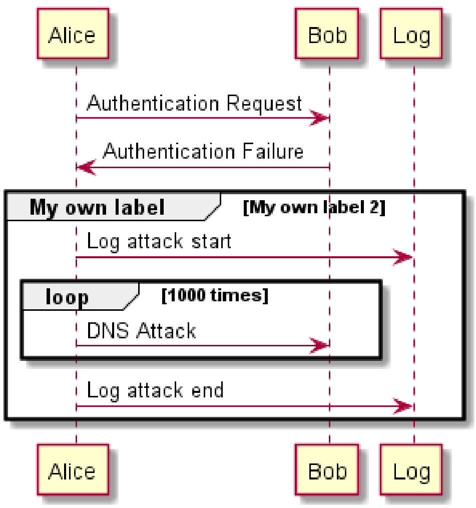
```
@startuml
Alice -> Bob: Authentication Request

alt successful case

    Bob -> Alice: Authentication Accepted

else some kind of failure

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

else Another type of failure

   Bob -> Alice: Please repeat

end
@enduml
```



1.12 group の 2 つ目のラベル

group では、[と] の間に 2 つ目のラベルを設定し、ヘッダに表示させることができます。

```
@startuml
Alice -> Bob: Authentication Request
Bob -> Alice: Authentication Failure
group My own label [My own label 2]
    Alice -> Log : Log attack start
    loop 1000 times
        Alice -> Bob: DNS Attack
    end
    Alice -> Log : Log attack end
end
@enduml
```



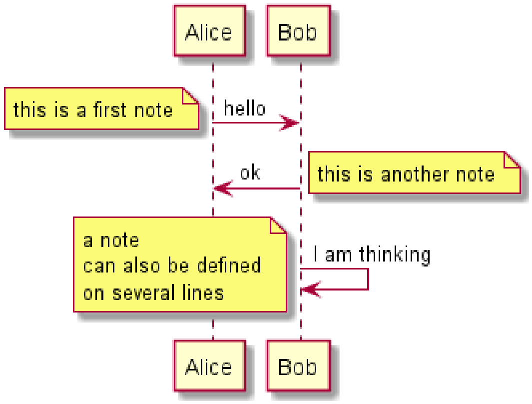
[Ref. QA-2503]

1.13 メッセージの注釈

メッセージのすぐ後ろにキーワード `note left` または `note right` を使用しメッセージの注釈をつけることが可能です。

`end note` キーワードを使って、複数行の注釈を付けることができます。

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

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

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



1.14 その他の注釈

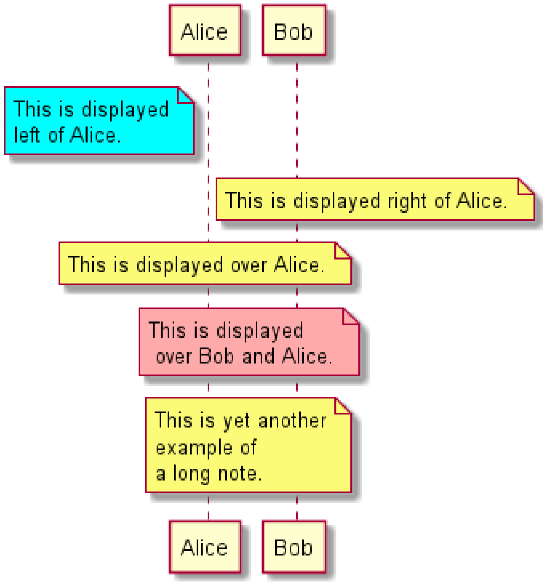
分類子との相対位置を指定して注釈を付けるには、次のものを使います:

注釈を目立たせるために、背景色を変えることができます。

また、キーワード `end note` を使って複数行の注釈を付けることができます。

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

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

note over Alice: This is displayed over Alice.

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

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

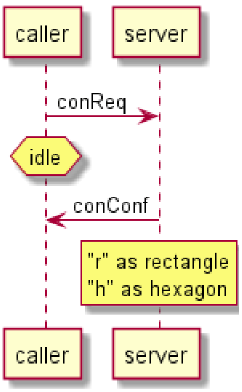
```
end note
@enduml
```



1.15 ノートの形を変える

キーワード hnote と rnote を使ってノートの形を変更できます。

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



[Ref. [QA-1765](https://forum.plantuml.net/1765/is-it-possible-to-have-different-shapes-for-notes?show=1806#c1806)]

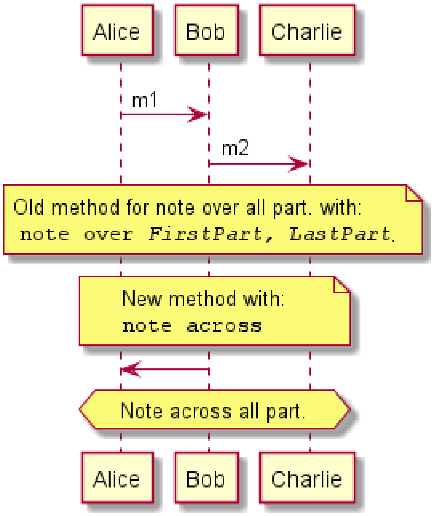
1.16 Note over all participants [across]

You can directly make a note over all participants, with the syntax:

- `note across: note_description`

```
@startuml
Alice->Bob:m1
Bob->Charlie:m2
```

```
note over Alice, Charlie: Old method for note over all part. with:\n ""note over //FirstPart, LastPa
note across: New method with:\n""note across""
Bob->Alice
hnote across:Note across all part.
@enduml
```



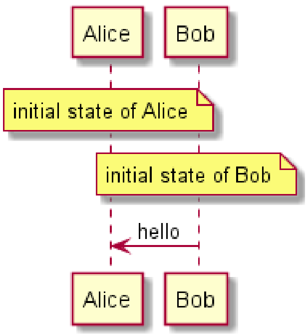
[Ref. QA-9738]

1.17 Several notes aligned at the same level [/]

You can make several notes aligned at the same level, with the syntax `/`:

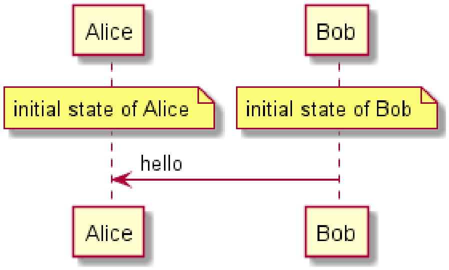
- without `/` *(by default, the notes are not aligned)*

```
@startuml
note over Alice : initial state of Alice
note over Bob : initial state of Bob
Bob -> Alice : hello
@enduml
```



- with `/` *(the notes are aligned)*

```
@startuml
note over Alice : initial state of Alice
/ note over Bob : initial state of Bob
Bob -> Alice : hello
@enduml
```

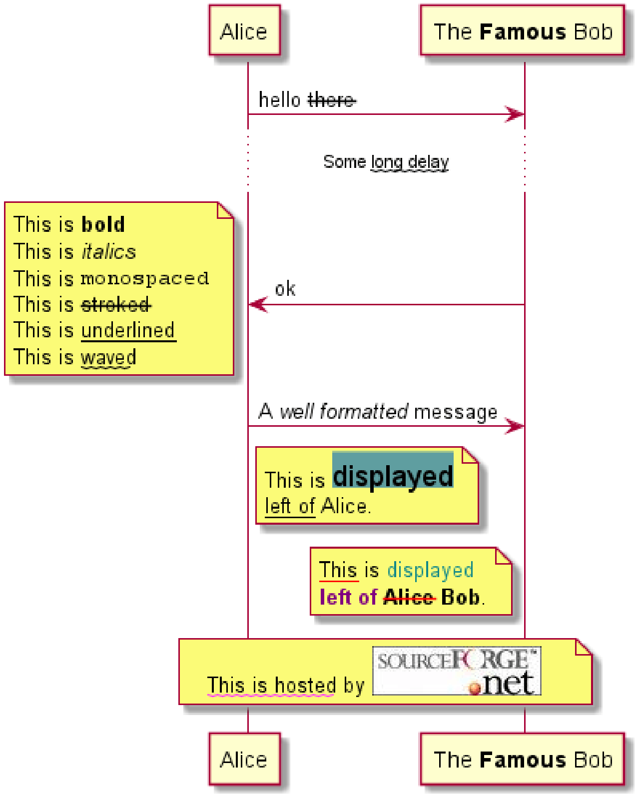


[Ref. QA-354]

1.18 Creole と HTML

PlantUML では creole フォーマットを使うこともできます。

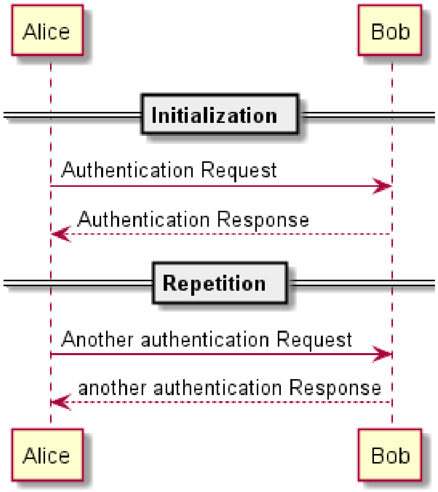
```
@startuml
participant Alice
participant "The **Famous** Bob" as Bob

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
```



1.19 境界線(区切り線)

== を使って、図を論理的なステップに分けることも出来ます。

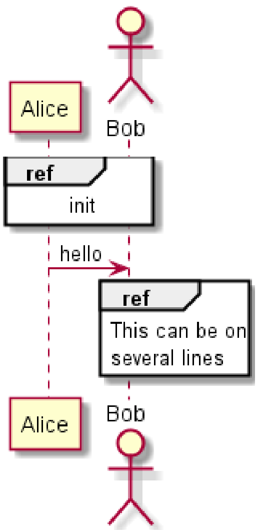
```
@startuml

== Initialization ==

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

== Repetition ==

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

@enduml
```



1.20 リファレンス

キーワード `ref over` を使用して、図中にリファレンスを挿入できます。

```
@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
```

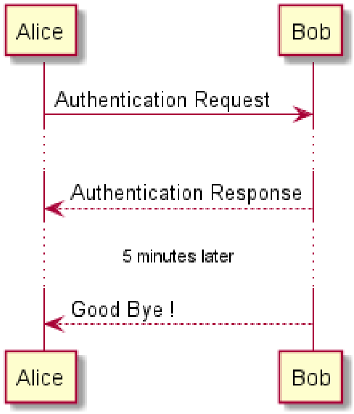


1.21 遅延

処理の遅延を表すために `...` が使えます。また、作成した遅延にコメントを付けることもできます。

```
@startuml

Alice -> Bob: Authentication Request
```

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

@enduml
```

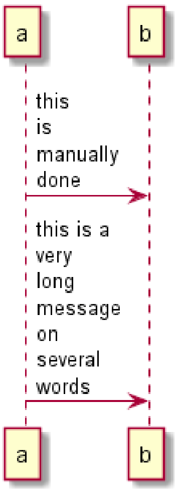


1.22 テキストの折り返し

を使って改行することで、長いメッセージを折り返すことができます。

また、maxMessageSize を設定するという方法もあります。

```
@startuml
skinparam maxMessageSize 50
participant a
participant b
a -> b :this\nis\nmanually\ndone
a -> b :this is a very long message on several words
@enduml
```



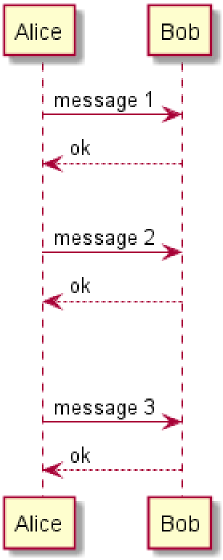
1.23 間隔

図の間隔を調整するために、記号 ||| を使用することができます。

さらにピクセル数を指定することもできます。

```
@startuml

Alice -> Bob: message 1
```

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

@enduml
```



1.24 ライフラインの活性化と破棄

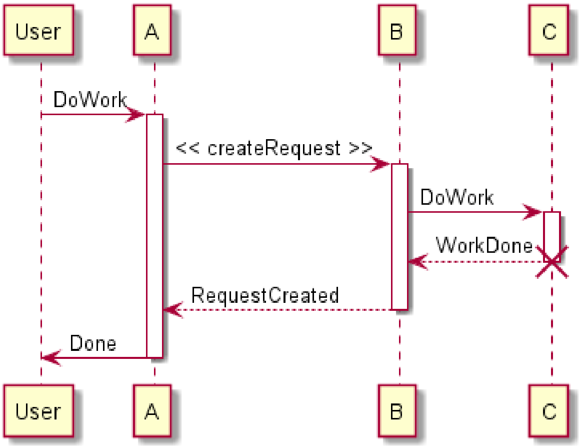
activate と deactivate を使って分類子の活性化を表します。

分類子の活性化はライフラインで表されます。

activate と deactivate は直前のメッセージに適用されます。

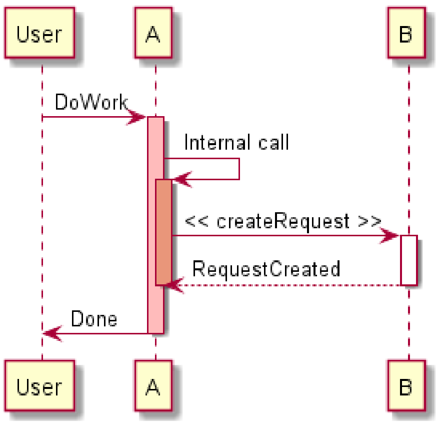
destroy は分類子のライフラインが終わったことを表します。

```
@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
```



ライフラインはネスト (入れ子に) することができ、色をつけることもできます。

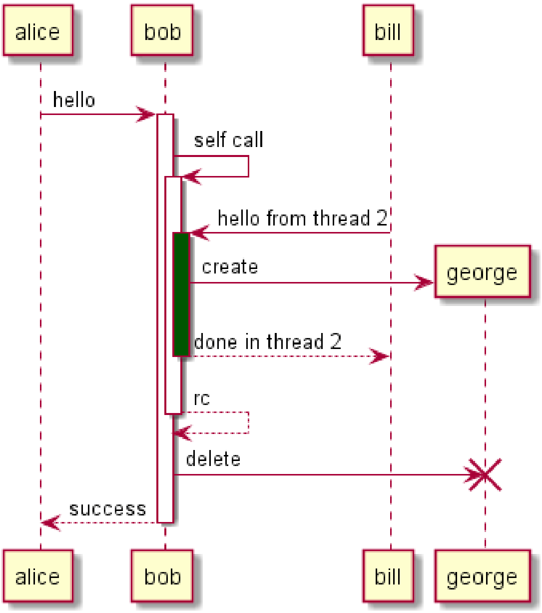
```
@startuml
participant User

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
```



自動的に活性化 (autoactivate) することもできます。この場合は return キーワードを使用します。

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

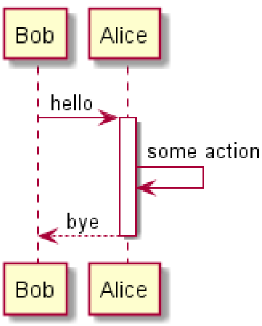
```
return rc
bob -> george !! : delete
return success

@enduml
```



1.25 Return

新しいコマンド `return` は、リターンメッセージを生成し、オプションでテキストラベルをつけることができます。リターンする先は最も最近活性化したライフラインです。構文は単純に return ラベルです。ラベルを与える場合には、通常のメッセージに与えることが可能な文字列を何でも与えることができます。

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

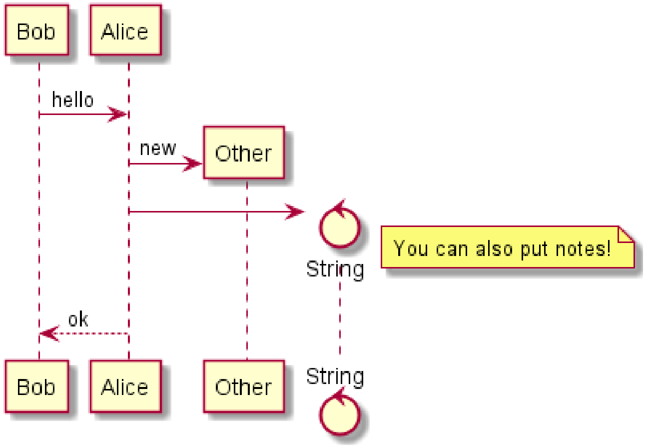


1.26 分類子の生成

キーワード `create` を、オブジェクトが最初のメッセージを受信する直前に置くことにより、このメッセージがオブジェクトを新しく 生成していることを強調して表現できます。

```
@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
```

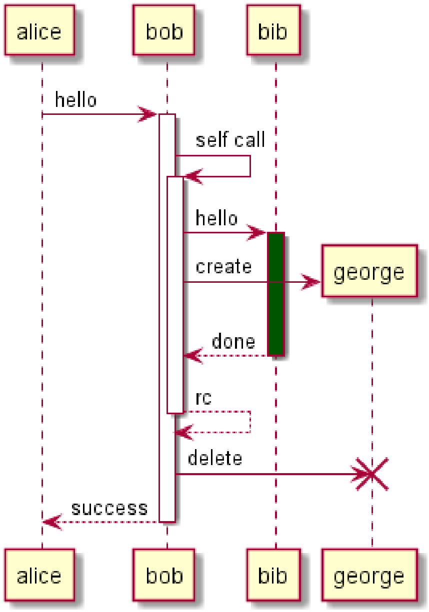


1.27 活性化、非活性化、生成のショートカット

対象の分類子を記述した直後に、次の記法を使うことができます。

- `++` 対象を活性化する (続けて #color のように色を記述することもできます)
- `--` 起点側を非活性化する
- `**` 対象のインスタンスを生成する
- `!!` 対象のインスタンスを破棄する

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

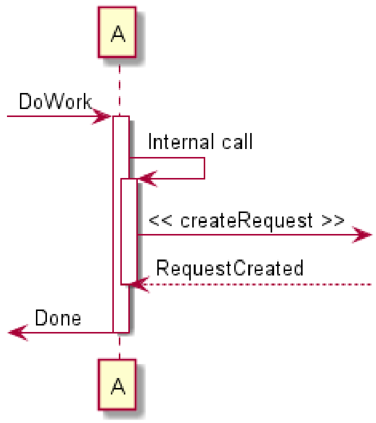


1.28 インとアウトのメッセージ

図の一部だけにフォーカスを当てたい場合には、「外から入ってくる」または「外に出ていく」メッセージを使えます。

左角括弧"[" を使って図の左端、右角括弧"]" を使って図の右側を表せます。

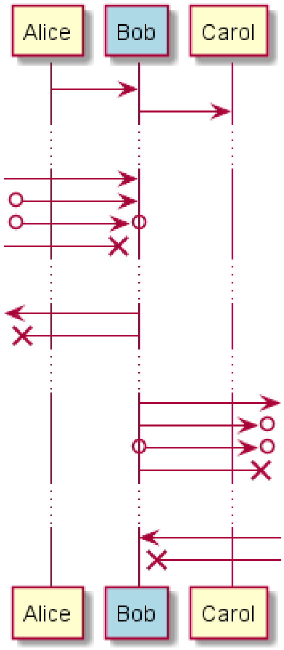
```
@startuml
[-> A: DoWork

activate A

A -> A: Internal call
activate A

A ->] : << createRequest >>

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



また、次の書き方も使えます：

```
@startuml
```

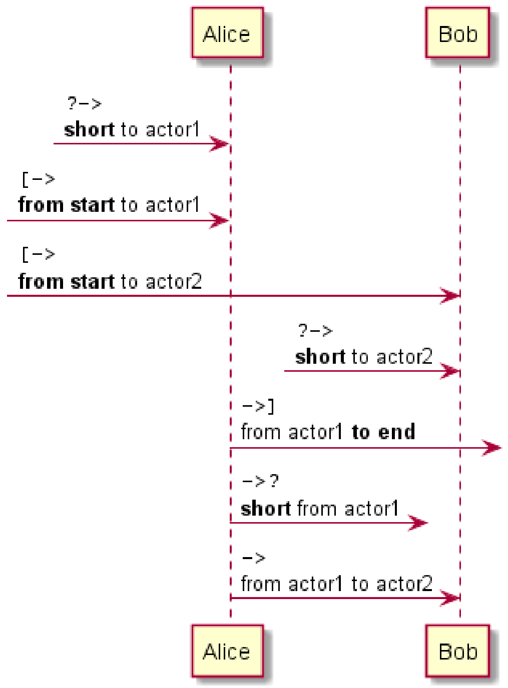
```
participant Alice
participant Bob #lightblue
Alice -> Bob
Bob -> Carol
...
[-> Bob
[o-> Bob
[o->o Bob
[x-> Bob
...
[<- Bob
[x<- Bob
...
Bob ->]
Bob ->o]
Bob o->o]
Bob ->x]
...
Bob <-]
Bob x<-]

@enduml
```



1.29 インとアウトのメッセージに短い矢印を使う

? で短い矢印を使用することができます。

```
@startuml
?-> Alice     : ""?->""\n**short** to actor1
[-> Alice     : ""[->""\n**from start** to actor1
[-> Bob       : ""[->""\n**from start** to actor2
?-> Bob       : ""?->""\n**short** to actor2
Alice ->]     : ""->]""\nfrom actor1 **to end**
Alice ->?     : ""->?""\n**short** from actor1
Alice -> Bob : ""->"" \nfrom actor1 to actor2
@enduml
```

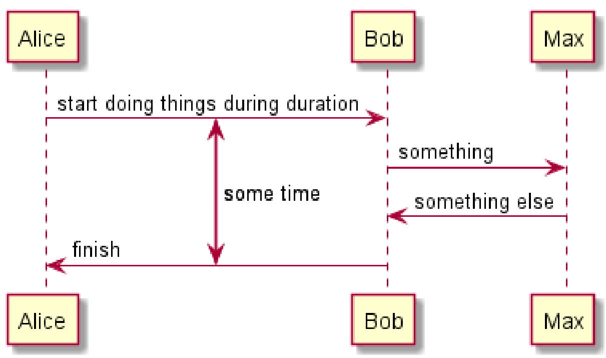


[Ref. QA-310]

1.30 アンカーと持続時間

teoz を使用するとダイアグラムにアンカーを追加することができ、それによって持続時間を表現することができます。

```
@startuml
!pragma teoz true

{start} Alice -> Bob : start doing things during duration
Bob -> Max : something
Max -> Bob : something else
{end} Bob -> Alice : finish

{start} <-> {end} : some time

@enduml
```

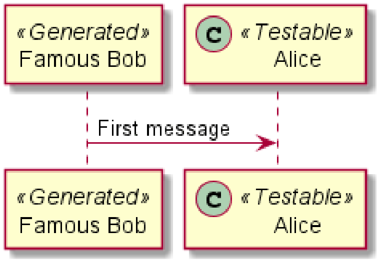


1.31 ステレオタイプとスポット

<< と >> を使い分類子にステレオタイプをつけることができます。

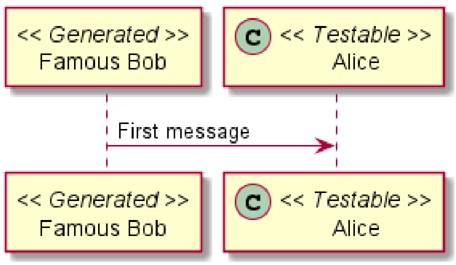
(X,color) と記述することによりステレオタイプに色付きの文字と円のアイコンをつけることができます。

```
@startuml

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

Bob->Alice: First message

@enduml
```

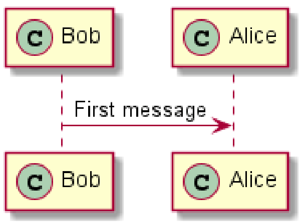


デフォルトでは *guillemet* キャラクターがステレオタイプを表示するために使用されます。スキンパラメータ `guillemet` を使用してこの動作を変更することができます：

```
@startuml

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

Bob->Alice: First message

@enduml
```



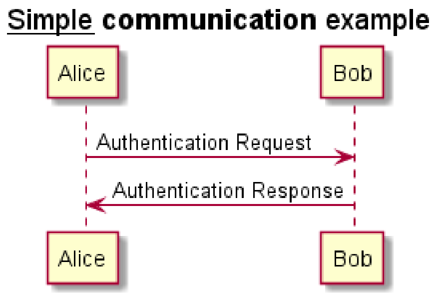

```
@startuml

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

Bob->Alice: First message

@enduml
```



1.32 タイトルについての詳細

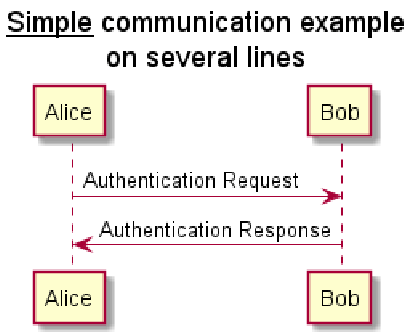
タイトルには creole フォーマットが使用できます。

```
@startuml

title __Simple__ **communication** example

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

@enduml
```



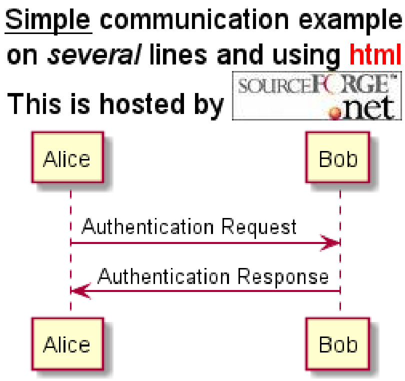
タイトルの記述では を使用して新しい行を追加することができます。

```
@startuml

title __Simple__ communication example\non several lines

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

@enduml
```



また、キーワード `title` と `end title` を使うことにより、タイトルを複数行にわたって記述できます。

```
@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
```

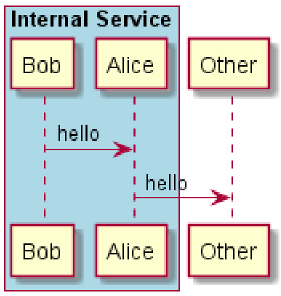


1.33 分類子の囲み

キーワード `box` と `end box` を使い、分類子のまわりにボックスを描くことができます。

タイトルや背景色をキーワード `box` に続けて任意で追加できます。

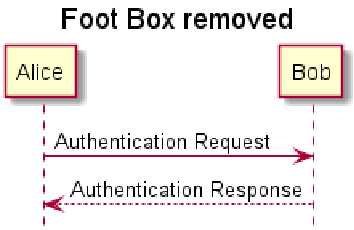
```
@startuml

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

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

@enduml
```



1.34 フッターの除去

図からフッターを削除するにはキーワード `hide footbox` を使います。

```
@startuml

hide footbox
title Foot Box removed

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

@enduml
```



1.35 スキンパラメータ

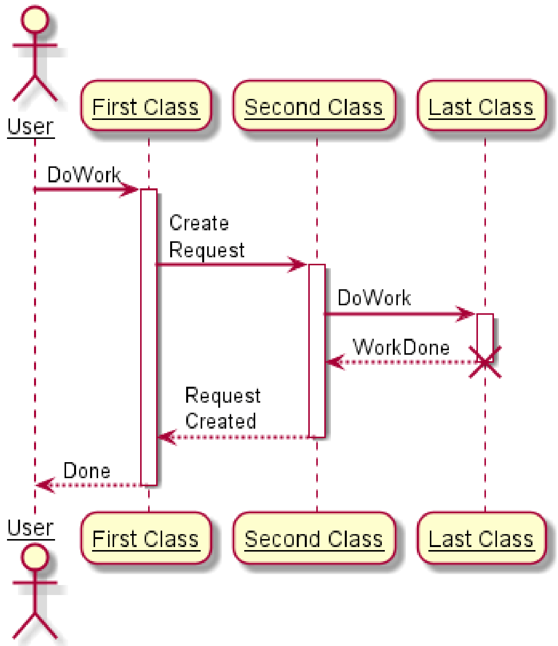
ダイアグラムの色やフォントを変更するには skinparam コマンドを使用します。

このコマンドは以下の場面で使用できます。

- ダイアグラム定義内で他のコマンドを同様に。
- インクルードされたファイル内。
- 設定ファイルのコマンドライン内や ANT タスク内。

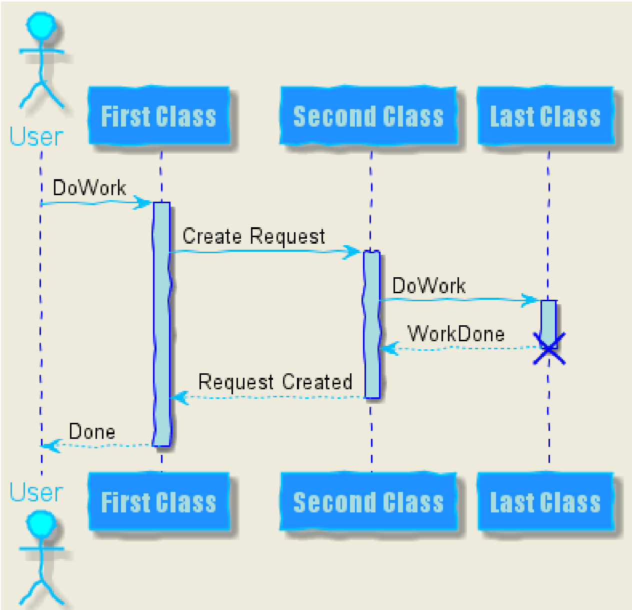
次の例のように他のパラメータを変えることもできます。

```
@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
```

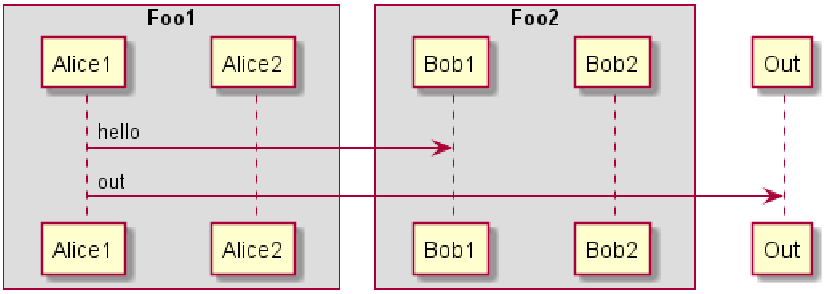
```
B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml
```



1.36 パディングの変更

パディングの設定を変更することができます。

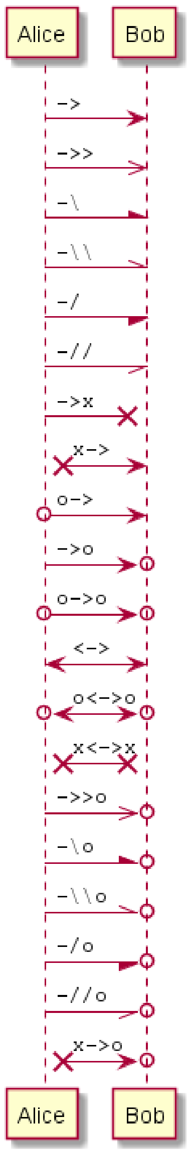
```
@startuml
skinparam ParticipantPadding 20
skinparam BoxPadding 10

box "Foo1"
participant Alice1
participant Alice2
end box
box "Foo2"
participant Bob1
participant Bob2
end box
Alice1 -> Bob1 : hello
Alice1 -> Out : out
@enduml
```



1.37 付録：全種類の矢印の例

1.37.1 通常の矢印

```
@startuml
participant Alice as a
participant Bob   as b
a ->     b : ""->   ""
a ->>    b : ""->>  ""
a -\     b : ""-\   ""
a -\\    b : ""-\\\\""
a -/     b : ""-/   ""
a -//    b : ""-//  ""
a ->x    b : ""->x  ""
a x->    b : ""x->  ""
a o->    b : ""o->  ""
a ->o    b : ""->o  ""
a o->o   b : ""o->o ""
a <->    b : ""<->  ""
a o<->o  b : ""o<->o""
a x<->x  b : ""x<->x""
a ->>o   b : ""->>o ""
a -\o    b : ""-\o  ""
a -\\o   b : ""-\\\\o""
a -/o    b : ""-/o  ""
a -//o   b : ""-//o ""
a x->o   b : ""x->o ""
@enduml
```

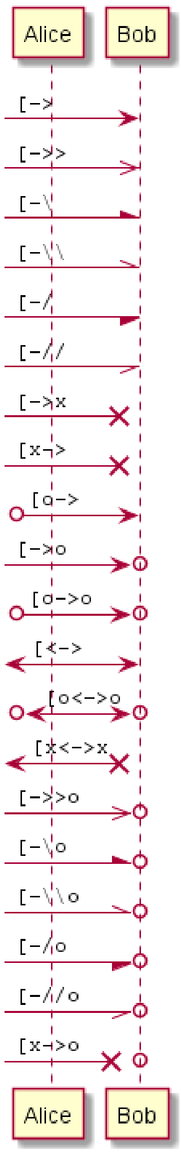


1.37.2 インとアウトのメッセージ（'['、']' を使用）

1.37.3 イン（'[' を使用）

```
@startuml
participant Alice as a
participant Bob   as b
[->       b : ""[->    ""
[->>      b : ""[->>   ""
[-\       b : ""[-\    ""
[-\\      b : ""[-\\\\""
[-/       b : ""[-/    ""
[-//      b : ""[-//   ""
[->x      b : ""[->x   ""
[x->      b : ""[x->   ""
[o->      b : ""[o->   ""
[->o      b : ""[->o   ""
[o->o     b : ""[o->o  ""
[<->      b : ""[<->   ""
[o<->o    b : ""[o<->o""
[x<->x    b : ""[x<->x""
[->>o     b : ""[->>o  ""
```

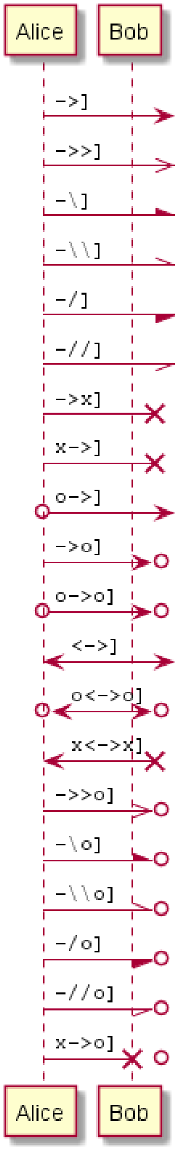
```
[-\o      b : ""[-\o   ""
[-\\o     b : ""[-\\\\o""
[-/o      b : ""[-/o   ""
[-//o     b : ""[-//o  ""
[x->o     b : ""[x->o  ""
@enduml
```



1.37.4 アウト（']' を使用）

```
@startuml
participant Alice as a
participant Bob   as b
a ->]       : ""->]    ""
a ->>]      : ""->>]   ""
a -\]       : ""-\]    ""
a -\\]      : ""-\\\\]""
a -/]       : ""-/]    ""
a -//]      : ""-//]   ""
a ->x]      : ""->x]   ""
a x->]      : ""x->]   ""
a o->]      : ""o->]   ""
a ->o]      : ""->o]   ""
```

```
a o->o]     : ""o->o] ""
a <->]      : ""<->]  ""
a o<->o]    : ""o<->o]""
a x<->x]    : ""x<->x]""
a ->>o]     : ""->>o] ""
a -\o]      : ""-\o]  ""
a -\\o]     : ""-\\\\o]""
a -/o]      : ""-/o]  ""
a -//o]     : ""-//o] ""
a x->o]     : ""x->o] ""
@enduml
```

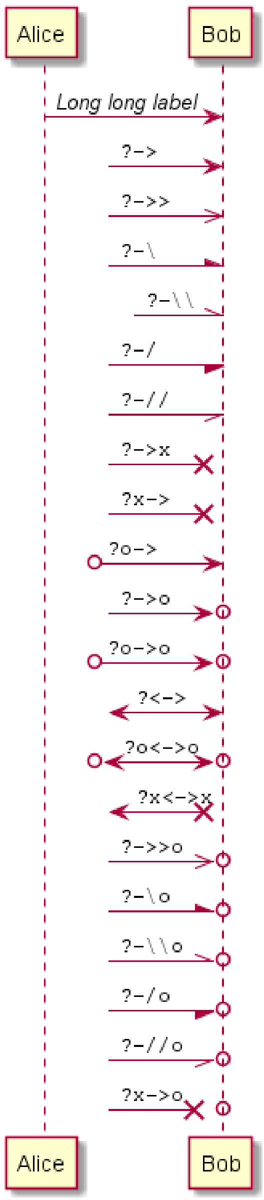


1.37.5 短いインとアウトのメッセージ（'?' を使用）

1.37.6 短いイン（'?' を使用）

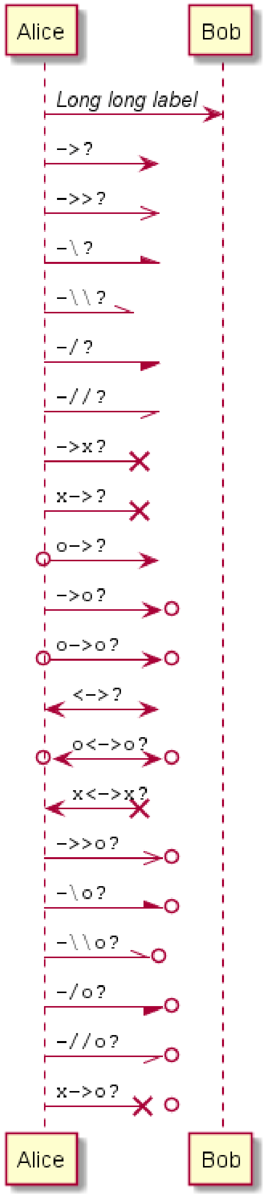
```
@startuml
participant Alice as a
participant Bob   as b
a ->      b : //Long long label//
?->       b : ""?->   ""
?->>      b : ""?->>  ""
```

```
?-\      b : ""?-\   ""
?-\\     b : ""?-\\\\""
?-/      b : ""?-/   ""
?-//     b : ""?-//  ""
?->x     b : ""?->x  ""
?x->     b : ""?x->  ""
?o->     b : ""?o->  ""
?->o     b : ""?->o  ""
?o->o    b : ""?o->o ""
?<->     b : ""?<->  ""
?o<->o   b : ""?o<->o""
?x<->x   b : ""?x<->x""
?->>o    b : ""?->>o ""
?-\o     b : ""?-\o  ""
?-\\o    b : ""?-\\\\o ""
?-/o     b : ""?-/o  ""
?-//o    b : ""?-//o ""
?x->o    b : ""?x->o ""
@enduml
```



1.37.7 短いアウト（'?' を使用）

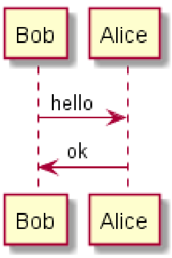
```
@startuml
participant Alice as a
participant Bob   as b
a ->     b : //Long long label//
a ->?      : ""->?   ""
a ->>?     : ""->>?  ""
a -\?      : ""-\?   ""
a -\\?     : ""-\\\\?""
a -/?      : ""-/?   ""
a -//?     : ""-//?  ""
a ->x?     : ""->x?  ""
a x->?     : ""x->?  ""
a o->?     : ""o->?  ""
a ->o?     : ""->o?  ""
a o->o?    : ""o->o? ""
a <->?     : ""<->?  ""
a o<->o?   : ""o<->o?""
a x<->x?   : ""x<->x?""
a ->>o?    : ""->>o? ""
a -\o?     : ""-\o?  ""
a -\\o?    : ""-\\\\o?""
a -/o?     : ""-/o?  ""
a -//o?    : ""-//o? ""
a x->o?    : ""x->o? ""
@enduml
```



1.38 特有の skinparam

1.38.1 デフォルト

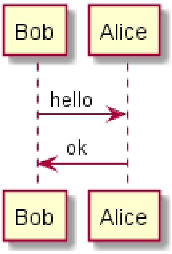
```
@startuml
Bob -> Alice : hello
Alice -> Bob : ok
@enduml
```



1.38.2 ライフラインの設定（lifelineStrategy）

- nosolid（デフォルト）

```
@startuml
skinparam lifelineStrategy nosolid
Bob -> Alice : hello
Alice -> Bob : ok
@enduml
```

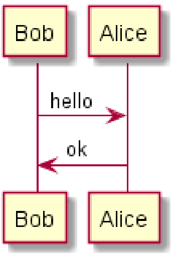


[Ref. QA-9016]

- solid

シーケンス図のライフラインを実線で表示するには、`skinparam lifelineStrategy solid` を設定します：

```
@startuml
skinparam lifelineStrategy solid
Bob -> Alice : hello
Alice -> Bob : ok
@enduml
```



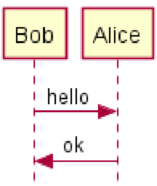
[Ref. QA-2794]

1.38.3 厳密な UML スタイル（style strictuml）

厳密な UML に準拠する（矢印の端を矢じり形ではなく三角形にする等）には、次のようにします：

- `skinparam style strictuml`

```
@startuml
skinparam style strictuml
Bob -> Alice : hello
Alice -> Bob : ok
@enduml
```

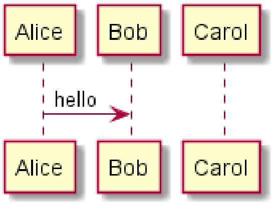


[Ref. QA-1047]

1.39 未接続の分類子を表示しない

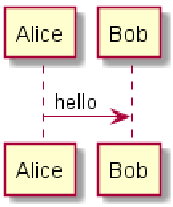
デフォルトでは、すべての分類子が表示されます。

```
@startuml
participant Alice
participant Bob
participant Carol

Alice -> Bob : hello
@enduml
```



`hide unlinked` を指定すると、接続されていない分類子を非表示にできます。

```
@startuml
hide unlinked
participant Alice
participant Bob
participant Carol

Alice -> Bob : hello
@enduml
```



[Ref. QA-4247]

2 ユースケース図

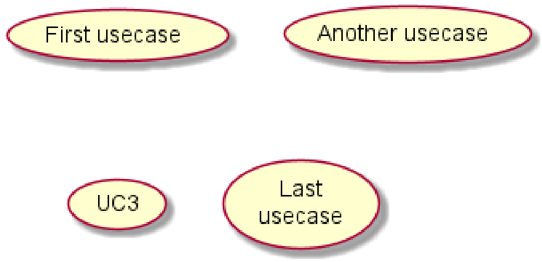
いくつかの例を示します。

2.1 ユースケース

ユースケースは丸括弧で囲んで使います (丸括弧の対は楕円に似ているからです)。

usecase キーワードを使ってユースケースを定義することもできます。as キーワードを使ってエイリアスを定義することもできます。このエイリアスはあとで、ユースケースの関係を定義するために使います。

```
@startuml

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

@enduml
```



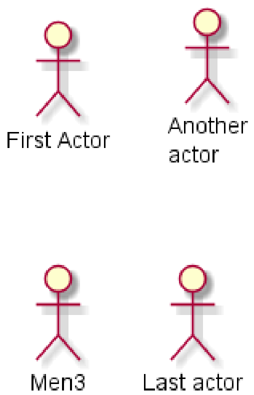
2.2 アクター

アクターは 2 つのコロンで囲まれます。

actor キーワードを使ってアクターを定義することもできます。as キーワードを使ってエイリアスを定義することもできます。このエイリアスはあとで、ユースケースの関係を定義するために使います。

後から説明しますが、アクターの定義は必須ではありません。

```
@startuml

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

@enduml
```



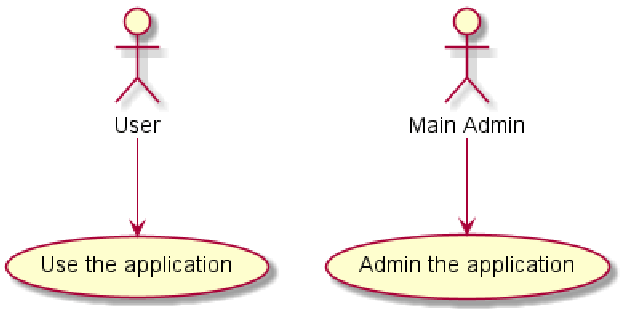
2.3 アクターのスタイルを変更する

アクターのスタイルを、デフォルトの棒人間以外に変更できます：

- `skinparam actorStyle awesome` コマンドで、awesome man スタイル
- `skinparam actorStyle hollow` コマンドで、hollow man スタイル

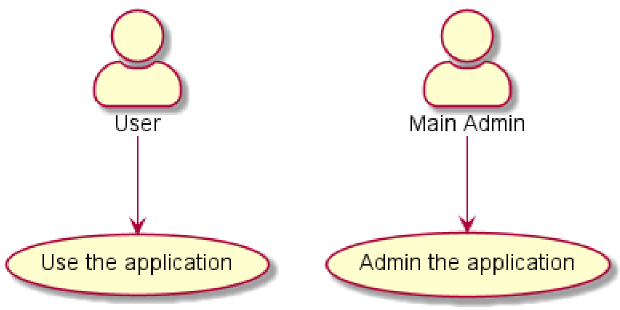
2.3.1 棒人間 (デフォルト)

```
@startuml
:User: --> (Use)
"Main Admin" as Admin
"Use the application" as (Use)
Admin --> (Admin the application)
@enduml
```



2.3.2 Awesome man

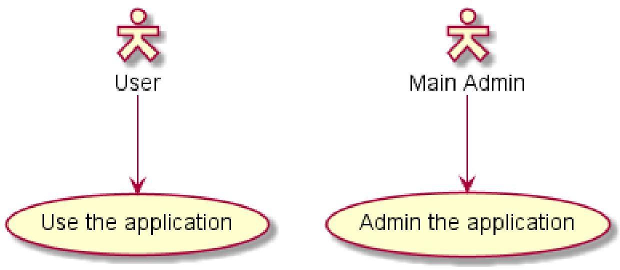
```
@startuml
skinparam actorStyle awesome
:User: --> (Use)
"Main Admin" as Admin
"Use the application" as (Use)
Admin --> (Admin the application)
@enduml
```



[Ref. QA-10493]

2.3.3 Hollow man

```
@startuml
skinparam actorStyle Hollow
:User: --> (Use)
"Main Admin" as Admin
"Use the application" as (Use)
Admin --> (Admin the application)
@enduml
```



[Ref. PR#396]

2.4　ユースケースの説明

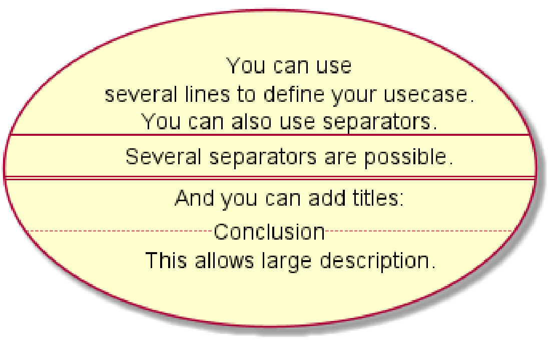
クオート記号を使うことにより、複数行にわたる説明を記述できます。

また、次の区切り記号を使用できます：

- -- (ダッシュ)
- .. (ピリオド)
- == (イコール)
- __ (アンダースコア)

これらのペアで囲んで、その間にテキストを記述することで、区切り記号の中にタイトルを記入できます。

```
@startuml

usecase UC1 as "You can use
several lines to define your usecase.
You can also use separators.
--
Several separators are possible.
==
And you can add titles:
..Conclusion..
This allows large description."

@enduml
```

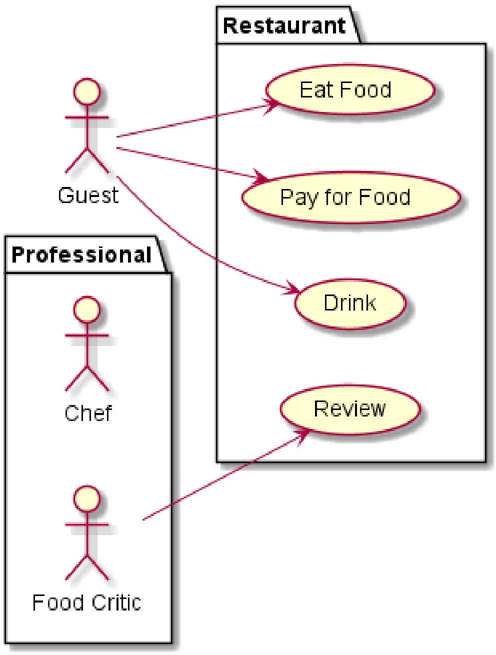


2.5　パッケージ

パッケージを使用して、アクターやユースケースをグループ化できます。

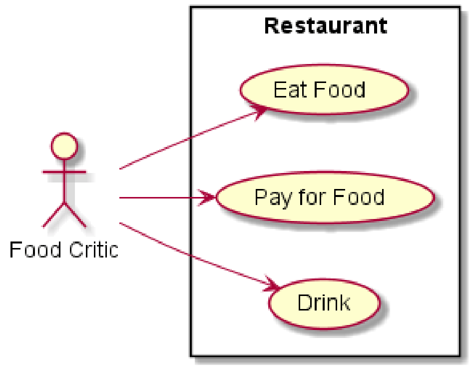
```
@startuml
left to right direction
actor Guest as g
package Professional {
  actor Chef as c
```

```
  actor "Food Critic" as fc
}
package Restaurant {
  usecase "Eat Food" as UC1
  usecase "Pay for Food" as UC2
  usecase "Drink" as UC3
  usecase "Review" as UC4
}
fc --> UC4
g --> UC1
g --> UC2
g --> UC3
@enduml
```



rectangle を使用するとパッケージの見た目を変更できます。

```
@startuml
left to right direction
actor "Food Critic" as fc
rectangle Restaurant {
  usecase "Eat Food" as UC1
  usecase "Pay for Food" as UC2
  usecase "Drink" as UC3
}
fc --> UC1
fc --> UC2
fc --> UC3
@enduml
```



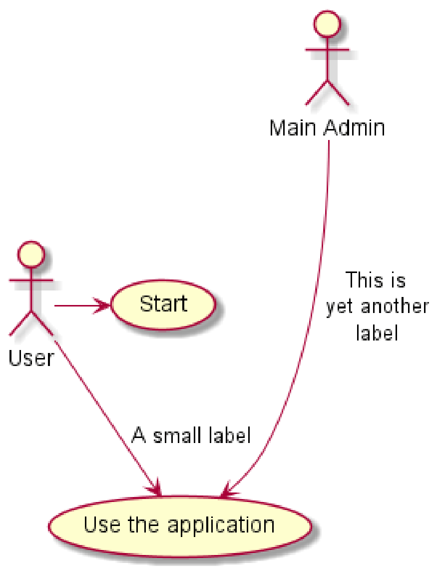
2.6 簡単な例

アクターとユースケースを繋げるには --> 矢印を使います。

矢印に使うハイフン - の数を増やすと矢印を長くできます。矢印の定義に : を使うことにより矢印にラベルをつけることができます。

以下の例では *User* は定義なしにアクターとして使われています。

```
@startuml

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

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

@enduml
```

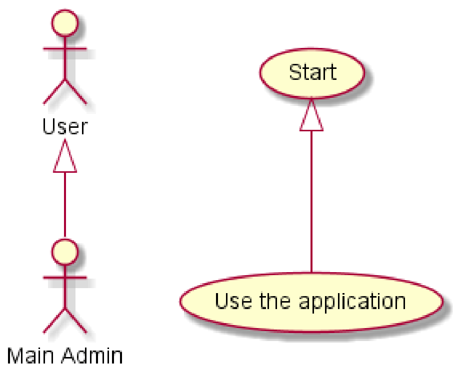


2.7 継承

もしアクターやユースケースが継承をする場合には、<|-- 記号を使います。

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

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

```
@enduml
```

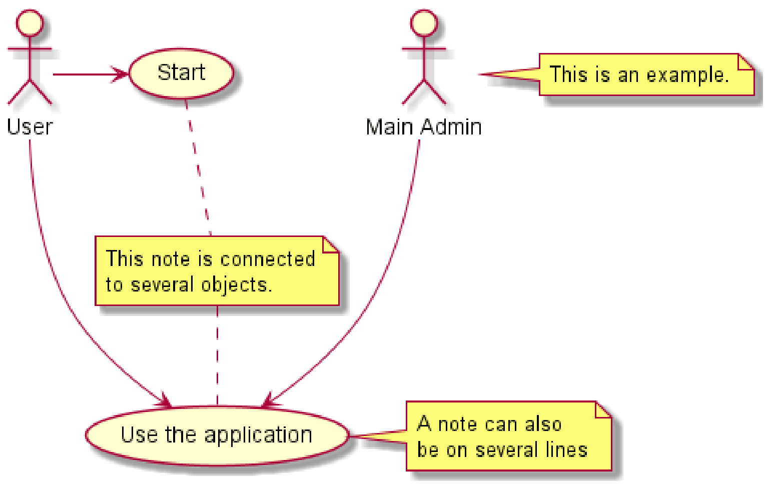


2.8 ノートの使用方法

オブジェクトに関連のあるノートを作成するには `note left of` 、`note right of` 、`note top of` 、`note bottom of` キーワードを使います。

または `note` キーワードを使ってノートを作成し、`..` 記号を使ってオブジェクトに紐づけることができます。

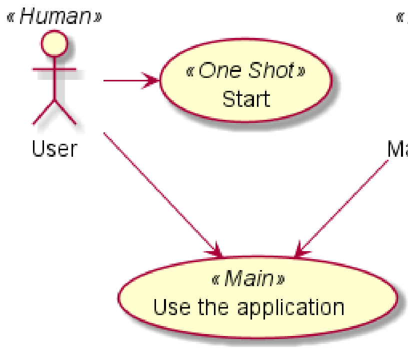
```
@startuml
:Main Admin: as Admin
(Use the application) as (Use)

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

Admin ---> (Use)

note right of Admin : This is an example.

note right of (Use)
  A note can also
  be on several lines
end note

note "This note is connected\nto several objects." as N2
(Start) .. N2
N2 .. (Use)
@enduml
```



2.9 ステレオタイプ

<< と >> を使い、アクターとユースケースを定義中にステレオタイプを追加できます。

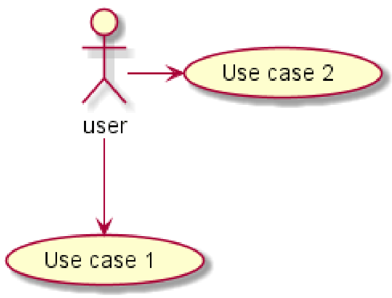
```
@startuml
User << Human >>
:Main Database: as MySql << Application >>
(Start) << One Shot >>
(Use the application) as (Use) << Main >>

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

MySql --> (Use)

@enduml
```



2.10 矢印の方向を変えるには

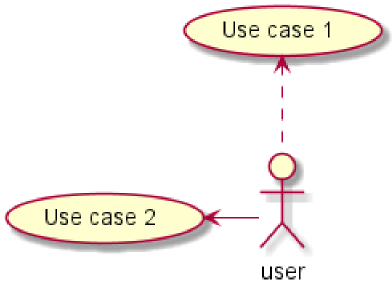
デフォルトでは、クラス間の線は 2 個のハイフン -- で表され、縦方向につながります。横方向の線を描くには以下のようにハイフン 1 つかドット 1 つを書きます。

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



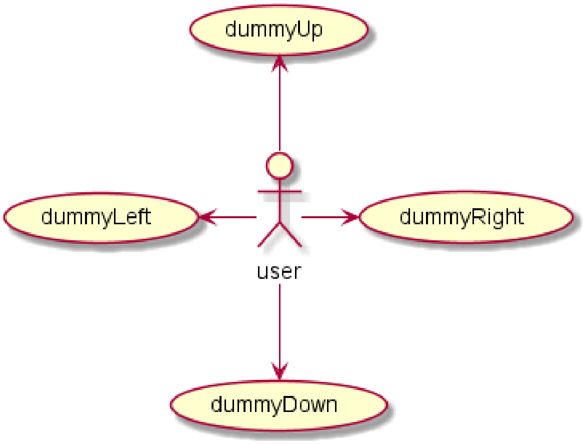
線を反対にすることでも方向を変えることができます。

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



矢印の内側に left、right、up、down を書くことによっても線の方向を変えられます。

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

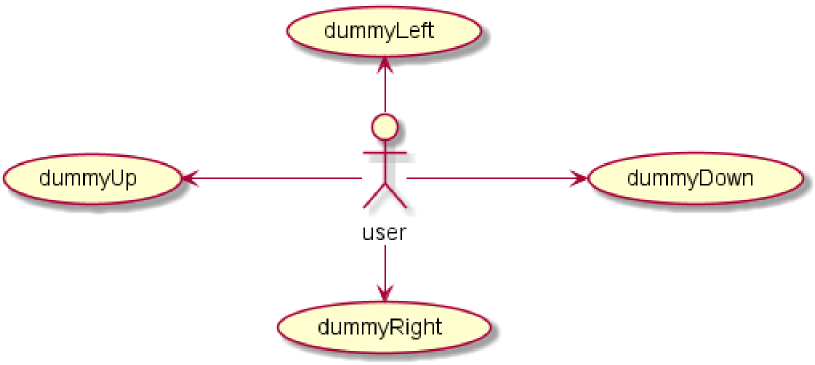


例えば、-down- ではなく -d- など、各方向の頭文字、または頭 2 文字（-do- ）だけ使って矢印を短く記述することも出来ます。

ただし、この機能の使いすぎには注意しましょう。ほとんどの場合、特別なことをしなくても *Graphviz* がその場にあった表示を選びます。

left to right direction パラメータを使用した場合は、次のようになります：

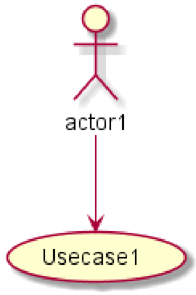
```
@startuml
left to right direction
:user: -left-> (dummyLeft)
:user: -right-> (dummyRight)
:user: -up-> (dummyUp)
:user: -down-> (dummyDown)
@enduml
```



2.11 図を分割する

newpage キーワードは、いくつかのページや画像に図を分割します。

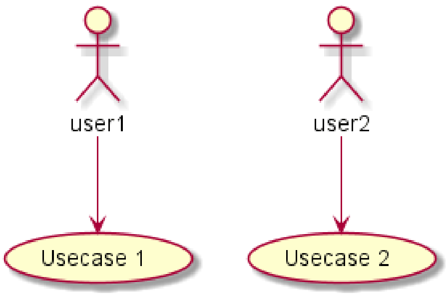
```
@startuml
:actor1: --> (Usecase1)
newpage
:actor2: --> (Usecase2)
@enduml
```



2.12 左から右に描画する

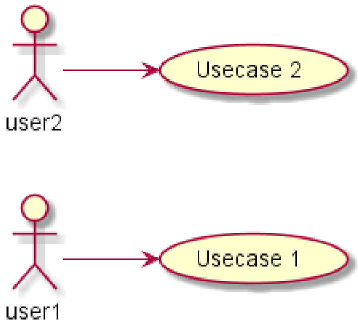
デフォルトの作図方向は **top to bottom** となっています。

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

@enduml
```



作図方向を **left to right** に変更するには `left to right direction` コマンドを使います。

```
@startuml

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

@enduml
```



2.13 スキン設定（Skinparam）

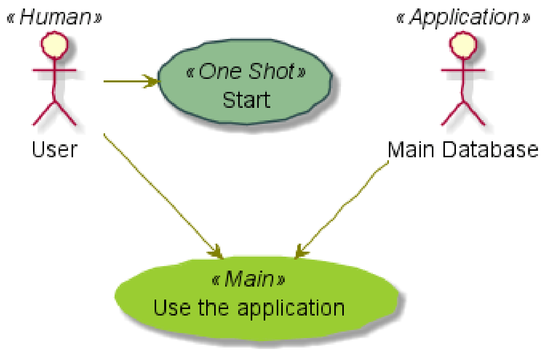
ダイアグラムの色やフォントを変更するには skinparam コマンドを使用します。

このコマンドは以下の場面で使用できます。

- ダイアグラム定義内で他のコマンドを同様に。
- インクルードされたファイル内。
- 設定ファイルのコマンドライン内や ANT タスク内。

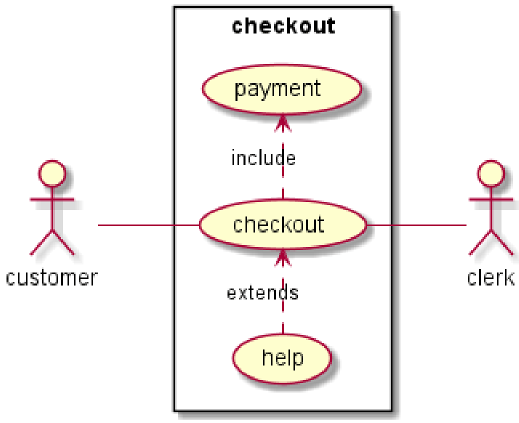
個別のステレオタイプ付きアクターやユースケースにそれぞれ色やフォントを定義することができます。

```
@startuml
skinparam handwritten true

skinparam usecase {
BackgroundColor DarkSeaGreen
BorderColor DarkSlateGray

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

ArrowColor Olive
ActorBorderColor black
ActorFontName Courier

ActorBackgroundColor<< Human >> Gold
}

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

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

MySql --> (Use)

@enduml
```



2.14 完全な例

```
@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
```

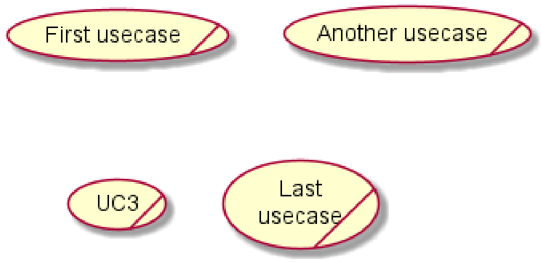


2.15 ビジネスユースケース

/を加えると、ビジネスユースケースを作成できます。

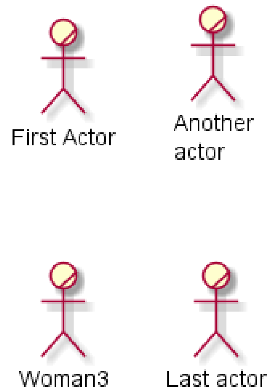
2.15.1 ビジネスユースケース

```
@startuml

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

@enduml
```



2.15.2 ビジネスアクター

```
@startuml

:First Actor:/
:Another\nactor:/ as Man2
actor/ Woman3
actor/ :Last actor: as Person1

@enduml
```



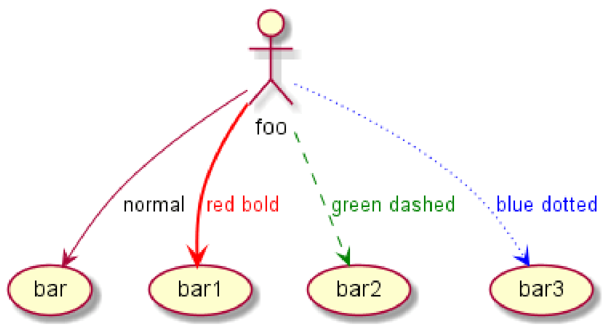
[Ref. QA-12179]

2.16 Change arrow color and style (inline style)

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

- `#color;line.[bold|dashed|dotted];text:color`

```
@startuml
actor foo
foo --> (bar) : normal
foo --> (bar1) #line:red;line.bold;text:red  : red bold
foo --> (bar2) #green;line.dashed;text:green : green dashed
foo --> (bar3) #blue;line.dotted;text:blue   : blue dotted
@enduml
```



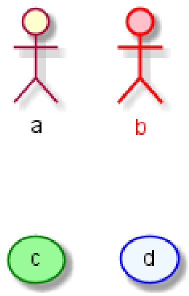
[Ref. QA-3770 and QA-3816] [See similar feature on deployment-diagram or class diagram]

2.17 Change element color and style (inline style)

You can change the color or style of individual element using the following notation:

- `#[color|back:color];line:color;line.[bold|dashed|dotted];text:color`

```
@startuml
actor a
actor b #pink;line:red;line.bold;text:red
usecase c #palegreen;line:green;line.dashed;text:green
usecase d #aliceblue;line:blue;line.dotted;text:blue
@enduml
```

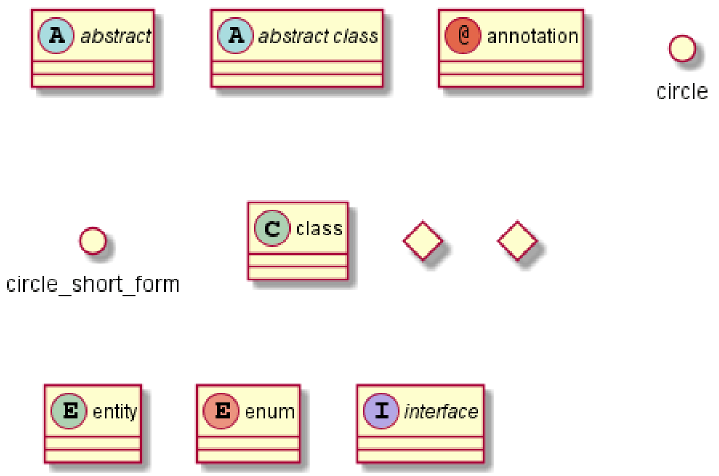


[Ref. QA-5340 and adapted from QA-6852]

3 クラス図

3.1 要素の定義

```
@startuml
abstract        abstract
abstract class  "abstract class"
annotation      annotation
circle          circle
()              circle_short_form
class           class
diamond         diamond
<>              diamond_short_form
entity          entity
enum            enum
interface       interface
@enduml
```



3.2 クラス間の関係

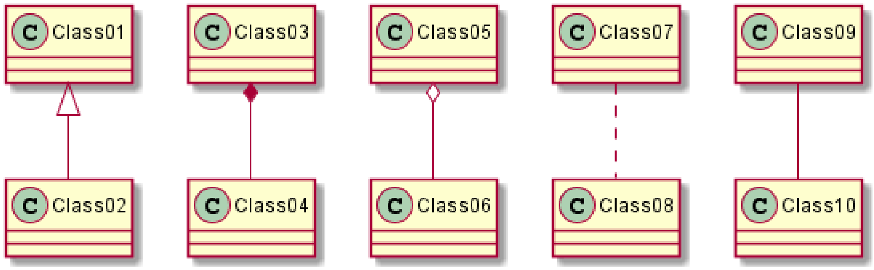
クラス間の関係は次の記号を使用して定義されています:

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

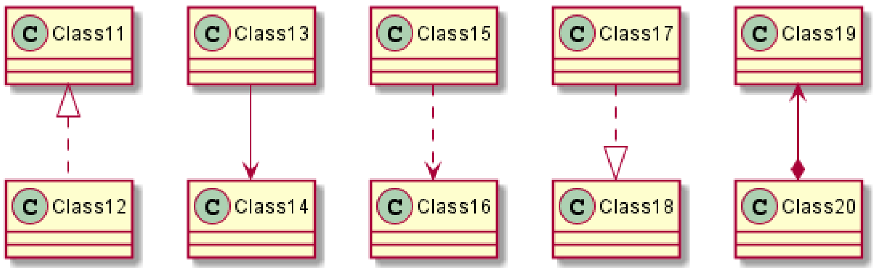
`--` を `..` に置き換えると点線にできます。

これらのルールを知ることで、以下の図面を描くことができます:

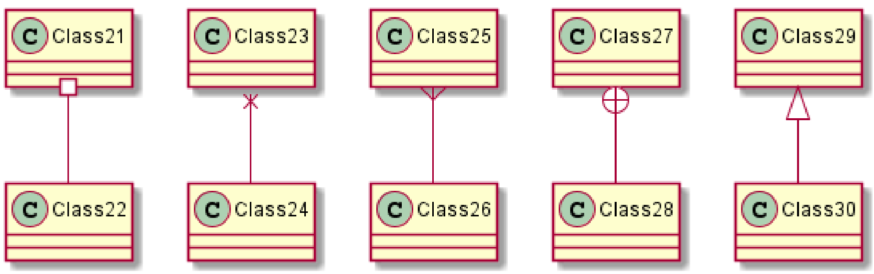
```
@startuml
Class01 <|-- Class02
Class03 *-- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10
@enduml
```




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




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

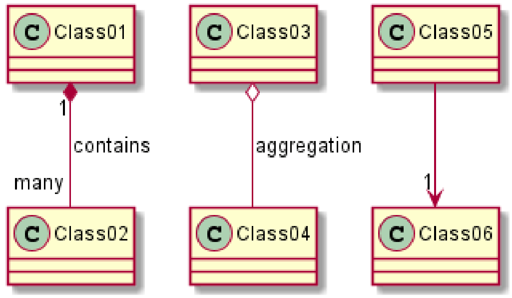


3.3 関係のラベル

: にテキストを続けることによって、関係へラベルを追加することが可能です。

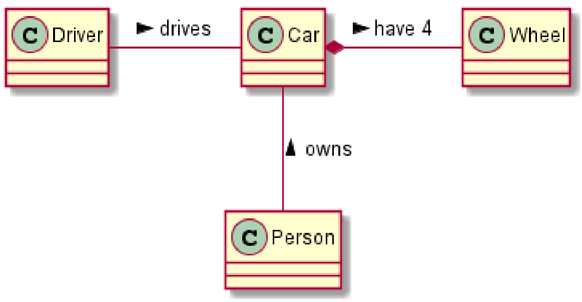
多重度を示す為に関係のそれぞれの側にダブルクォーテーション"" を使うことができます。

```
@startuml

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

Class03 o-- Class04 : aggregation

Class05 --> "1" Class06

@enduml
```



ラベルの最初または最後に < か > を使って、他のオブジェクトへの関係を示す矢印を追加できます。

```
@startuml
class Car

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

@enduml
```

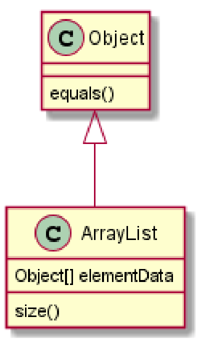


3.4 メソッドの追加

: に続けてフィールド名やメソッド名を記述すると、フィールドやメソッドを宣言できます。

システムは括弧をチェックしてメソッドとフィールドのどちらなのかを選択します。

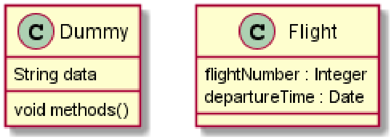
```
@startuml
Object <|-- ArrayList

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

@enduml
```



波括弧 {} を使って、フィールドやメソッドをくくることもできます。

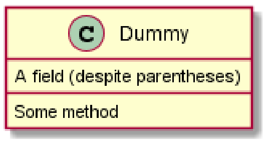
構文はタイプや名前の順番について非常に柔軟であることに注意してください。

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

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



{field} や {method} 修飾子を用いれば、構文によりフィールドやメソッドだと通常は解釈されるものを強制的に変更することができます。

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

@enduml
```

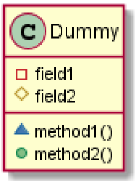


3.5 可視性の定義

メソッドやフィールドを定義するときに対応する項目の可視性を定義する記号を使用することができます。

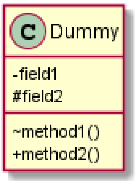
Character	Icon for field	Icon for method	Visibility
-	□	■	private
#	◇	◆	protected
~	△	▲	package private
+	○	●	public

```
@startuml

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

@enduml
```



コマンド `skinparam classAttributeIconSize 0` を使用してこの機能を切ることができます。

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

@enduml
```

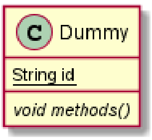


3.6 Abstract と Static

静的または抽象的なメソッドまたはフィールドは `{static}` または `{abstract}` 修飾子を使用することで定義することができます。

これらの修飾子は行の始めまたは終りに使用することができます。`{static}` の代わりに `{classifier}` もまた使用できます。

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



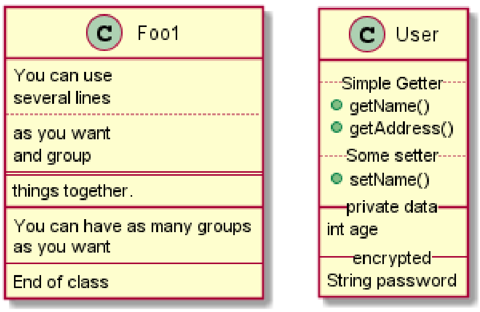
3.7 高等なクラス本体

デフォルトでは，メソッドやフィールドは PlantUML によって自動再編成されます。メソッドやフィールドに独自の順序付けを定義するためのセパレータを使用できます。以下のセパレータが使用できます：`--` `..` `==` `__`

セパレータ内でタイトルを使用することもできます：

```
@startuml
class Foo1 {
  You can use
  several lines
  ..
  as you want
```

```
  and group
  ==
  things together.
  --
  You can have as many groups
  as you want
  --
  End of class
}

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

@enduml
```



3.8 注釈とステレオタイプ

ステレオタイプは、キーワード `class` に `<<` と `>>` で定義されます。

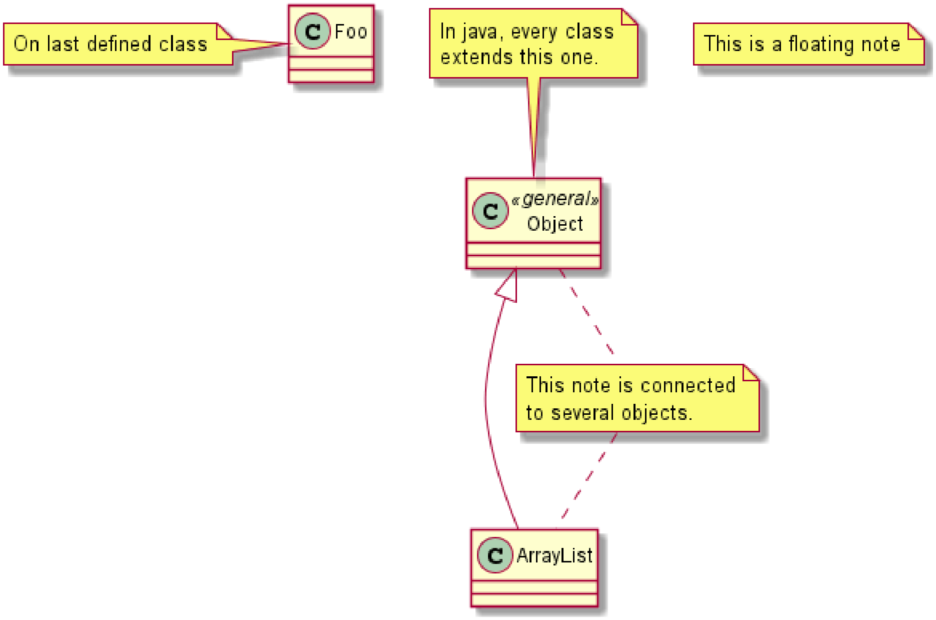
注釈の定義には、キーワード `note left of`, <code>note right of</code>, `note top of`, `note bottom of` も使用できます。

クラス定義の最後には `note left`, `note right`, `note top`, `note bottom` も使用できます。

注釈は、キーワード `note` とで単独に定義することができ、記号 `..` を使用して他のオブジェクトとリンクすることもできます。

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

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

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

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

```
@enduml
```



3.9 注釈の詳細

次のようないくつかの HTML タグを使用することも可能です (Creole 表現を参照)：

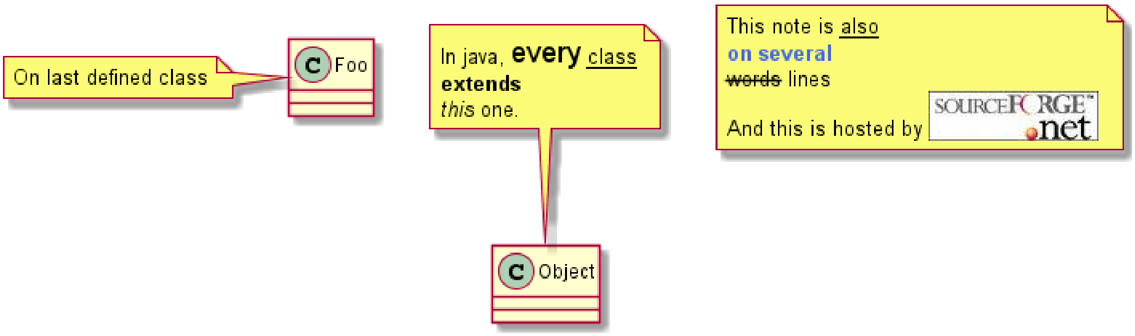
- <b>
- <u>
- <i>
- <s>, <del>, <strike>
- <font color="#AAAAAA"> or <font color="colorName">
- <color:#AAAAAA> or <color:colorName>
- <size:nn> to change font size
- <img src="file"> or <img:file>: the file must be accessible by the filesystem

また、複数行にまたがる注釈も可能です。

クラス定義の最後には `note left`, `note right`, `note top`, `note bottom` も使用できます。

```
@startuml

class Foo
note left: On last defined class

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

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

```
@enduml
```

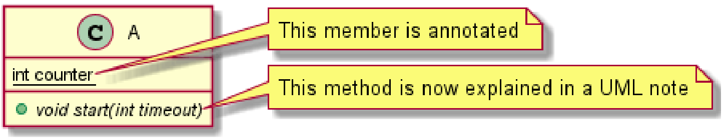


3.10 フィールド (フィールド、属性、メンバー) またはメソッドへの注釈

フィールド (フィールド、属性、メンバー) またはメソッドに注釈を追加することができます。

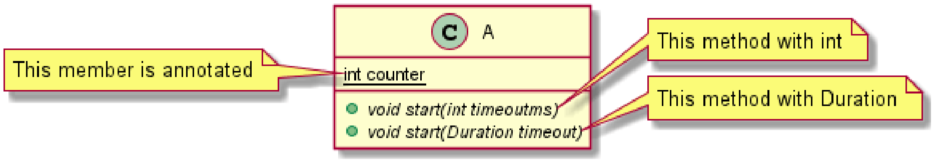
3.10.1 フィールドまたはメンバーへの注釈

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



3.10.2 同名のメソッドへの注釈

```
@startuml
class A {
{static} int counter
+void {abstract} start(int timeoutms)
+void {abstract} start(Duration timeout)
}
note left of A::counter
  This member is annotated
end note
note right of A::"start(int timeoutms)"
  This method with int
end note
note right of A::"start(Duration timeout)"
  This method with Duration
end note
@enduml
```



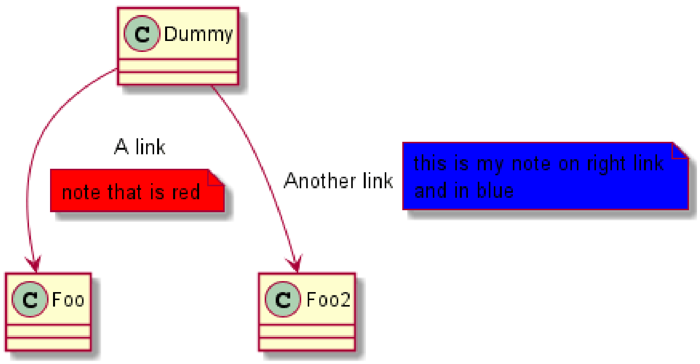
[Ref. QA-3474 and QA-5835]

3.11 リンクへの注釈

リンク定義の直後に `note on link` を使用して、リンクに注釈を加えることが可能です。

もし注釈の相対位置を変えたい場合には、ラベル `note left on link`, `note right on link`, `note top on link`, `note bottom on link` も使用できます。

```
@startuml

class Dummy
Dummy --> Foo : A link
note on link #red: note that is red

Dummy --> Foo2 : Another link
note right on link #blue
this is my note on right link
and in blue
end note

@enduml
```



3.12 抽象クラスとインタフェース

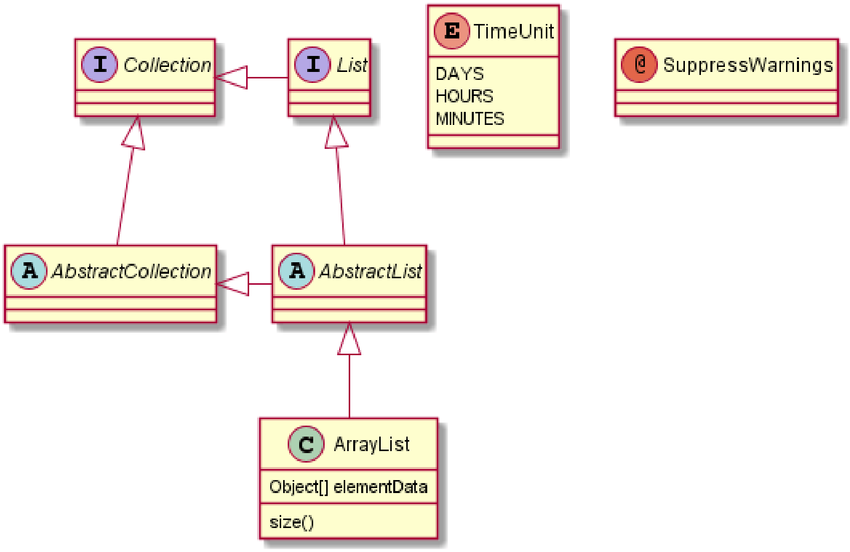
抽象クラスは、キーワード `abstract` または `abstract class` を使用して宣言できます。

そのクラスは イタリック体で印字されます。

キーワード `interface`, `annotation` と `enum` も使用できます。

```
@startuml

abstract class AbstractList
abstract AbstractCollection
interface List
interface Collection

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

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

```
AbstractList <|-- ArrayList

class ArrayList {
  Object[] elementData
  size()
}

enum TimeUnit {
  DAYS
  HOURS
  MINUTES
}

annotation SuppressWarnings

@enduml
```

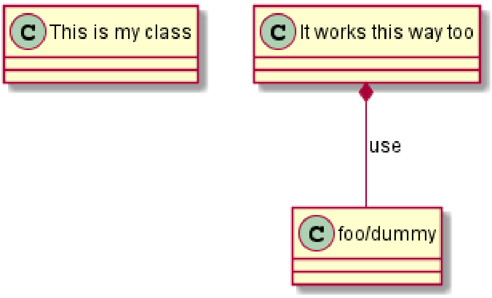


3.13 非文字の使用

クラス (または列挙型...) の表示に文字以外を使用したい場合は、次のいずれかの方法ですることができます :

- クラス定義にはキーワード as を使用する
- クラス名の前後に引用符 "" を入れる

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

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



3.14 属性、メソッド等の非表示

コマンド hide/show を使用して、クラスの表示をパラメータ化できます。

基本のコマンドは hide empty members です。このコマンドは属性やメソッドが空の場合に非表示にします。

empty members の代わりに使用することができます：

- empty fields または empty attributes は空のフィールドに、
- empty methods は空のメソッドに、
- fields または attributes は、それらが記述されていても非表示になります、
- methods はメソッドが記述されていても非表示になります、
- members はフィールドと メソッドが記述されていても非表示になります、
- circle はクラス名の前の丸で囲んだ文字に、
- stereotype はステレオタイプに。

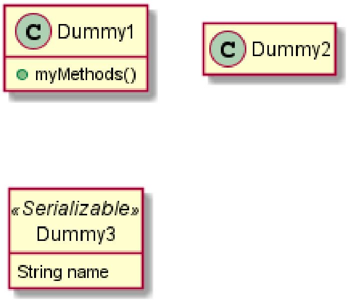
キーワード hide または show のすぐ後ろに提供することもできます：

- class は全てのクラスに、
- interface は全てのインタフェースに、
- enum は全ての列挙型に、
- <<foo1>> は *foo1* でステレオタイプ化されたクラスに、
- 既存のクラス名。

コマンド show/hide をルールや例外の定義にそれぞれ使用することができます。

```
@startuml

class Dummy1 {
  +myMethods()
}

class Dummy2 {
  +hiddenMethod()
}

class Dummy3 <<Serializable>> {
String name
}

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

```
@enduml
```

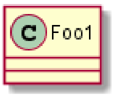


3.15 非表示クラス

コマンド `show/hide` でクラスを非表示にすることができます。

これは大規模なインクルードファイルを定義する場合で、ファイルのインクルードの後でいくつかのクラスを非表示にしたい場合に有用である可能性が有ります。

```
@startuml

class Foo1
class Foo2

Foo2 *-- Foo1

hide Foo2

@enduml
```

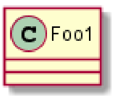


3.16 Remove classes

You can also use the `remove` commands to remove classes.

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

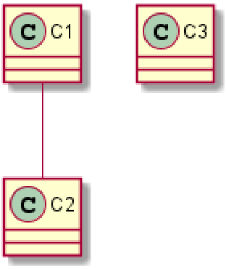
```
@startuml

class Foo1
class Foo2

Foo2 *-- Foo1

remove Foo2

@enduml
```



3.17 Hide or Remove unlinked class

By default, all classes are displayed:

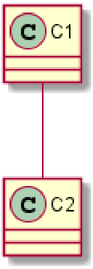
```
@startuml
class C1
class C2
class C3
C1 -- C2
@enduml
```



But you can:

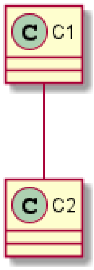
- `hide @unlinked` classes:

```
@startuml
class C1
class C2
class C3
C1 -- C2

hide @unlinked
@enduml
```



- or `remove @unlinked` classes:

```
@startuml
class C1
class C2
class C3
C1 -- C2

remove @unlinked
@enduml
```

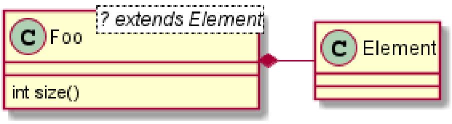


[Adapted from QA-11052]

3.18 ジエネリクスの使用

括弧 < と > を使用してジェネリクスの使用をクラスに定義できます。

```
@startuml

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

@enduml
```



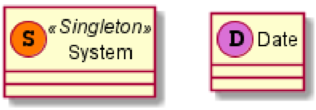
この描画は `skinparam genericDisplay old` コマンドにより非表示にすることができます。

3.19 特殊な目印

通常、目印文字 (C,I,E,A) は、クラス、インターフェイス、列挙型と抽象クラスのために使用されます。

しかし、つぎの例のように単一の文字と色を追加し、ステレオタイプを定義するクラスに独自の目印を作成することができます：

```
@startuml

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



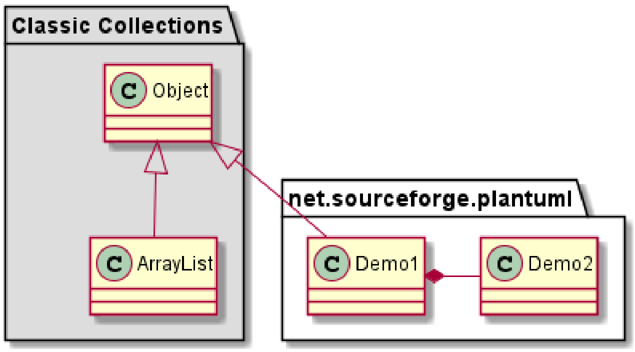
3.20 パッケージ

キーワード package を使用してパッケージを定義でき、必要に応じてパッケージの背景色（HTML カラーコードまたは名前）を宣言します。

パッケージ定義は入れ子にできることに注意してください。

```
@startuml

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

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

@enduml
```

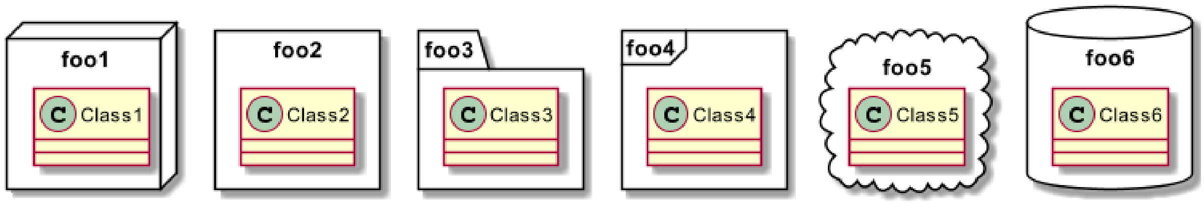


3.21 パッケージスタイル

パッケージに利用可能なさまざまなスタイルがあります。

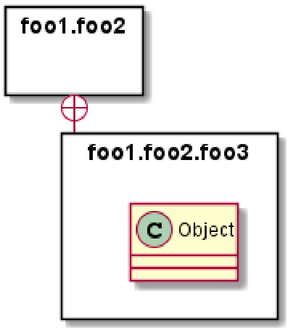
コマンド skinparam packageStyle を使用してデフォルトのスタイルを設定する、またはパッケージのステレオタイプを使用する、のどちらかで指定することができます。or by using a stereotype on the package:

```
@startuml
scale 750 width
package foo1 <<Node>> {
  class Class1
}

package foo2 <<Rectangle>> {
  class Class2
}

package foo3 <<Folder>> {
  class Class3
}

package foo4 <<Frame>> {
  class Class4
}

package foo5 <<Cloud>> {
  class Class5
}

package foo6 <<Database>> {
  class Class6
}

@enduml
```



次の例のように、パッケージ間のリンクを定義することもできます：

```
@startuml

skinparam packageStyle rectangle

package foo1.foo2 {
}

package foo1.foo2.foo3 {
  class Object
}

foo1.foo2 +-- foo1.foo2.foo3

@enduml
```



3.22 名前空間

パッケージ内では、クラスの名前はこのクラスの一意な識別子です。それは、全く同じ名前の 2 つのクラスを異なるパッケージに持つことができないことを意味します。

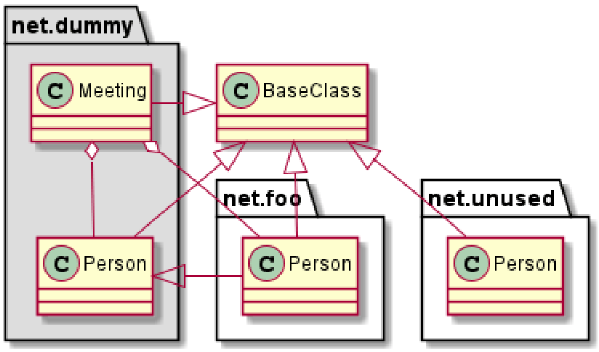
そのような場合、パッケージの代わりに名前空間を使用したらいいでしょう。

名前空間からの完全修飾名によりクラスを参照することができます。デフォルトの名前空間からのクラスは、一つのドットで修飾します。

明示的に名前空間を作成する必要はないことに注意してください：完全修飾されたクラスは自動的に適切な名前空間に置かれています。

```
@startuml

class BaseClass

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

    .BaseClass <|- Meeting
}

namespace net.foo {
```

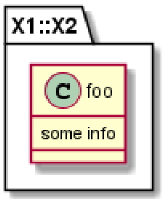
```
  net.dummy.Person  <|- Person
  .BaseClass <|-- Person

  net.dummy.Meeting o-- Person
}

BaseClass <|-- net.unused.Person

@enduml
```



3.23 自動的に名前空間を作成する

コマンド `set namespaceSeparator ???` を使用して、（ドット以外の）別の区切り文字を定義することができます。

```
@startuml

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

@enduml
```



コマンド `set namespaceSeparator none` を使用して、自動的に名前空間を作成する機能を無効にすることができます。

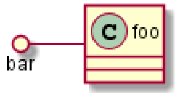
```
@startuml

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

@enduml
```

3.24 ロリポップ(棒付きキャンディー)インタフェース

次の構文を使用して、クラスにロリポップインタフェースを定義することもできます:

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

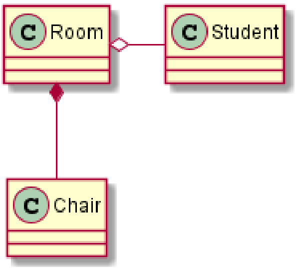
```
@startuml
class foo
bar ()- foo
@enduml
```



3.25 矢印の向きを変える

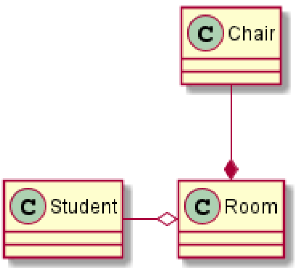
デフォルトではクラス間のリンクは 2 つのダッシュ -- を持っており、垂直に配向されています。次のように単一のダッシュ(またはドット)を置くことによって水平方向にリンクを使用することが可能です。

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



リンクをひっくり返すことにより向きを変えることができる:

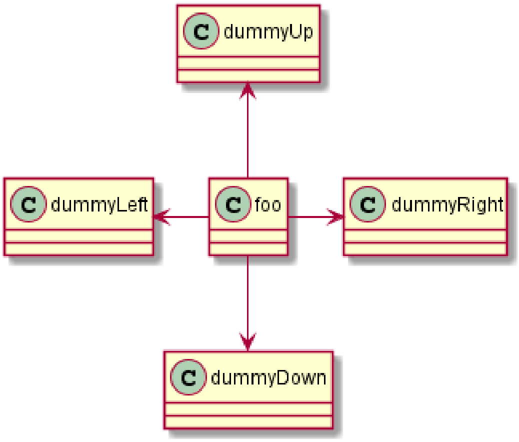
```
@startuml
Student -o Room
Chair --* Room
@enduml
```



キーワード left, right, up, down を矢印の内側に置くことにより、矢印の方向を変えることも可能です:

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

```
foo -down-> dummyDown
@enduml
```

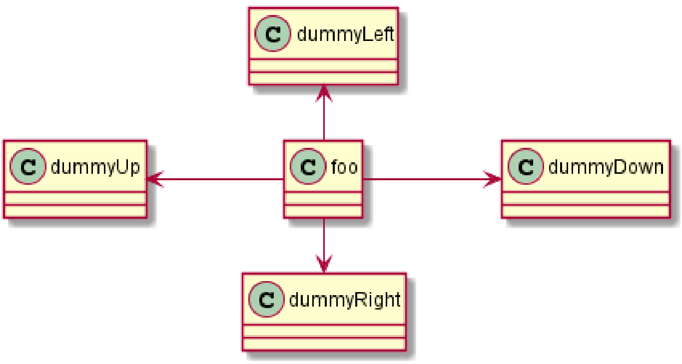


方向の最初の文字を使用して矢印を短縮することができます (例えば、-d- を -down- の代わりに、または、最初の 2 文字 (-do-)。

この機能を悪用してはならないことに注意してください。*Graphviz* は微調整のいらない良い結果を通常は与えてくれます。

left to right direction パラメータを使用した場合は次のようになります。

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

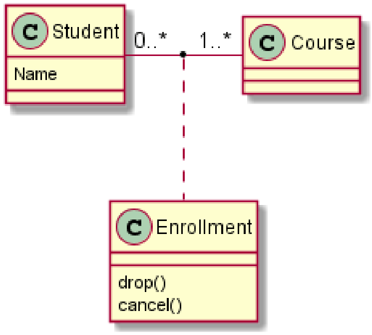


3.26 関連クラス

この例のように、2 つのクラスの関係を定義した後で 関連クラスを定義することができます。

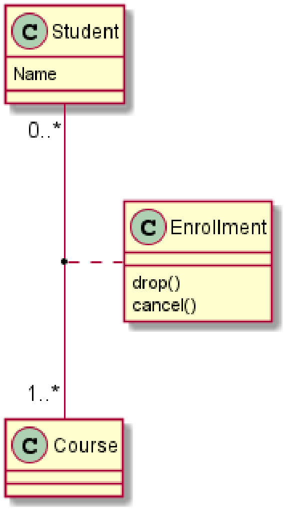
```
@startuml
class Student {
  Name
}
Student "0..*" - "1..*" Course
(Student, Course) .. Enrollment

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

```
@enduml
```



別の方向にそれを定義することができます：

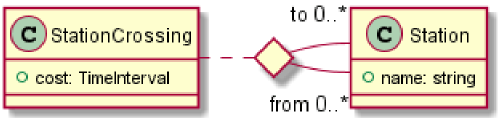
```
@startuml
class Student {
  Name
}
Student "0..*" -- "1..*" Course
(Student, Course) . Enrollment

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



3.27 同一クラスに複数の関連

```
@startuml
class Station {
    +name: string
}

class StationCrossing {
    +cost: TimeInterval
}

<> diamond

StationCrossing . diamond
```

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



[Ref. Incubation: Associations]

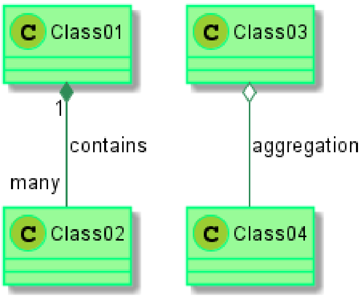
3.28 化粧をする

ダイアグラムの色やフォントを変更するには skinparam コマンドを使用します。

このコマンドは以下の場面で使用できます。

- ダイアグラム定義内で他のコマンドを同様に。
- インクルードされたファイル内。
- 設定ファイルのコマンドライン内や Ant タスク内。

```
@startuml

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

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

Class03 o-- Class04 : aggregation

@enduml
```

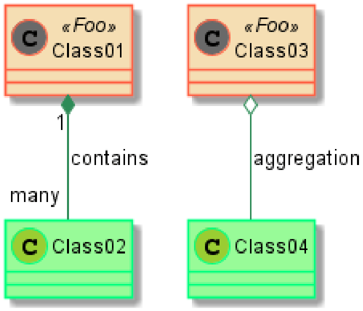


3.29 ステレオタイプの化粧

ステレオタイプクラスに特定の色やフォントを定義することができます。

```
@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
```



3.30 色のグラデーション

表記を使用して、クラスや注釈に個別の色を宣言することが可能です。

標準的な色の名前または RGB コードのいずれかを様々な記法で使用することができます。色を参照してください。

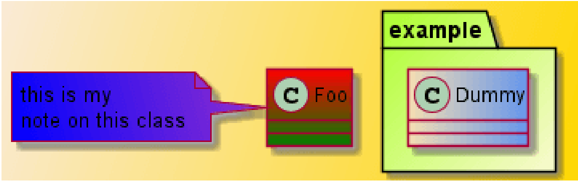
次の構文で背景に色のグラデーションを設定することもできます。2 つの色の名前を次のいずれかで区切って記述してください：

- |
- /
- \
- -

グラデーションの方向に応じて記号を使い分けてください。

例：

```
@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
```



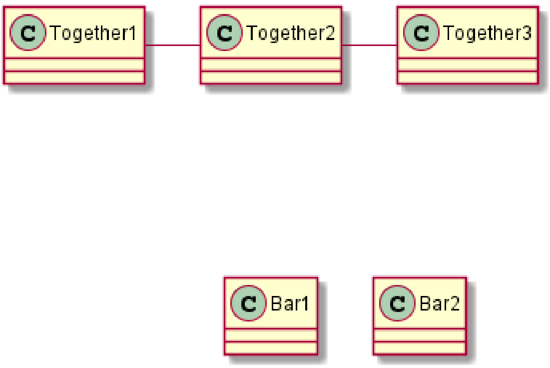
3.31 レイアウトの手助け

ときには、デフォルトのレイアウトでは完璧とは言えないことがあります...

together キーワードを使って複数のクラスをグループにまとめることができます: レイアウトエンジンは、それらのクラスを (あたかも同じパッケージにあるかのように) グループにまとめようとします。

hidden リンクを使ってレイアウトを強制することも可能です。

```
@startuml

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


@enduml
```



3.32 大きなファイルの分割

時には、ある非常に大きな画像ファイルを受け取ることがあるでしょう。

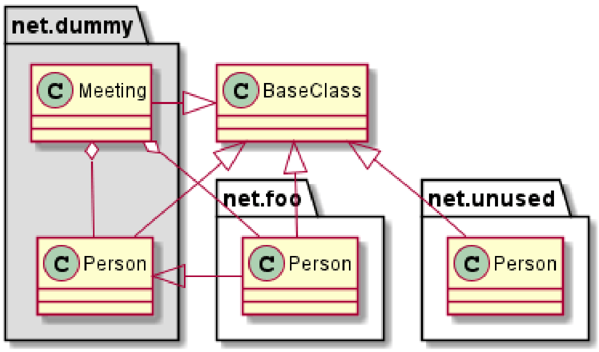
生成された画像を複数のファイルに分割するコマンド page (hpages)x(vpages) を使用することができます :

hpages は横方向のページ数を示すコマンドであり、そして vpages は縦方向のページ数を示すコマンドです。

特定のスキンパラメータ設定を使用して、分割されたページに罫線を配置することもできます (例を参照)。

```
@startuml
' Split into 4 pages
page 2x2
```

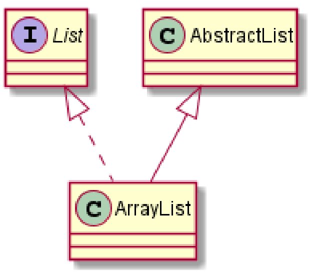
```
skinparam pageMargin 10
skinparam pageExternalColor gray
skinparam pageBorderColor black

class BaseClass

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

    .BaseClass <|- Meeting

}

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

  net.dummy.Meeting o-- Person
}

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



3.33 継承 (extends) と実装 (implements)

extends キーワードと implements キーワードを使用することができます。

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



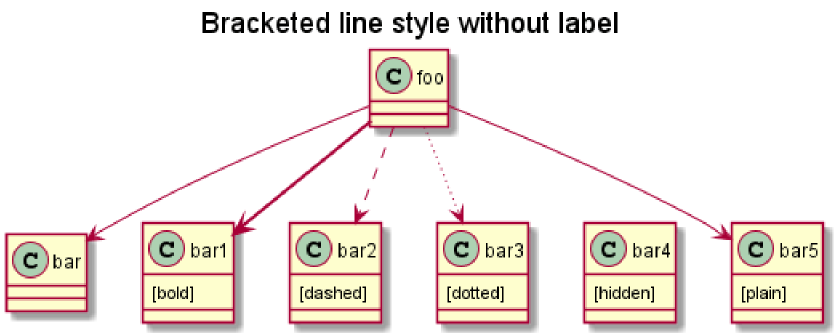
3.34 角括弧を使用した関係（リンク、矢印）のスタイル

3.34.1 線のスタイル

関係（リンク、矢印）に bold、dashed、dotted、hidden、plain のスタイルを指定することができます。

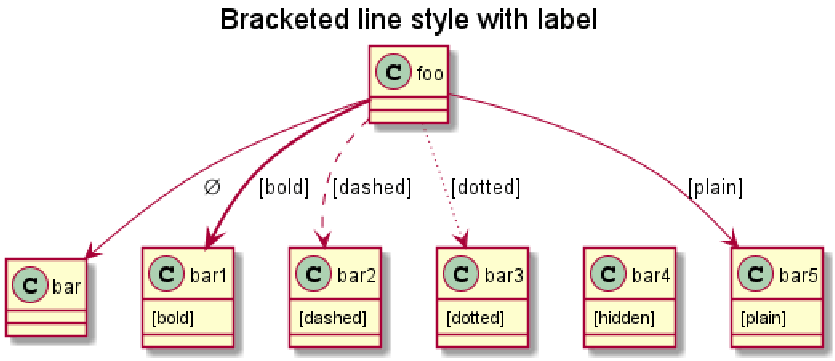
- ラベル無し

```
@startuml
title Bracketed line style without label
class foo
class bar
bar1 : [bold]
bar2 : [dashed]
bar3 : [dotted]
bar4 : [hidden]
bar5 : [plain]

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



- ラベル有り

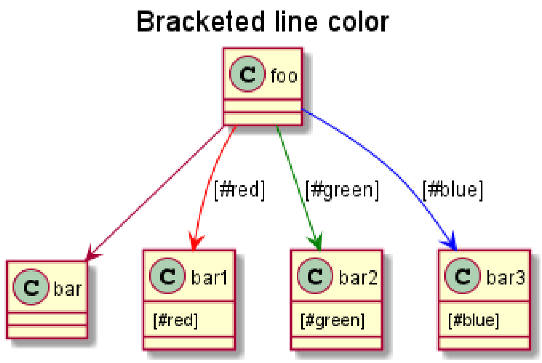
```
@startuml
title Bracketed line style with label
class foo
class bar
bar1 : [bold]
bar2 : [dashed]
bar3 : [dotted]
bar4 : [hidden]
bar5 : [plain]

foo --> bar          :
foo -[bold]-> bar1   : [bold]
foo -[dashed]-> bar2 : [dashed]
foo -[dotted]-> bar3 : [dotted]
foo -[hidden]-> bar4 : [hidden]
foo -[plain]-> bar5  : [plain]

@enduml
```



[Adapted from QA-4181]

3.34.2 線の色

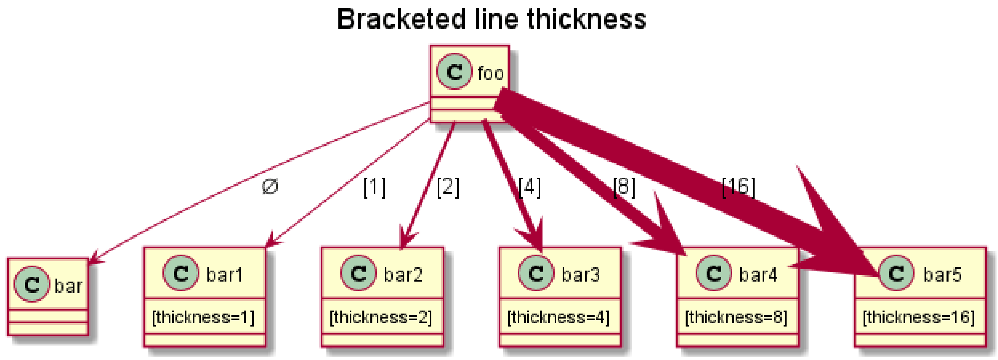
```
@startuml
title Bracketed line color
class foo
class bar
bar1 : [#red]
bar2 : [#green]
bar3 : [#blue]

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



3.34.3 線の太さ

```
@startuml
title Bracketed line thickness
class foo
class bar
bar1 : [thickness=1]
bar2 : [thickness=2]
bar3 : [thickness=4]
bar4 : [thickness=8]
bar5 : [thickness=16]

foo --> bar                 :
foo -[thickness=1]-> bar1   : [1]
foo -[thickness=2]-> bar2   : [2]
foo -[thickness=4]-> bar3   : [4]
```

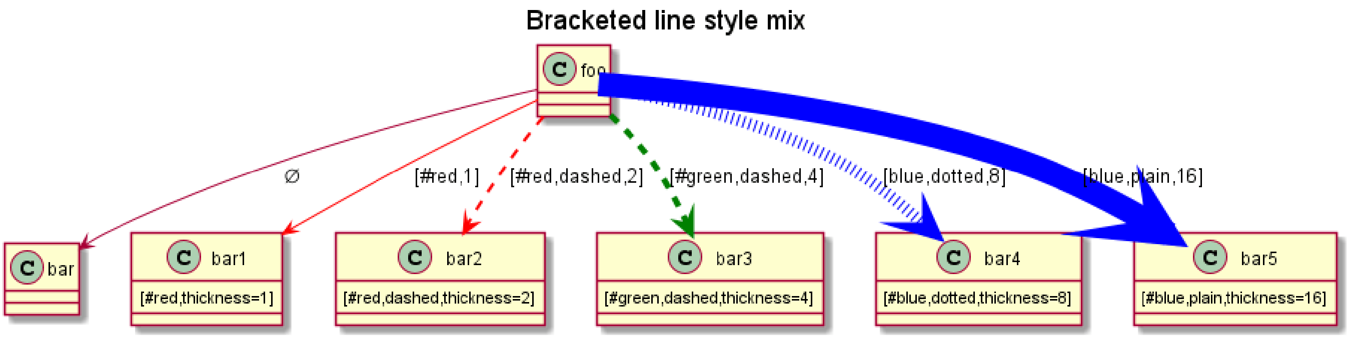
```
foo -[thickness=8]-> bar4   : [8]
foo -[thickness=16]-> bar5  : [16]

@enduml
```



[Ref. QA-4949]

3.34.4 混合

```
@startuml
title Bracketed line style mix
class foo
class bar
bar1 : [#red,thickness=1]
bar2 : [#red,dashed,thickness=2]
bar3 : [#green,dashed,thickness=4]
bar4 : [#blue,dotted,thickness=8]
bar5 : [#blue,plain,thickness=16]

foo --> bar                             :
foo -[#red,thickness=1]-> bar1          : [#red,1]
foo -[#red,dashed,thickness=2]-> bar2   : [#red,dashed,2]
foo -[#green,dashed,thickness=4]-> bar3 : [#green,dashed,4]
foo -[#blue,dotted,thickness=8]-> bar4  : [blue,dotted,8]
foo -[#blue,plain,thickness=16]-> bar5  : [blue,plain,16]
@enduml
```



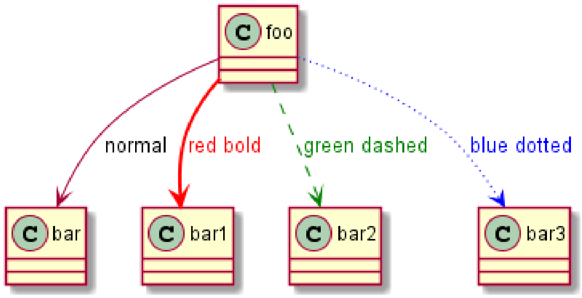
3.35 関係 (リンク、矢印) の色とスタイルを変更する (インラインスタイル)

個別の関係ごとに色とスタイルを変更するには、次の記法を使用します :

- `#color;line.[bold|dashed|dotted];text:color`

```
@startuml
class foo
foo --> bar : normal
```

```
foo --> bar1 #line:red;line.bold;text:red  : red bold
foo --> bar2 #green;line.dashed;text:green : green dashed
foo --> bar3 #blue;line.dotted;text:blue   : blue dotted
@enduml
```



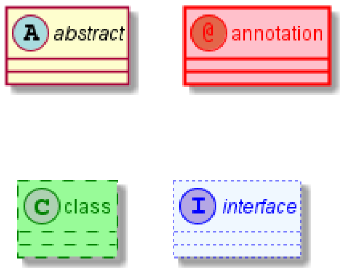
[See similar feature on deployment]

3.36 クラスの色とスタイルを変更する（インラインスタイル）

個別のクラスごとに色とスタイルを変更するには、次の記法を使用します：

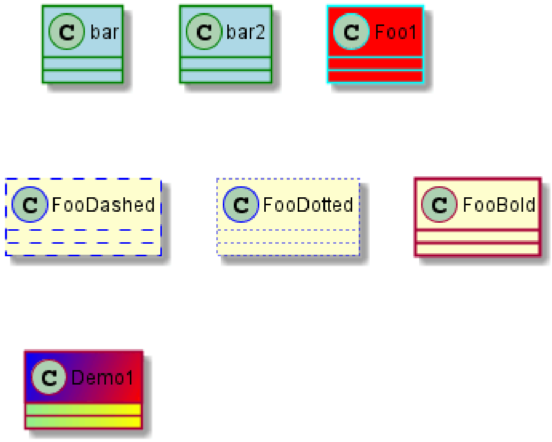
- `#[color|back:color];header:color;line:color;line.[bold|dashed|dotted];text:color`

```
@startuml
abstract   abstract
annotation annotation #pink;line:red;line.bold;text:red
class      class      #palegreen;line:green;line.dashed;text:green
interface  interface  #aliceblue;line:blue;line.dotted;text:blue
@enduml
```



例：

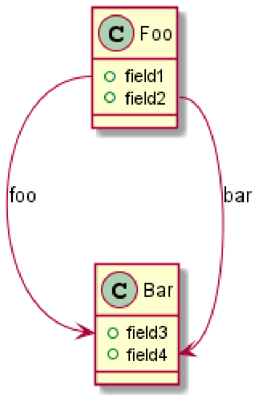
```
@startuml
class bar #line:green;back:lightblue
class bar2 #lightblue;line:green

class Foo1 #back:red;line:00FFFF
class FooDashed #line.dashed:blue
class FooDotted #line.dotted:blue
class FooBold #line.bold
class Demo1 #back:lightgreen|yellow;header:blue/red
@enduml
```



[Ref. QA-3770]

3.37 クラスメンバ間の矢印

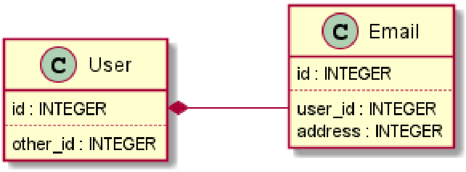
```
@startuml
class Foo {
+ field1
+ field2
}

class Bar {
+ field3
+ field4
}

Foo::field1 --> Bar::field3 : foo
Foo::field2 --> Bar::field4 : bar
@enduml
```



[Ref. QA-3636]

```
@startuml
left to right direction

class User {
  id : INTEGER
  ..
  other_id : INTEGER
}

class Email {
```

```
  id : INTEGER
  ..
  user_id : INTEGER
  address : INTEGER
}

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



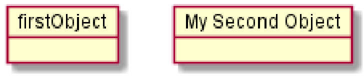
[Ref. QA-5261]

4 オブジェクト図

4.1 オブジェクトの定義

オブジェクトのインスタンスを、キーワード `object` を使用して定義します。

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



4.2 オブジェクト間の関係

オブジェクト間の関係は次の記号を用いて定義します:

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

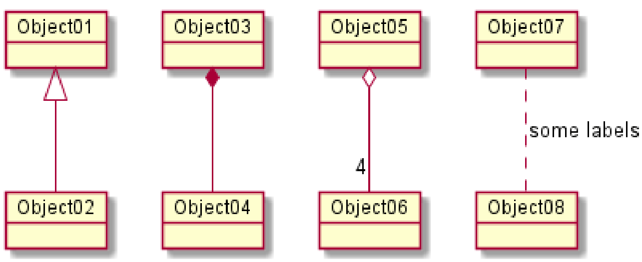
`--` を `..` に置き換えることで点線を示すことができます。

これらのルールを知ることで、以下の図を描くことができます。

関係にラベルをつけることができ、`:` を用い、ラベルの文字列を続けます。

関係の各側のスペースを含む文字列を引用符 `""` で囲むことができます。

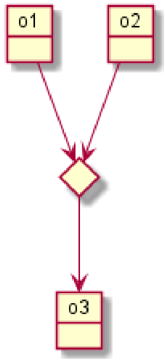
```
@startuml
object Object01
object Object02
object Object03
object Object04
object Object05
object Object06
object Object07
object Object08

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



4.3 n-項関連

```
@startuml
object o1
object o2
diamond dia
```

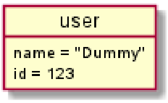
```
object o3

o1  --> dia
o2  --> dia
dia --> o3
@enduml
```



4.4 フィールドの追加

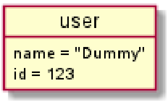
フィールドを宣言するには、シンボル : にフィールド名を続けます。

```
@startuml

object user

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

@enduml
```



全てのフィールドを括弧 {} で括って範囲を示すことも可能です。

```
@startuml

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

@enduml
```



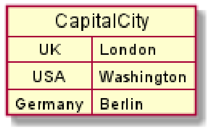
4.5 クラス図と共通の機能

- 属性、メソッド等の非表示
- 注釈の詳細
- 非文字の使用
- 化粧をする

4.6 マップテーブル(連想配列)

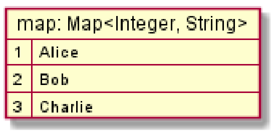
map キーワードとセパレータ => を使って、マップテーブル(連想配列)を定義することができます。

```
@startuml
map CapitalCity {
 UK => London
 USA => Washington
 Germany => Berlin
}
@enduml
```



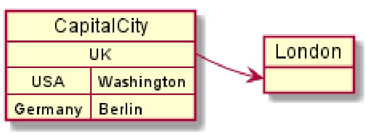

```
@startuml
map "Map **Contry => CapitalCity**" as CC {
 UK => London
 USA => Washington
 Germany => Berlin
}
@enduml
```


```
@startuml
map "map: Map<Integer, String>" as users {
 1 => Alice
 2 => Bob
 3 => Charlie
}
@enduml
```



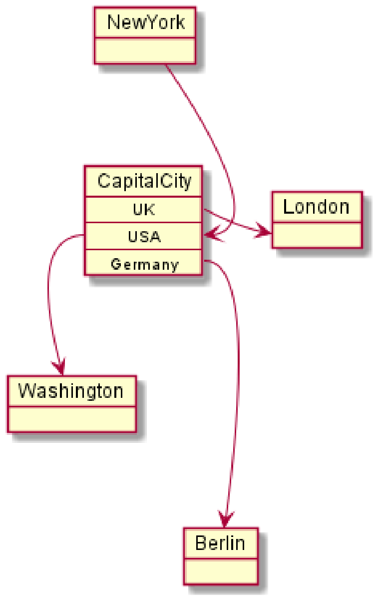
オブジェクトにリンクを追加します。

```
@startuml
object London

map CapitalCity {
 UK *-> London
 USA => Washington
 Germany => Berlin
}
@enduml
```




```
@startuml
object London
```

```
object Washington
object Berlin
object NewYork

map CapitalCity {
 UK *-> London
 USA *--> Washington
 Germany *---> Berlin
}

NewYork --> CapitalCity::USA
@enduml
```



[Ref. #307]

5 アクティビティ図 (レガシー版)

これはアクティビティ図の古い記法です。現行の新しいバージョンは、アクティビティ図を参照してください。

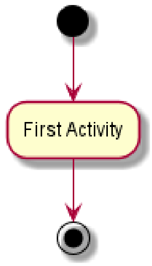
5.1 単純なアクティビティ

`(*)` をアクティビティ図の開始点と終了点に使います。

場合によっては、`(*top)` を使用して開始点を図の一番上に置くこともできます。

`-->` で矢印を表します。

```
@startuml

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

@enduml
```

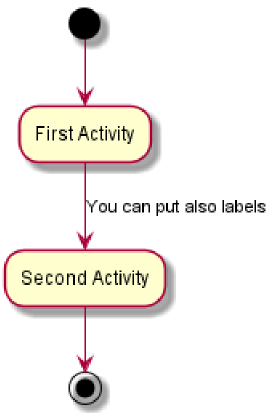


5.2 矢印のラベル

デフォルトで、矢印は最後に書いたアクティビティを起点に描かれます。

矢印にラベルを付けるには、矢印の定義の直後に角括弧 `[`と`]` を使います。

```
@startuml

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

@enduml
```



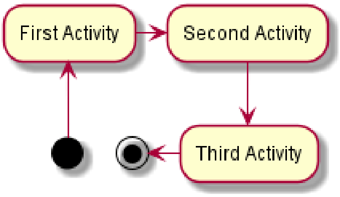
5.3 矢印の方向を変える

水平矢印には `->` を使用できます。次の構文を使用して矢印の方向を強制することができます。

- `-down->` (デフォルトの矢印)

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

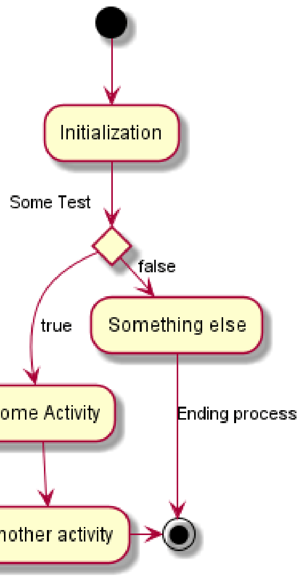
```
@startuml

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

@enduml
```



5.4 分岐

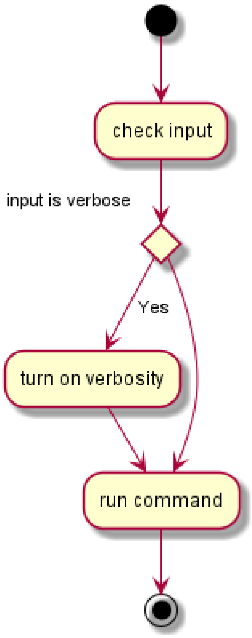
キーワード if/then/else を使用してブランチを定義することができます。

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

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

@enduml
```



残念ながら、図のテキストで同じアクティビティを繰り返すことがあります：

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



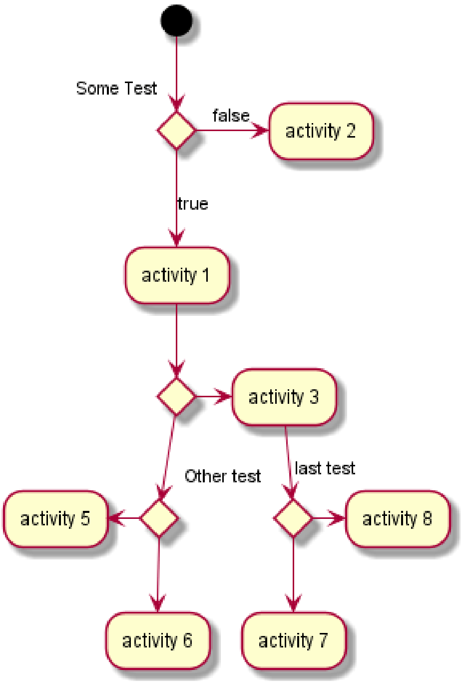
5.5 もっと分岐

デフォルトでは、分岐は最後に定義されたアクティビティに接続されますが、これを上書きしてキーワード if でリンクを定義することは可能です。

分岐をネストすることも可能です。

```
@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
```

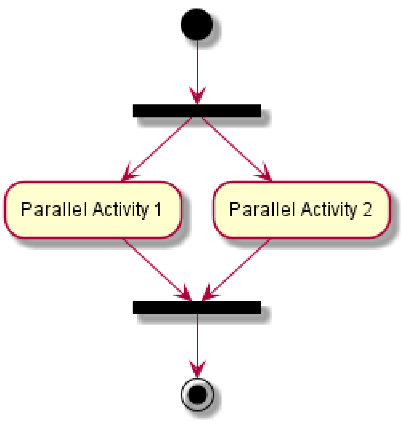
```
a3 --> if "last test" then
  --> "activity 7"
else
  -> "activity 8"
endif

@enduml
```



5.6 同期

=== code === を使用して同期バーを表示できます。

```
@startuml

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

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

--> (*)

@enduml
```

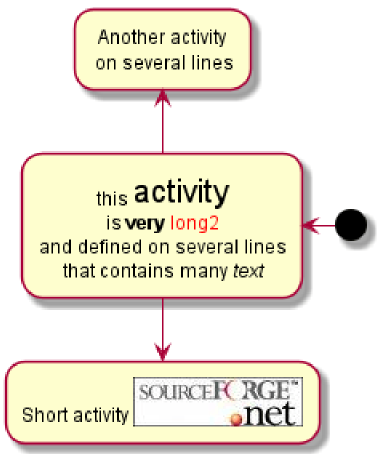


5.7 長いアクティビティの記述

アクティビティを宣言するとき、説明文を複数の行にまたがせることができます。説明に を追加することもできます。

キーワード as を使ってアクティビティに短いコードを与えることもできます。このコードは、図の説明の後半で使用できます。

```
@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
```



5.8 注釈

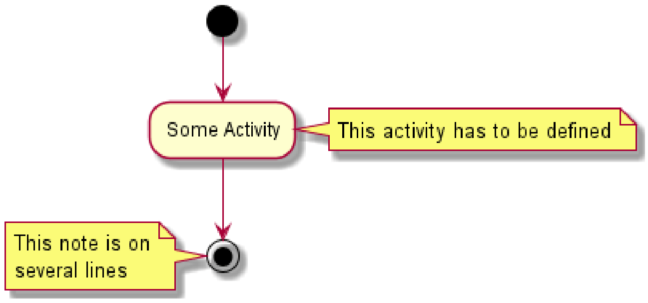
注釈をつけるアクティビティの説明の直後にあるコマンド note left, note right, note top or note bottom, を使用して、アクティビティに注釈を追加することができます。

開始点に注釈を付ける場合は、図の説明の最初に注釈を定義します。

キーワード endnote を使用して、複数の行に注釈を付けることもできます。

```
@startuml

(*) --> "Some Activity"
```

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

@enduml
```



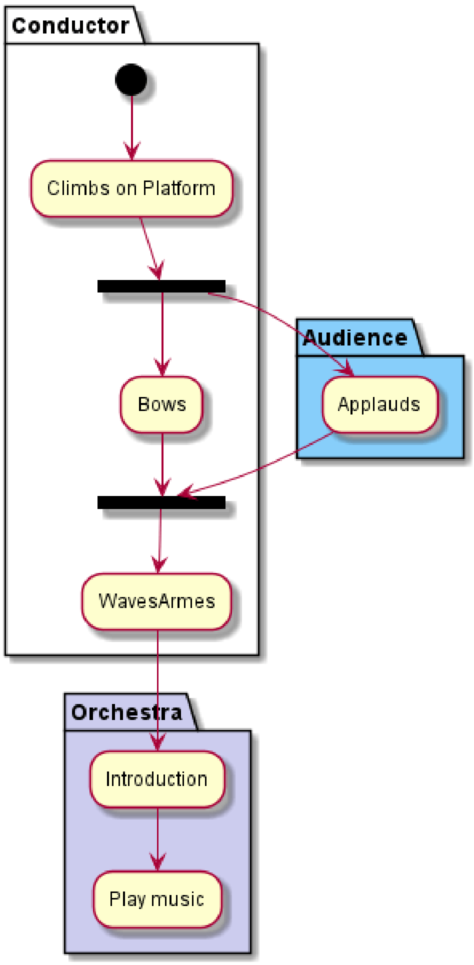
5.9 パーティション

キーワード partition を使用してパーティションを定義し、必要に応じてパーティションの背景色を宣言することができます（HTML カラーコードまたは名前を使用）。

アクティビティを宣言すると、自動的に最後に使用されたパーティションに配置されます。

閉じ括弧 } を使用してパーティション定義を閉じることができます。

```
@startuml

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

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

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

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

@enduml
```



5.10 スキンパラメータ

コマンド skinparam を使用して、図面の色とフォントを変更することができます。

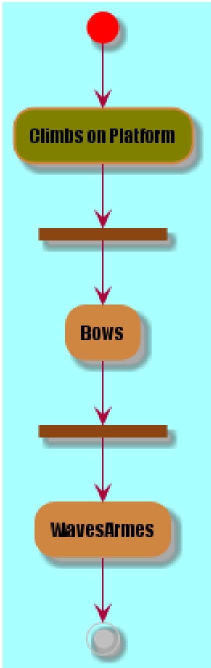
このコマンドを使用することができます :

- 図の定義中では、他のコマンドと同様に、
- インクルードされたファイルの中で、
- コマンドラインまたは ANT タスクで提供される構成ファイルの中で。

定型アクティビティには、特定の色とフォントを定義できます。

```
@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
```

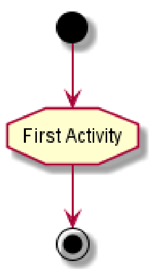
```
--> === S2 ===
--> WavesArmes
--> (*)

@enduml
```



5.11 八角形

コマンド skinparam activityShape octagon を使用して、アクティビティの形状を八角形に変更できます。

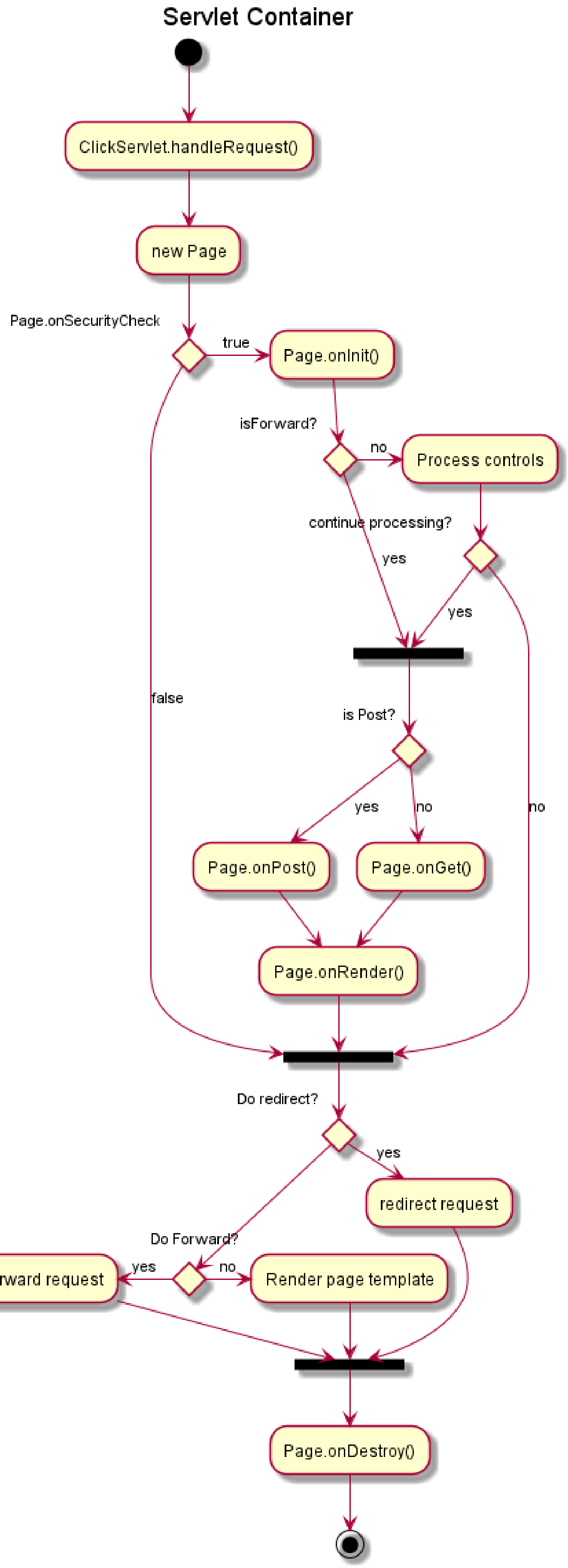
```
@startuml
'Default is skinparam activityShape roundBox
skinparam activityShape octagon

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

@enduml
```



5.12 完全な例

```
@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
Do Forward?
yes
no
Forward request
Render page template
Page.onDestroy()

6 アクティビティ図（ベータ版）

アクティビティ図の古い構文には、メンテナンスが難しいなど、いくつかの制限と欠点がありました。

そのため、書式や構文をよりよく定義できるように、ベータ版として全く新しい構文と実装が提案されています（V7947 以降）。

この新しい実装には、（シーケンス図と同様に）Graphviz パッケージのインストールを必要としないという利点もあります。

将来的に古い構文は新しい構文に置換されるでしょう。しかし、上位互換性が確保され、古い構文もそのまま認識可能となる予定です。

新しい構文へ移行することが強く推奨されています。

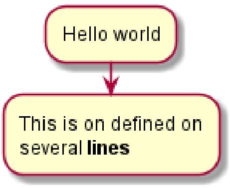
6.1 単純なアクティビティ

アクティビティのラベルは: で開始し; で終了します。

テキストの書式設定は、Creole 記法の Wiki 構文を使用して行うことができます。

それらは定義順に暗黙的にリンクされます。

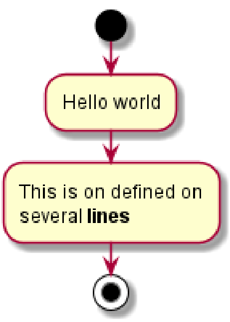
```
@startuml
:Hello world;
:This is on defined on
several **lines**;
@enduml
```



6.2 開始／終了

図の開始と終了を示すために、キーワード `start` と `stop` を使用できます。

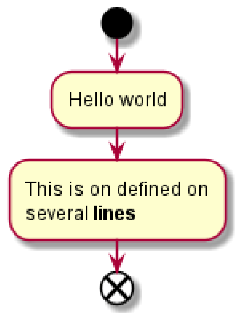
```
@startuml
start
:Hello world;
:This is on defined on
several **lines**;
stop
@enduml
```



キーワード `end` もまた使用できます。

```
@startuml
start
```

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



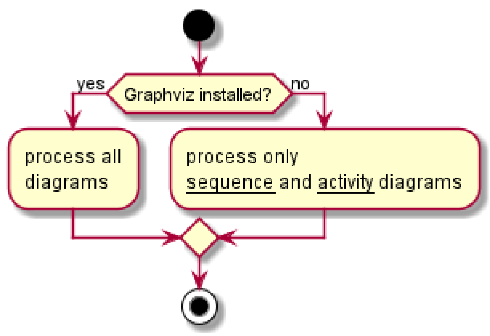
6.3 条件文

図に条件分岐を追加したい場合は、キーワード `if`、`then` そして `else` を使用することができます。ラベルは括弧を使用することで与えることができます。

3 種類の構文を使うことができます。

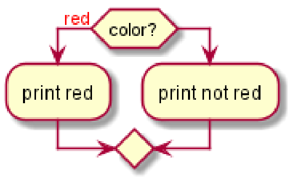
- `if (...) then (...)`

```
@startuml

start

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

stop

@enduml
```



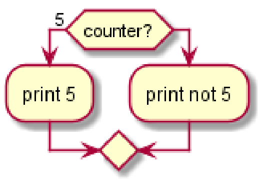
- `if (...) is (...) then`

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



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

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

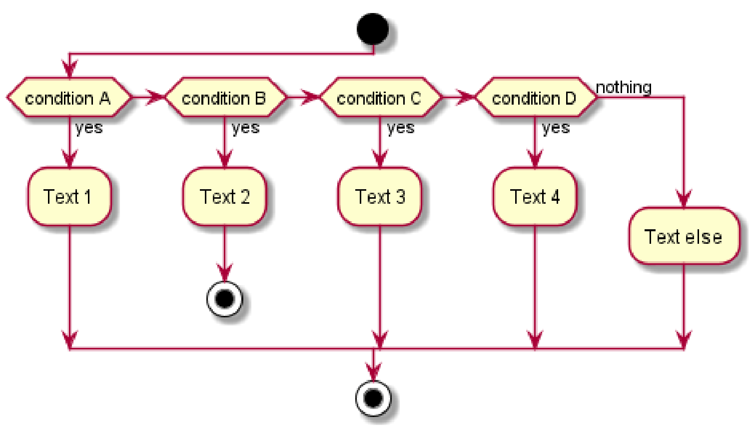


[Ref. QA-301]

6.3.1 複数条件 (水平モード)

いくつもの条件分岐がある場合には、キーワード `elseif` を使用できます。(デフォルトで水平モードになります):

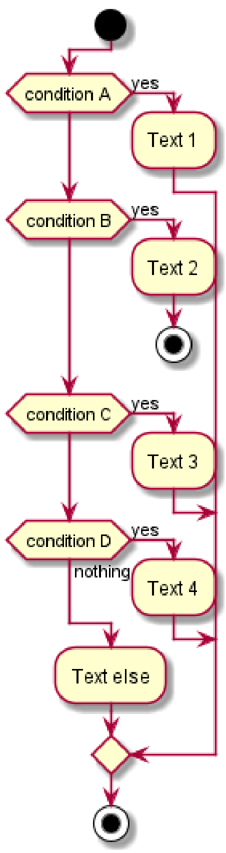
```
@startuml
start
if (condition A) then (yes)
  :Text 1;
elseif (condition B) then (yes)
  :Text 2;
  stop
elseif (condition C) then (yes)
  :Text 3;
elseif (condition D) then (yes)
  :Text 4;
else (nothing)
  :Text else;
endif
stop
@enduml
```



6.3.2 複数条件 (垂直モード)

!pragma useVerticalIf on コマンドを使用すると、垂直モードの分岐になります:

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



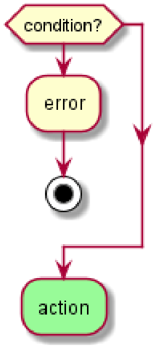
[Ref. QA-3931]

6.4 アクションの停止を伴う条件文 [kill, detach]

if 節内でアクションを停止できます。

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

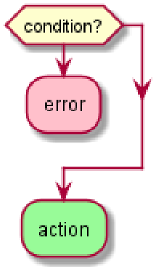
```
endif
#palegreen:action;
@enduml
```



ただし、明確なアクションで停止したい場合は、キーワード「kill」または「detach」を使用できます:

- kill

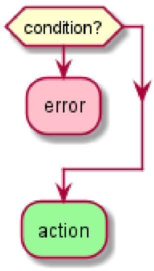
```
@startuml
if (condition?) then
  #pink:error;
  kill
endif
#palegreen:action;
@enduml
```



[Ref. QA-265]

- detach

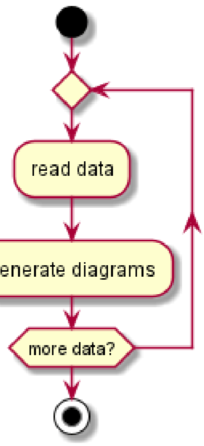
```
@startuml
if (condition?) then
  #pink:error;
  detach
endif
#palegreen:action;
@enduml
```



6.5 繰り返し(後判定)

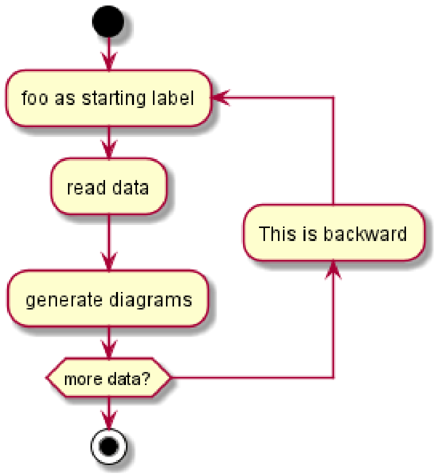
繰り返し処理(後判定)がある場合には、キーワード repeat と repeat while を使用できます。

```
@startuml

start

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

stop

@enduml
```



アクティビティを repeat の戻り先にすることもできます。また、backward キーワードを使用して、戻りのパスにアクティビティを挿入することもできます。

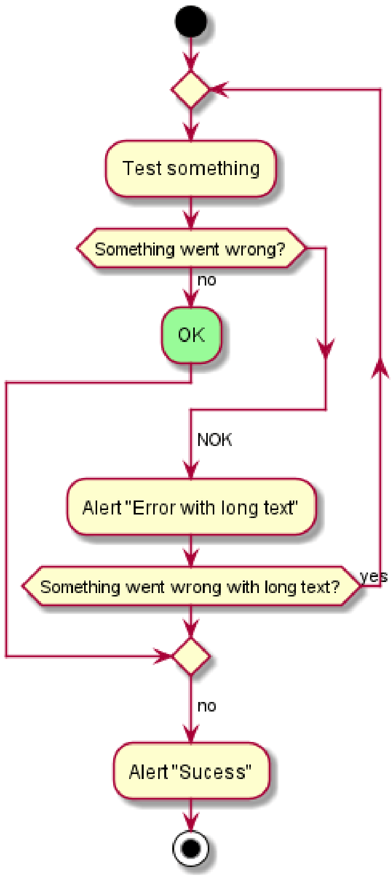
```
@startuml

start

repeat :foo as starting label;
  :read data;
  :generate diagrams;
backward:This is backward;
repeat while (more data?)

stop

@enduml
```



6.6 repeat 節を中断する [break]

ループ内でアクションを中断することができます。

```
@startuml
start
repeat
  :Test something;
    if (Something went wrong?) then (no)
      #palegreen:OK;
      break
    endif
    ->NOK;
    :Alert "Error with long text";
repeat while (Something went wrong with long text?) is (yes)
->no;
:Alert "Sucess";
stop
@enduml
```

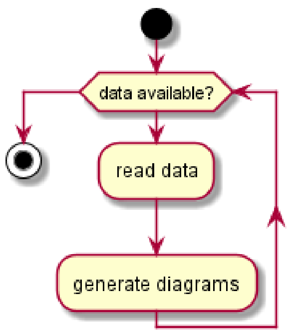


[Ref. QA-6105]

6.7 繰り返し（前判定）

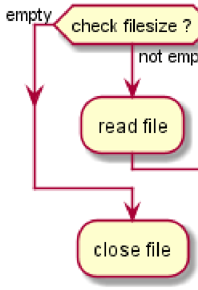
繰り返し処理（前判定）がある場合には、キーワード `while` と `end while` を使用できます。

```
@startuml

start

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

stop

@enduml
```



キーワード endwhile の後ろ、または、キーワード is を使用することで、ラベルを与えることができます。

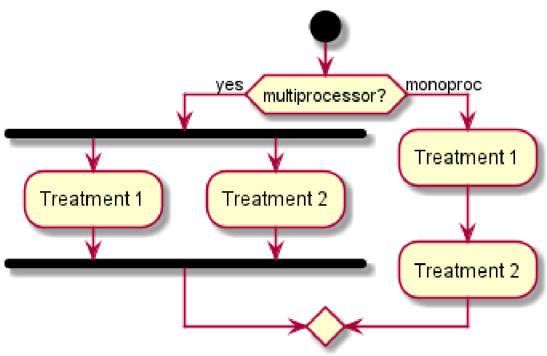
```
@startuml
while (check filesize ?) is (not empty)
  :read file;
endwhile (empty)
:close file;
@enduml
```



6.8 並列処理

並列処理を示すために、キーワード fork、fork again そして end fork が使用できます。

```
@startuml

start

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

@enduml
```



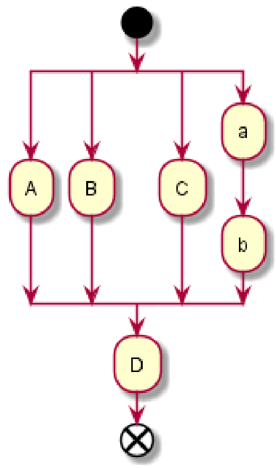
6.9 処理の分岐

6.9.1 Split

split、split again、end split キーワードを使って、プロセスの分岐を表すことができます。

```
@startuml
```

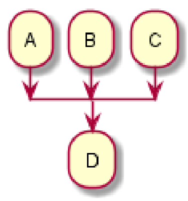
```
start
split
   :A;
split again
   :B;
split again
   :C;
split again
   :a;
   :b;
end split
:D;
end
@enduml
```



6.9.2 入力の分岐 (複数開始)

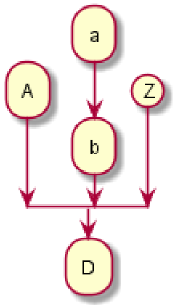
入力の分岐を表現するには、hidden で矢印を隠します。

```
@startuml
split
   -[hidden]->
   :A;
split again
   -[hidden]->
   :B;
split again
   -[hidden]->
   :C;
end split
:D;
@enduml
```




```
@startuml
split
```

```
    -[hidden]->
    :A;
split again
    -[hidden]->
    :a;
    :b;
split again
    -[hidden]->
    (Z)
end split
:D;
@enduml
```

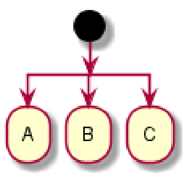


[Ref. QA-8662]

6.9.3 出力の分岐 (複数終了)

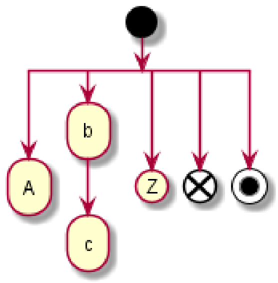
出力の分岐を表現するには、kill または detach を使用します。

```
@startuml
start
split
    :A;
    kill
split again
    :B;
    detach
split again
    :C;
    kill
end split
@enduml
```




```
@startuml
start
split
    :A;
    kill
split again
    :b;
    :c;
```

```
    detach
split again
    (Z)
    detach
split again
    end
split again
    stop
end split
@enduml
```

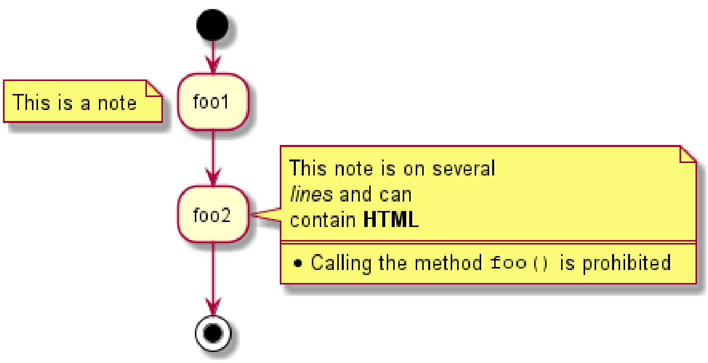


6.10 注釈

Creole 表記の Wiki 構文を使用することで、テキストの書式設定ができます。

キーワード floating を使用し、注釈を遊離させることもできます。

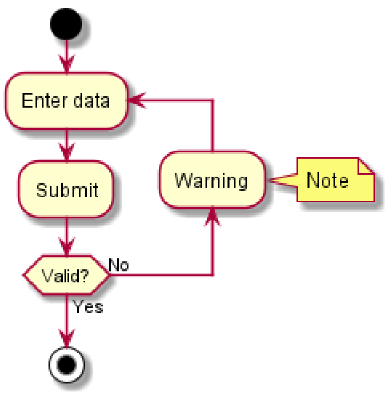
```
@startuml

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
```



戻り方向 (backward) のアクティビティに注釈をつけることもできます。

```
@startuml
start
```

```
repeat :Enter data;
:Submit;
backward :Warning;
note right: Note
repeat while (Valid?) is (No) not (Yes)
stop
@enduml
```

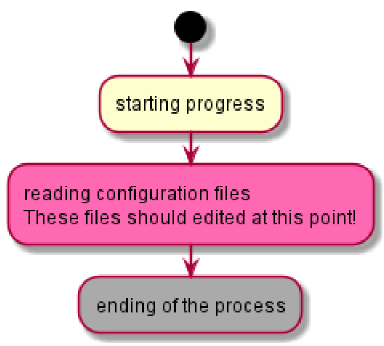


[Ref. QA-11788]

6.11 色指定

各アクティビティに、色を指定することができます。

```
@startuml

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

@enduml
```

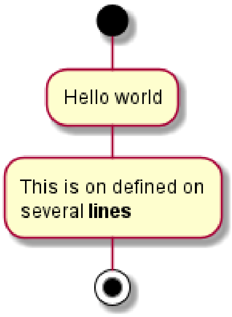


6.12 矢印無しの線

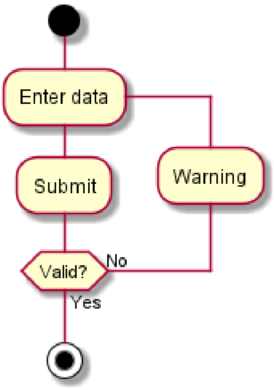
skinparam ArrowHeadColor none を指定すると、アクティビティの接続線を矢印無しにすることができます。

```
@startuml
skinparam ArrowHeadColor none
start
:Hello world;
:This is on defined on
several **lines**;
```

```
stop
@enduml
```




```
@startuml
skinparam ArrowHeadColor none
start
repeat :Enter data;
:Submit;
backward :Warning;
repeat while (Valid?) is (No) not (Yes)
stop
@enduml
```



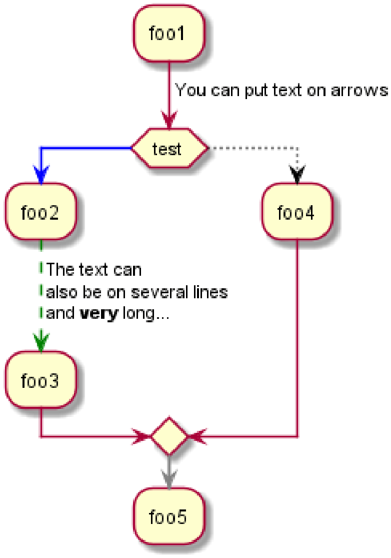
6.13 矢印

記号-> を用いて、矢印にテキストを添えることができ、また、色を変えることもできます。

点線、破線、太線、または、矢印なし、もまた可能です。

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

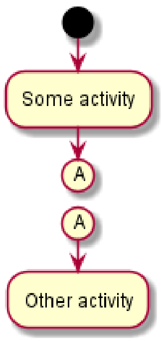
```
@enduml
```



6.14 コネクタ

半角括弧を使用して、コネクタを記述することができます。

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

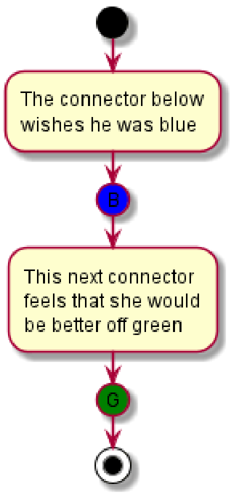


6.15 コネクタの色

コネクタに色を設定することができます。

```
@startuml
start
:The connector below
wishes he was blue;
#blue:(B)
:This next connector
feels that she would
be better off green;
#green:(G)
stop
```

```
@enduml
```

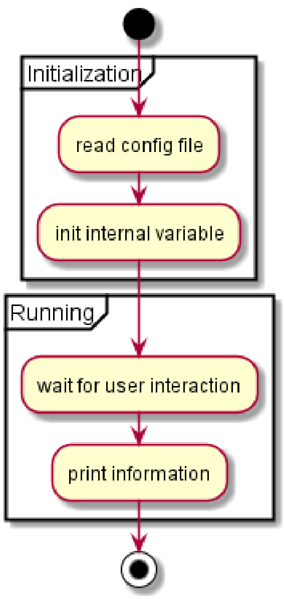


[Ref. QA-10077]

6.16 グループ化 (パーティション)

partition を定義して、複数のアクティビティをグループ化できます:

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

stop
@enduml
```

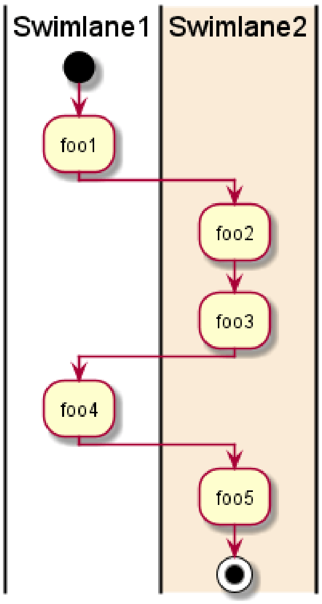


6.17 スイムレーン

パイプ記号 | を用いて、複数のスイムレーンを定義することができます。

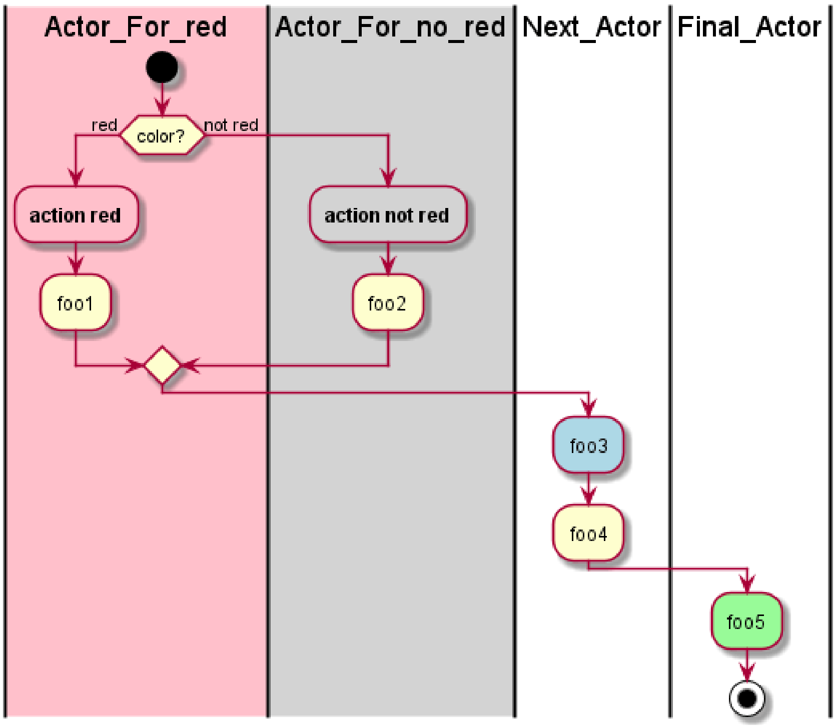
さらに、スイムレーン毎に色を変えることができます。

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



スイムレーンの中で、if 条件文や repeat、while のループを使用できます。

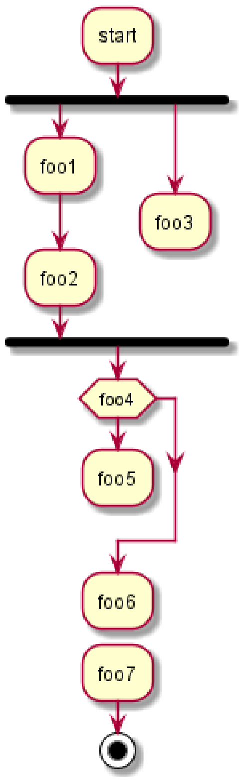
```
@startuml
|#pink|Actor_For_red|
start
if (color?) is (red) then
#pink:**action red**;
:foo1;
else (not red)
|#lightgray|Actor_For_no_red|
#lightgray:**action not red**;
:foo2;
endif
|Next_Actor|
#lightblue:foo3;
:foo4;
|Final_Actor|
#palegreen:foo5;
stop
@enduml
```



6.18 矢印の除去 (detach, kill)

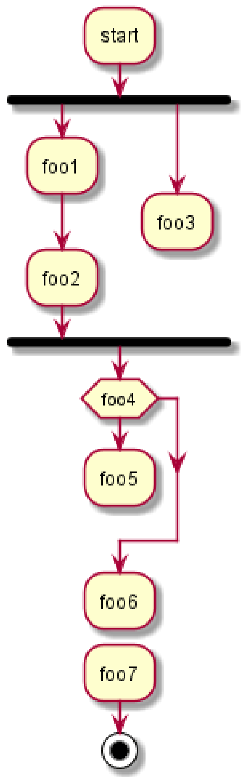
キーワード detach または kill を使用して、矢印を取り除くことができます。

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



- kill

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



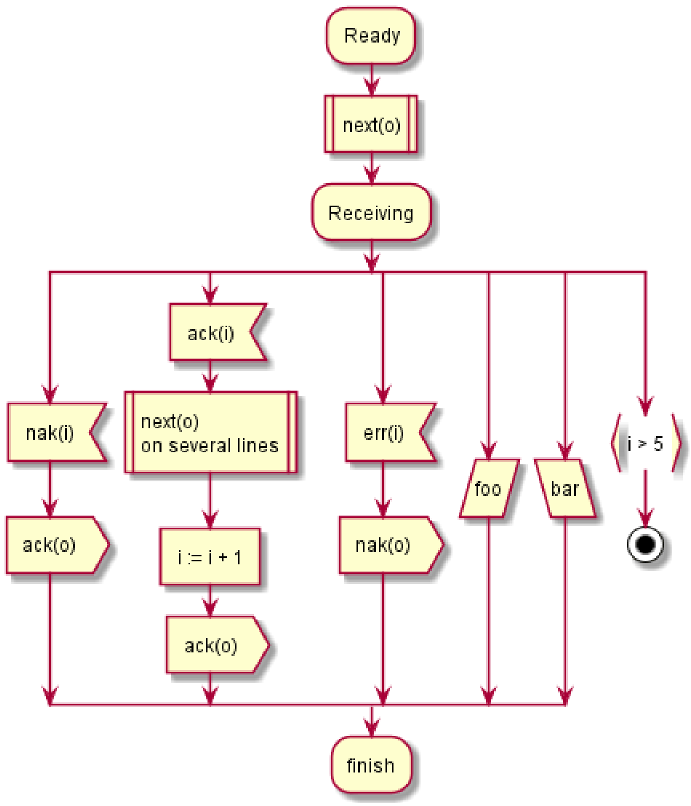
6.19 SDL 図

終端記号; を置き換えることで、アクティビティの表現形式を変えることができます :

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

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

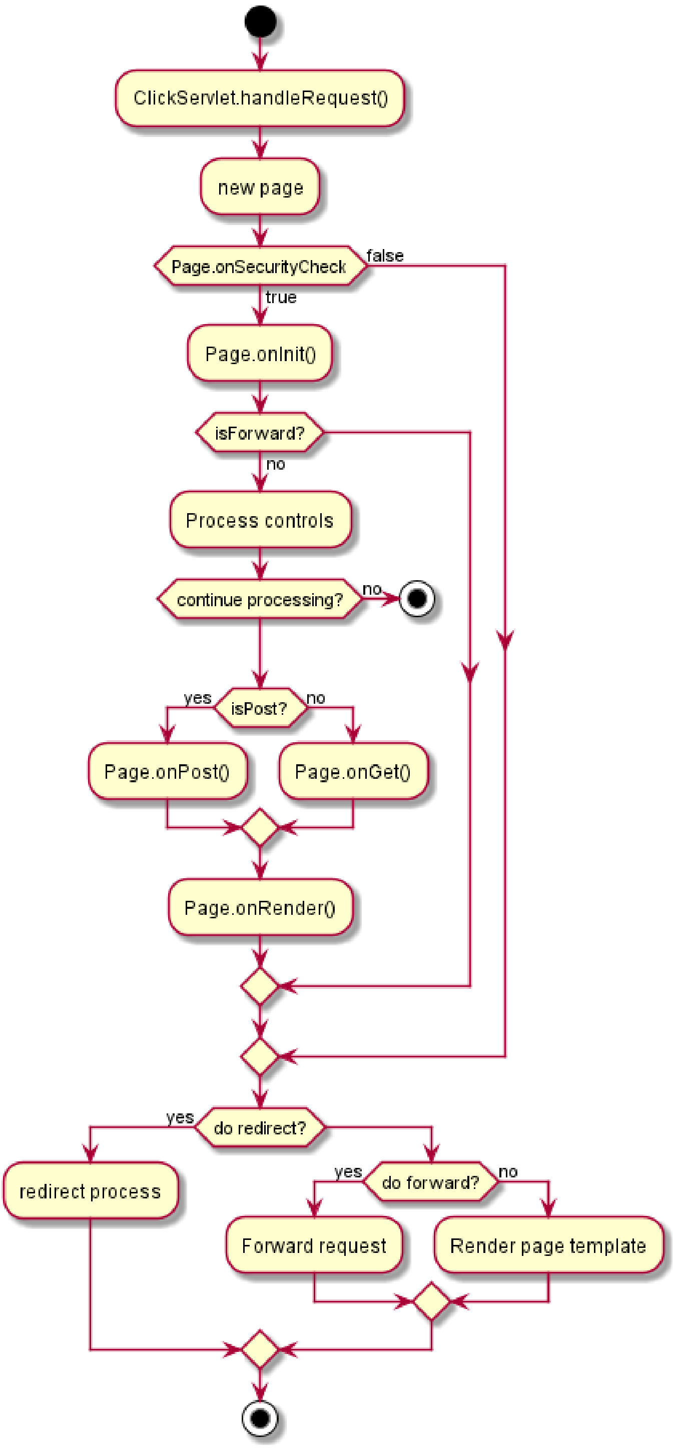
```
split again
 :bar\\
split again
 :i > 5}
stop
end split
:finish;
@enduml
```



6.20 完全な例

```
@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
```

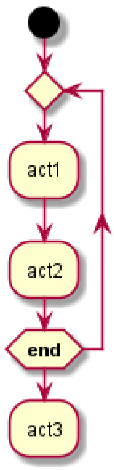


6.21　条件のスタイル

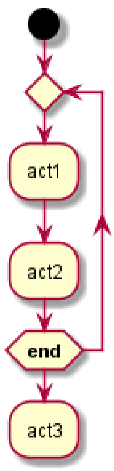
6.21.1　inside スタイル (デフォルト)

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

```
@enduml
```

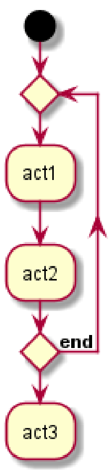



```
@startuml
start
repeat
  :act1;
  :act2;
repeatwhile (<b>end)
:act3;
@enduml
```



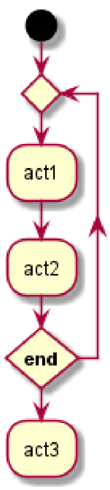
6.21.2 diamond スタイル

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

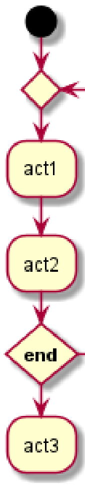


6.21.3 InsideDiamond (または *foo1*) スタイル

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




```
@startuml
skinparam conditionStyle foo1
start
repeat
  :act1;
  :act2;
repeatwhile (<b>end)
:act3;
@enduml
```



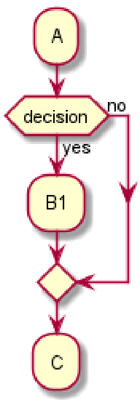
[Ref. QA-1290 and #400]

6.22 条件終了のスタイル

6.22.1 diamond スタイル (デフォルト)

- 分岐が 1 つの場合

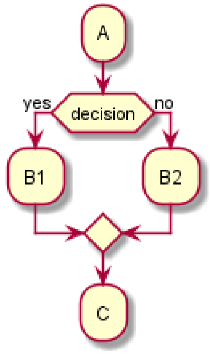
```
@startuml
skinparam ConditionEndStyle diamond
:A;
if (decision) then (yes)
    :B1;
else (no)
endif
:C;
@enduml
```



- 分岐が 2 つ (B1, B2) の場合

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

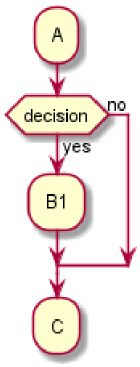
```
@enduml
```



6.22.2 水平ライン (hline) スタイル

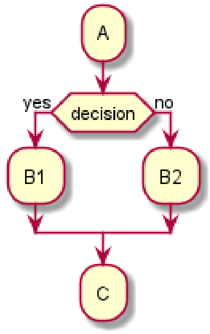
- 分岐が 1 つの場合

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



- 分岐が 2 つ (B1, B2) の場合

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



[Ref. QA-4015]

7 コンポーネント図

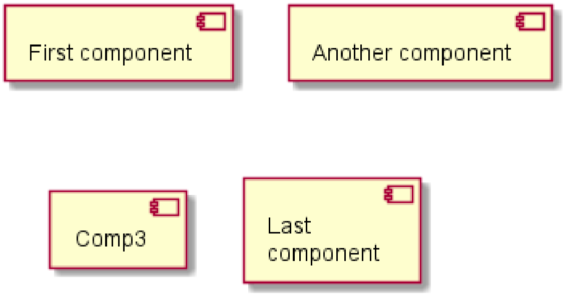
いくつかの例をご覧ください。

7.1 コンポーネント

コンポーネントは括弧でくくります。

また、component キーワードでもコンポーネントを定義できます。そして、コンポーネントには as キーワードにより別名をつけることができます。この別名は、後でリレーションを定義するときに使えます。

```
@startuml

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

@enduml
```



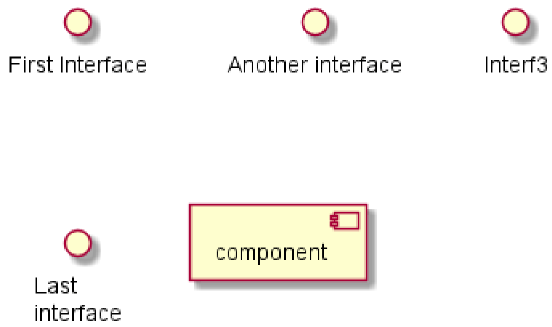
7.2 インタフェース

インタフェースは丸括弧 () でシンボルを囲うことで定義できます。(何故なら見た目が丸いからです。)

もちろん interface キーワードを使って定義することもできます。as キーワードでエイリアスを定義できます。このエイリアスは後で、関係を定義する時に使えます。

後で説明されますが、インタフェースの定義は省略可能です。

```
@startuml

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

[component]
footer //Adding "component" to force diagram to be a **component diagram**//
@enduml
```

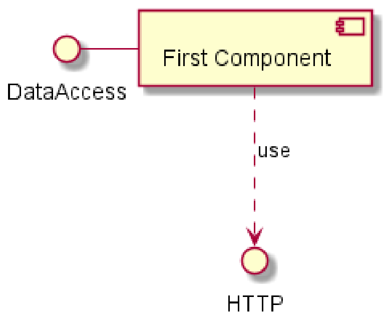


Adding "component" to force diagram to be a component diagram

7.3 基本的な例

要素間の関係は、破線 (..)、直線 (--), 矢印 (-->) の組合せで構成されます。

```
@startuml

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

@enduml
```

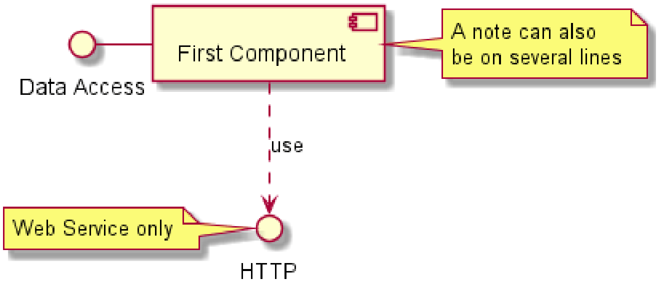


7.4 ノートの使用方法

オブジェクトに関連のあるノートを作成するには `note left of` 、`note right of` 、`note top of` 、`note bottom of` キーワードを使います。`note left of` , `note right of` , `note top of` , `note bottom of`

または `note` キーワードを使ってノートを作成し、`..` 記号を使ってオブジェクトに紐づけることができます。

```
@startuml

interface "Data Access" as DA

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

note left of HTTP : Web Service only

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

@enduml
```

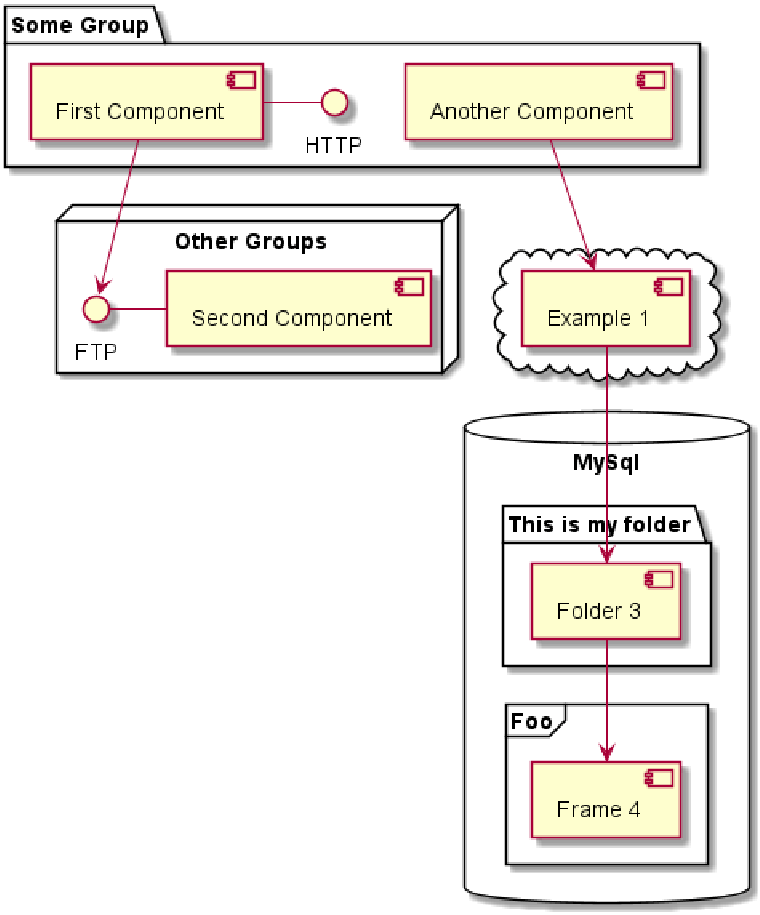


7.5 コンポーネントのグループ化

いくつかのキーワードをグループコンポーネントやインタフェースに使用することができます：

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

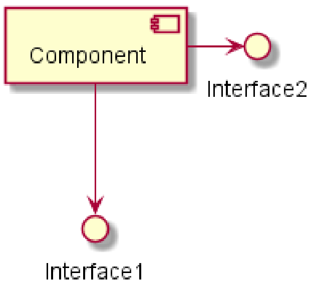
```
@startuml

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

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

cloud {
  [Example 1]
}


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


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

@enduml
```



7.6 矢印の方向を変える

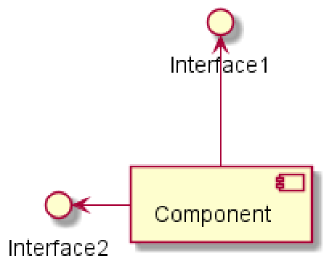
デフォルトではクラス間のリンクは 2 つのダッシュ -- を持っており垂直方向に配向されています。次のように単一のダッシュ（またはドット）を置くことによって水平方向のリンクを使用することが可能です：

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



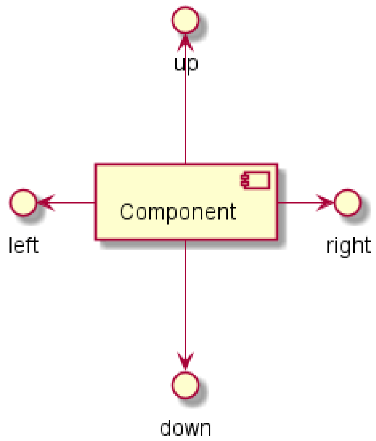
リンクを反対にすることで方向を変更することもできます。

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



また、left を加えることで矢印の向きを変更することもできます。right, up, down などのキーワードを矢印の間に記述します。:

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

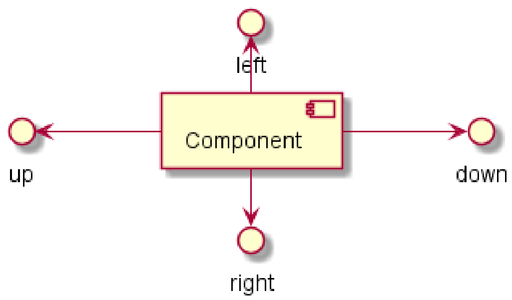


方向の最初の文字のみを使用して矢印を短くすることができます (例えば、-down- の代わりに-d-)、または最初の 2 文字 (-do-)。

この機能を悪用してはならないことに注意してください : *Graphviz* は微調整の必要がない良い結果を通常は与えてくれます。

left to right direction パラメータを指定した場合は、次のようになります。:

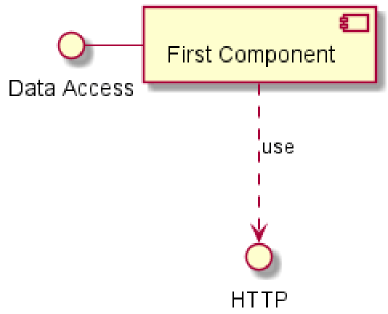
```
@startuml
left to right direction
[Component] -left-> left
[Component] -right-> right
[Component] -up-> up
[Component] -down-> down
@enduml
```



7.7 UML2 表記の使用

デフォルトでは、UML2 表記が使用されます *(v1.2020.13-14* 以降*)*。

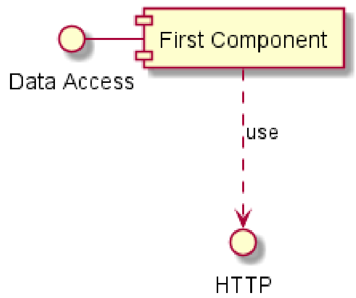
```
@startuml

interface "Data Access" as DA

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

@enduml
```



7.8 UML1 表記の使用

コマンド `skinparam componentStyle uml1` は、UML1 表記に切り替えるために使用されます。

```
@startuml
skinparam componentStyle uml1

interface "Data Access" as DA

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

@enduml
```

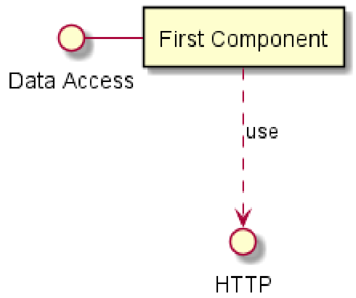


7.9 四角形表記の使用 (UML 表記をしない)

`skinparam componentStyle rectangle` コマンドを使用すると、UML 表記ではなく四角形による表記を行うことができます。

```
@startuml
skinparam componentStyle rectangle

interface "Data Access" as DA

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

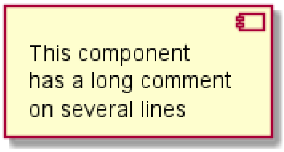
```
@enduml
```



7.10 長い説明

角括弧を使用して説明を複数行で記述することができます。

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



7.11 個々の色

コンポーネント定義のあとに色を指定することができます。

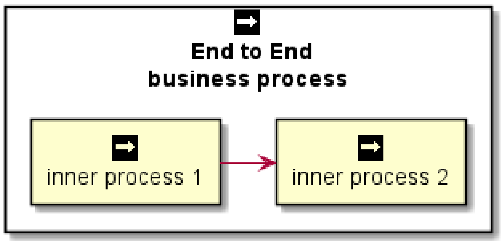
```
@startuml
component  [Web Server] #Yellow
@enduml
```

7.12 ステレオタイプでスプライトを使用

ステレオタイプのコンポーネント内にスプライトを使用することができます。

```
@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
```



7.13 見かけを変える

ダイアグラムの色やフォントを変更するには skinparam コマンドを使用します。

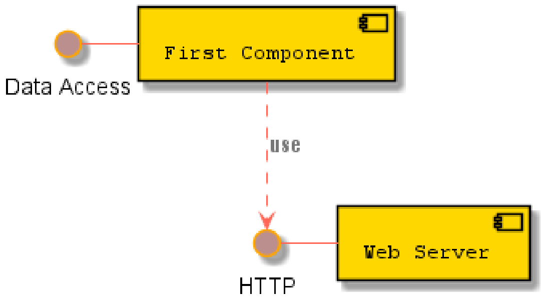
このコマンドは以下の場面で使用できます。

- ダイアグラム定義内で他のコマンドを同様に。
- インクルードされたファイル内。
- 設定ファイルのコマンドライン内や Ant タスク内。

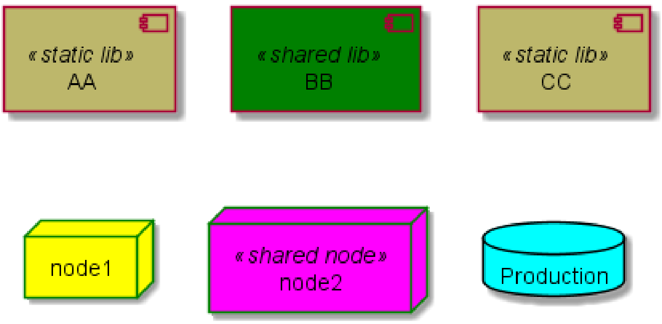
ステレオタイプのコンポーネントおよびインタフェースのための特定の色とフォントを定義することができます。

```
@startuml

skinparam interface {
  backgroundColor RosyBrown
  borderColor orange
}

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

() "Data Access" as DA

DA - [First Component]
[First Component] ..> () HTTP : use
HTTP - [Web Server] << Apache >>
```

```
@enduml
```




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

node node1
node node2 <<shared node>>
database Production

skinparam component {
    backgroundColor<<static lib>> DarkKhaki
    backgroundColor<<shared lib>> Green
}

skinparam node {
borderColor Green
backgroundColor Yellow
backgroundColor<<shared node>> Magenta
}
skinparam databaseBackgroundColor Aqua

@enduml
```



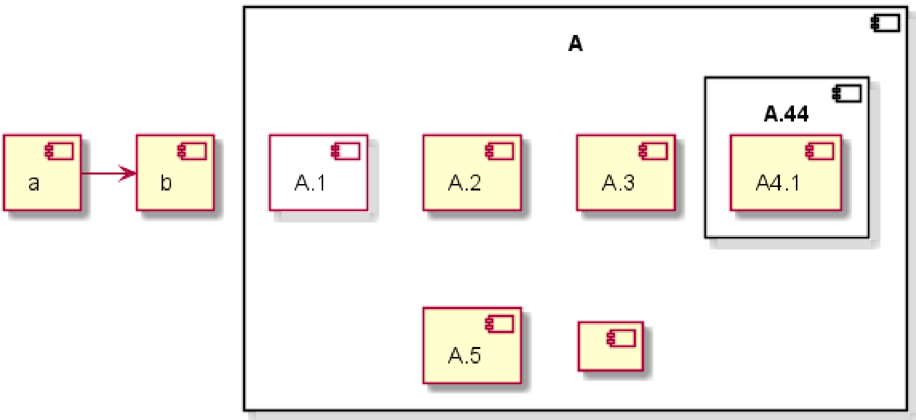
7.14 特有の skinparam

7.14.1 componentStyle

- デフォルトで (もしくは skinparam componentStyle uml2 を指定することで)、コンポーネントにはアイコンが表示されます。

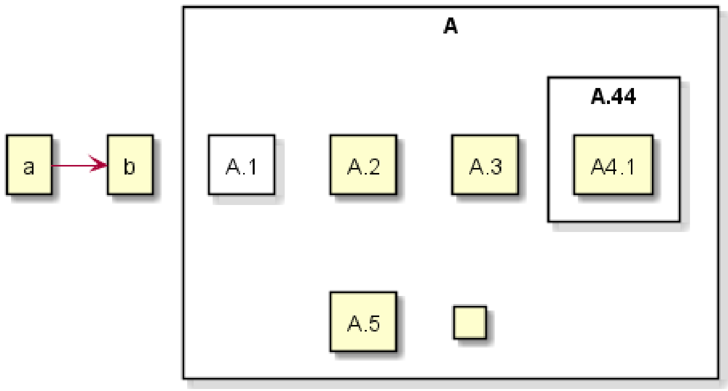
```
@startuml
skinparam BackgroundColor transparent
skinparam componentStyle uml2
component A {
   component "A.1" {
}
```

```
    component A.44 {
        [A4.1]
}
    component "A.2"
    [A.3]
    component A.5 [
A.5]
    component A.6 [
]
}
[a]->[b]
@enduml
```



- これを非表示にして四角形だけを表示するには、`skinparam componentStyle rectangle` を指定します。

```
@startuml
skinparam BackgroundColor transparent
skinparam componentStyle rectangle
component A {
    component "A.1" {
}
    component A.44 {
        [A4.1]
}
    component "A.2"
    [A.3]
    component A.5 [
A.5]
    component A.6 [
]
}
[a]->[b]
@enduml
```

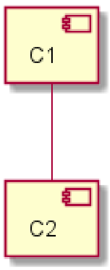


[Ref. 10798]

7.15 Hide or Remove unlinked component

By default, all components are displayed:

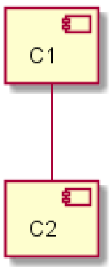
```
@startuml
component C1
component C2
component C3
C1 -- C2
@enduml
```



But you can:

- `hide @unlinked` components:

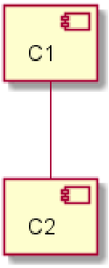
```
@startuml
component C1
component C2
component C3
C1 -- C2

hide @unlinked
@enduml
```



- or `remove @unlinked` components:

```
@startuml
```

```
component C1
component C2
component C3
C1 -- C2

remove @unlinked
@enduml
```

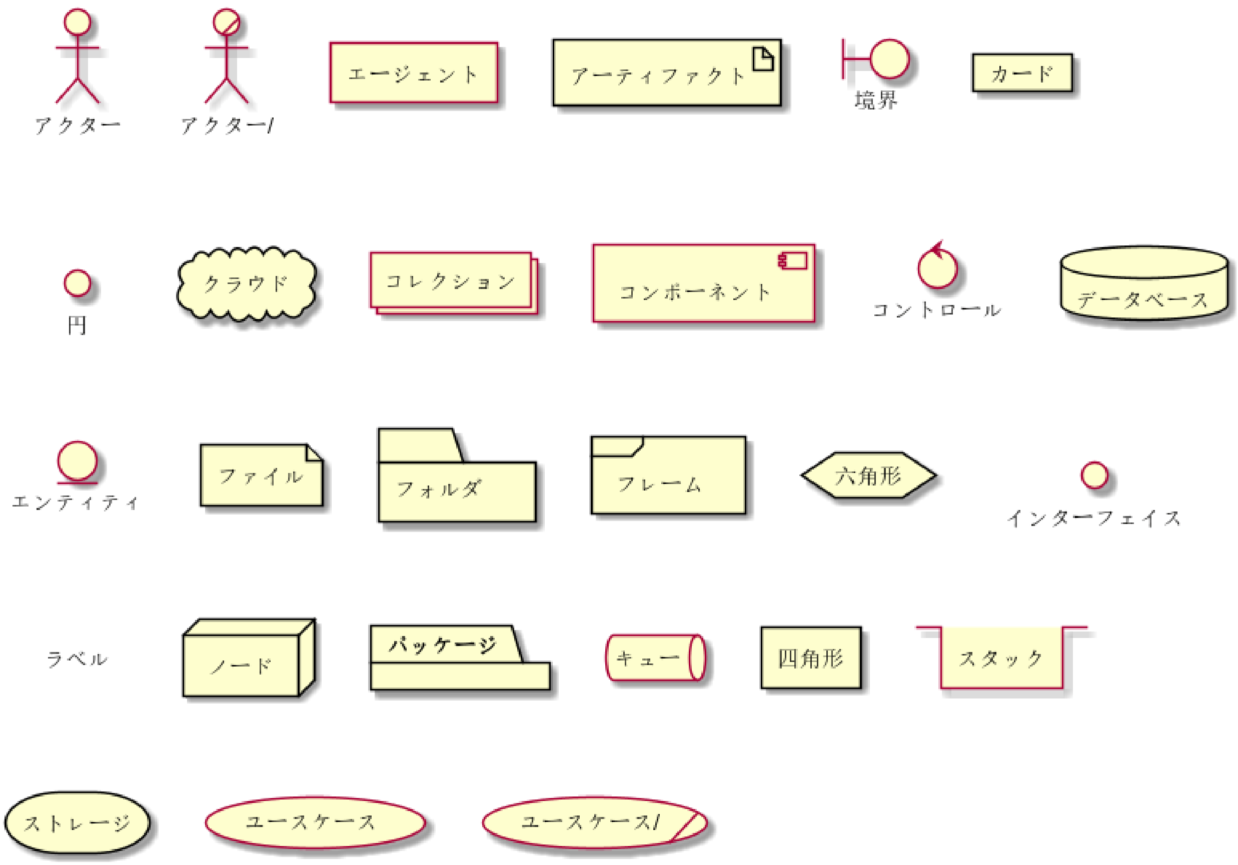


[Ref. QA-11052]

8 配置図

8.1 要素の宣言

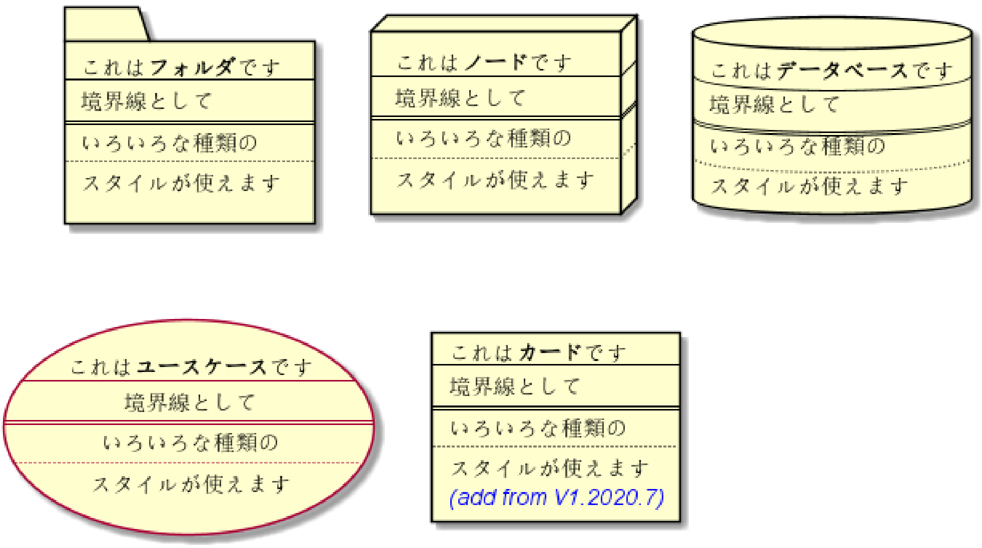
```
@startuml
actor アクター
actor/ "アクター/"
agent エージェント
artifact アーティファクト
boundary 境界
card カード
circle 円
cloud クラウド
collections コレクション
component コンポーネント
control コントロール
database データベース
entity エンティティ
file ファイル
folder フォルダ
frame フレーム
hexagon 六角形
interface インターフェイス
label ラベル
node ノード
package パッケージ
queue キュー
rectangle 四角形
stack スタック
storage ストレージ
usecase ユースケース
usecase/ "ユースケース/"
@enduml
```



説明文が長くなる場合は、オプションでテキストを [] の中に書くこともできます。

```
@startuml
folder フォルダ [
これは<b>フォルダ</b>です
----
境界線として
====
いろいろな種類の
....
スタイルが使えます
]

node ノード [
これは<b>ノード</b>です
----
境界線として
====
いろいろな種類の
....
スタイルが使えます
]

database データベース [
これは<b>データベース</b>です
----
境界線として
====
いろいろな種類の
....
スタイルが使えます
```

```
]

usecase ユースケース [
これは<b>ユースケース</b>です
----
境界線として
====
いろいろな種類の
....
スタイルが使えます
]

card カード [
これは<b>カード</b>です
----
境界線として
====
いろいろな種類の
....
スタイルが使えます
<i><color:blue>(add from V1.2020.7)</color></i>
]
@enduml
```



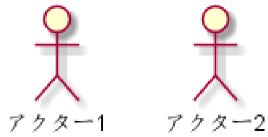
8.2 要素の宣言 (省略記法)

いくつかの省略記法を使って要素を宣言することができます。

通常記法のキーワード	省略記法のキーワード	通常
actor	: a :	acto
component	[c]	comp
interface	() i	inte
usecase	(u)	usec

8.2.1 アクター

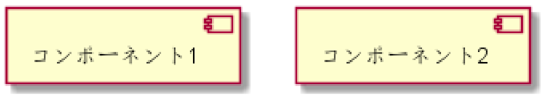
```
@startuml

actor アクター1
:アクター2:

@enduml
```



注意: 二重山括弧 (guillemet) を使用したアクターの古い記法がありますが、現在は非推奨であり、削除される予定です。今後は使用しないでください。

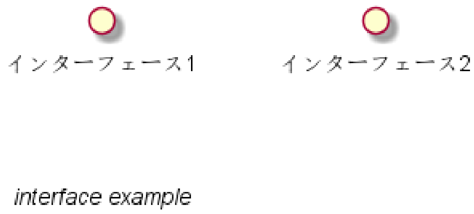
8.2.2 コンポーネント

```
@startuml

component コンポーネント1
[コンポーネント2]

@enduml
```



8.2.3 インターフェース

```
@startuml

interface インターフェース1
() "インターフェース2"

label "//interface example//"
@enduml
```



8.2.4 ユースケース

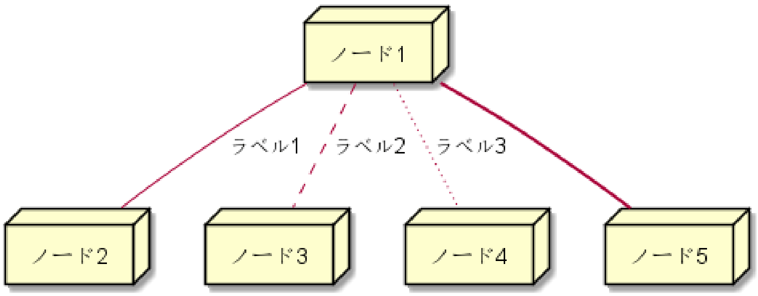
```
@startuml

usecase ユースケース1
(ユースケース2)

@enduml
```

8.3 リンク、矢印

要素の間をシンプルなリンクで結ぶことができます。リンクにラベルを付けることもできます。

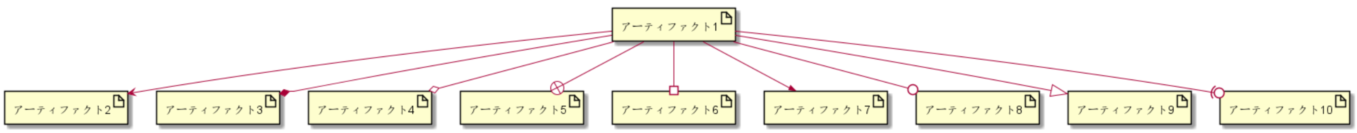
```
@startuml

node ノード1
node ノード2
node ノード3
```

```
node ノード4
node ノード5
ノード1 -- ノード2 : ラベル1
ノード1 .. ノード3 : ラベル2
ノード1 ~~ ノード4 : ラベル3
ノード1 == ノード5

@enduml
```



複数の種類のリンクを使うこともできます。

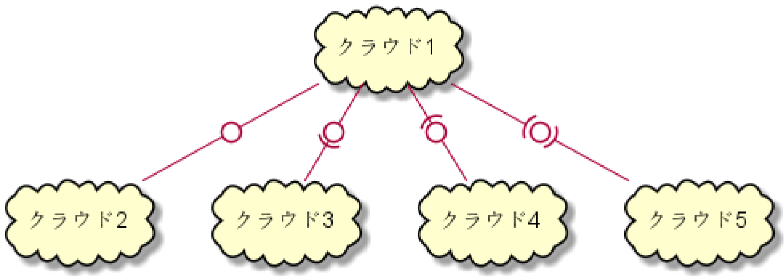
```
@startuml

artifact アーティファクト1
artifact アーティファクト2
artifact アーティファクト3
artifact アーティファクト4
artifact アーティファクト5
artifact アーティファクト6
artifact アーティファクト7
artifact アーティファクト8
artifact アーティファクト9
artifact アーティファクト10
アーティファクト1 --> アーティファクト2
アーティファクト1 --* アーティファクト3
アーティファクト1 --o アーティファクト4
アーティファクト1 --+ アーティファクト5
アーティファクト1 --# アーティファクト6
アーティファクト1 -->> アーティファクト7
アーティファクト1 --0 アーティファクト8
アーティファクト1 --^ アーティファクト9
アーティファクト1 --(0 アーティファクト10

@enduml
```



次のような種類のリンクも使用できます。

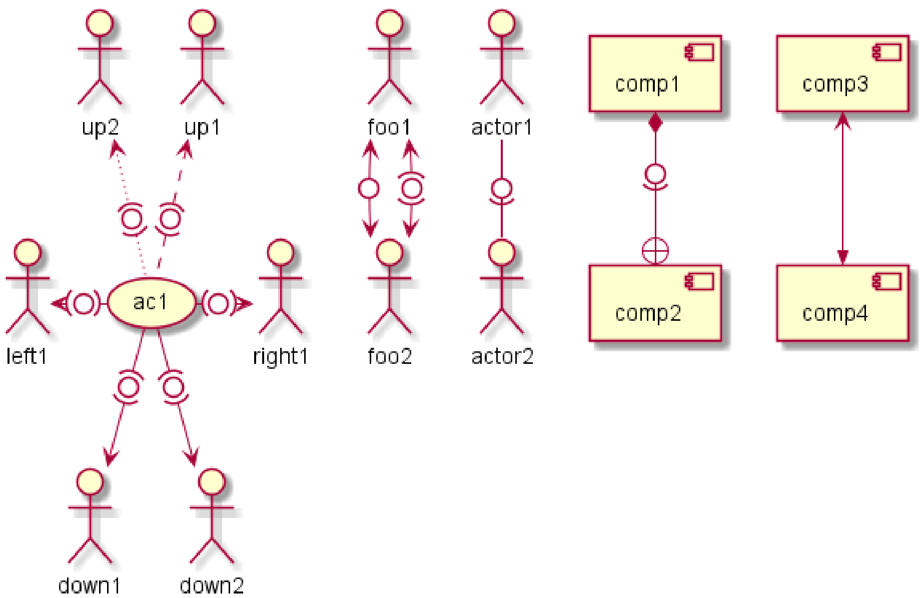
```
@startuml

cloud クラウド1
cloud クラウド2
cloud クラウド3
cloud クラウド4
cloud クラウド5
クラウド1 -0- クラウド2
```

```
クラウド1 -0)- クラウド3
クラウド1 -(0- クラウド4
クラウド1 -(0)- クラウド5

@enduml
```



別の例:

```
@startuml
actor foo1
actor foo2
foo1 <-0-> foo2
foo1 <-(0)-> foo2

(ac1) -le(0)-> left1
ac1 -ri(0)-> right1
ac1 .up(0).> up1
ac1 ~up(0)~> up2
ac1 -do(0)-> down1
ac1 -do(0)-> down2

actor1 -0)- actor2

component comp1
component comp2
comp1 *-0)-+ comp2
[comp3] <-->> [comp4]
@enduml
```



[Ref. QA-1736]

See all type on **Appendix**.

8.4 各括弧を使用した矢印のスタイル

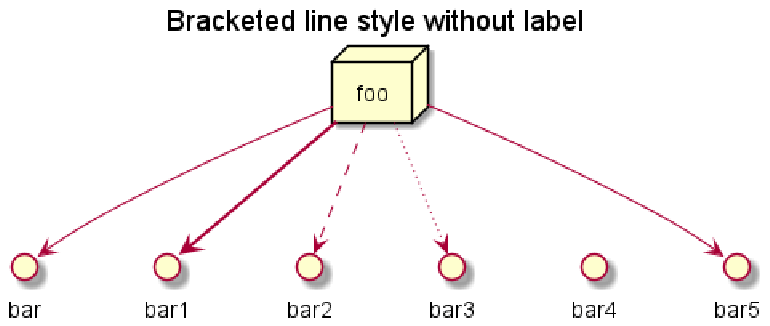
(角括弧を使用したクラスの関係(リンク、矢印)のスタイルと同様)

8.4.1 線のスタイル

矢印に bold、dashed、dotted、hidden、plain のスタイルを指定することができます:

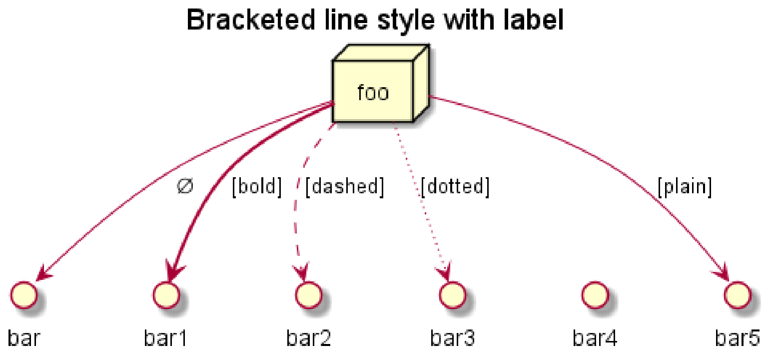
- ラベル無し

```
@startuml
node foo
title Bracketed line style without label
foo --> bar
foo -[bold]-> bar1
foo -[dashed]-> bar2
foo -[dotted]-> bar3
foo -[hidden]-> bar4
foo -[plain]-> bar5
@enduml
```



- ラベル有り

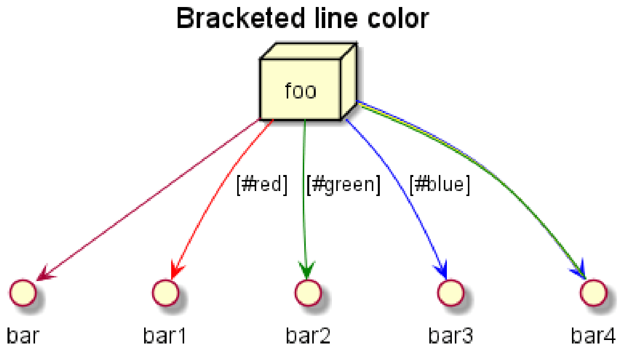
```
@startuml
title Bracketed line style with label
node foo
foo --> bar          :
foo -[bold]-> bar1   : [bold]
foo -[dashed]-> bar2 : [dashed]
foo -[dotted]-> bar3 : [dotted]
foo -[hidden]-> bar4 : [hidden]
foo -[plain]-> bar5  : [plain]
@enduml
```



[Adapted from QA-4181]

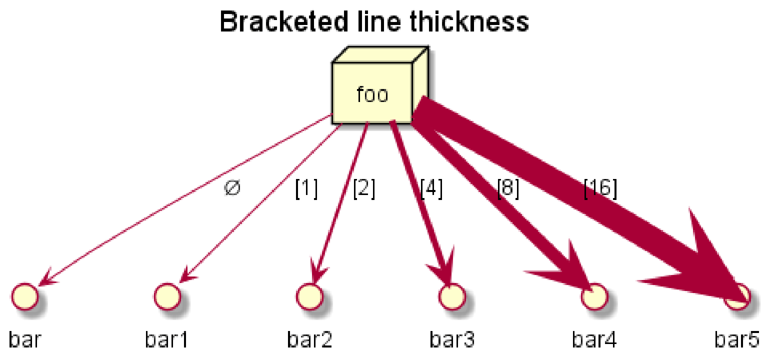
8.4.2 線の色

```
@startuml
title Bracketed line color
node  foo
foo --> bar
foo -[#red]-> bar1     : [#red]
foo -[#green]-> bar2   : [#green]
foo -[#blue]-> bar3    : [#blue]
foo -[#blue;#yellow;#green]-> bar4
@enduml
```



8.4.3 線の太さ

```
@startuml
title Bracketed line thickness
node foo
foo --> bar                  :
foo -[thickness=1]-> bar1    : [1]
foo -[thickness=2]-> bar2    : [2]
foo -[thickness=4]-> bar3    : [4]
foo -[thickness=8]-> bar4    : [8]
foo -[thickness=16]-> bar5   : [16]
@enduml
```

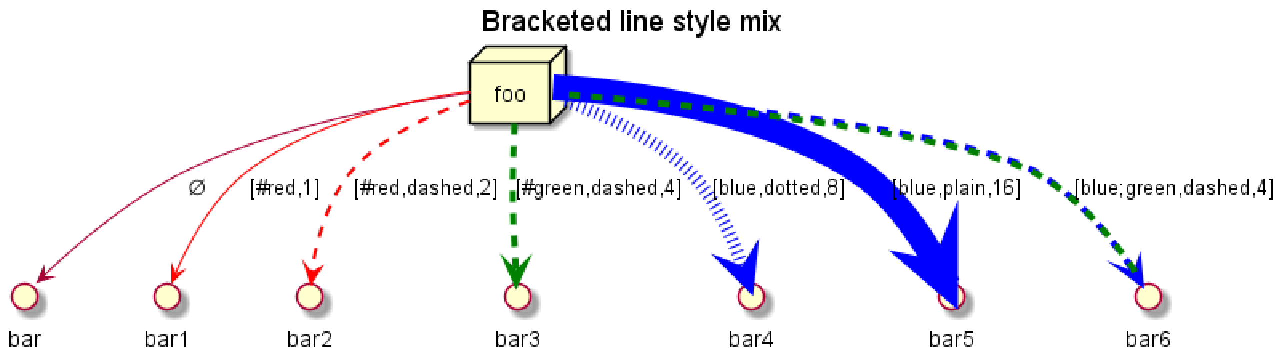


[Adapted from QA-4949]

8.4.4 混合

```
@startuml
title Bracketed line style mix
node foo
foo --> bar                              :
foo -[#red,thickness=1]-> bar1           : [#red,1]
foo -[#red,dashed,thickness=2]-> bar2    : [#red,dashed,2]
foo -[#green,dashed,thickness=4]-> bar3 : [#green,dashed,4]
```

```
foo -[#blue,dotted,thickness=8]-> bar4  : [blue,dotted,8]
foo -[#blue,plain,thickness=16]-> bar5  : [blue,plain,16]
foo -[#blue;#green,dashed,thickness=4]-> bar6  : [blue;green,dashed,4]
@enduml
```

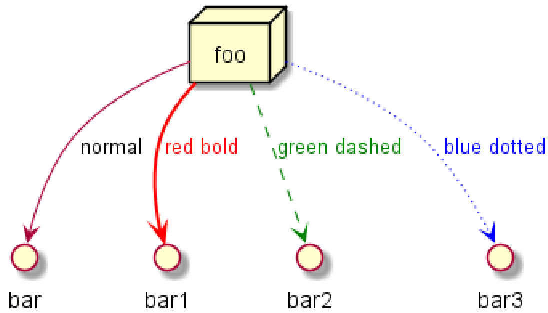


8.5 矢印の色とスタイルを変更する（インラインスタイル）

個別の矢印ごとに色とスタイルを変更するには、次の記法を使用します：

- `#color;line.[bold|dashed|dotted];text:color`

```
@startuml
node foo
foo --> bar : normal
foo --> bar1 #line:red;line.bold;text:red  : red bold
foo --> bar2 #green;line.dashed;text:green : green dashed
foo --> bar3 #blue;line.dotted;text:blue   : blue dotted
@enduml
```

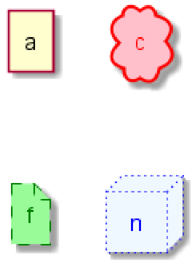


[Ref. QA-3770 and QA-3816] [See similar feature on class diagram]

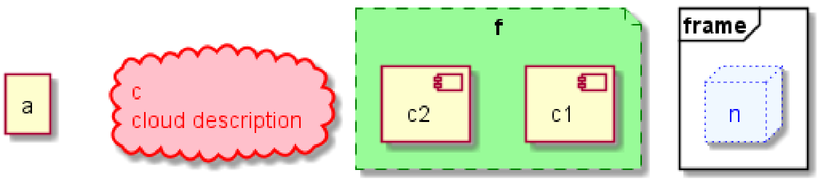
8.6 要素の色とスタイルを変更する（インラインスタイル）

それぞれ個別の要素について、色とスタイルを変更するには、次の記法を使用します：`#color;line:color;line.[bold`

```
@startuml
agent a
cloud c #pink;line:red;line.bold;text:red
file  f #palegreen;line:green;line.dashed;text:green
node  n #aliceblue;line:blue;line.dotted;text:blue
@enduml
```




```
@startuml
agent a
cloud c #pink;line:red;line.bold;text:red [
c
cloud description
]
file  f #palegreen;line:green;line.dashed;text:green {
[c1]
[c2]
}
frame frame {
node  n #aliceblue;line:blue;line.dotted;text:blue
}
@enduml
```



[Ref. QA-6852]

8.7　入れ子にできる要素

次の要素は入れ子にできます：

```
@startuml
artifact アーティファクト {
}
card カード {
}
cloud クラウド {
}
component コンポーネント {
}
database データベース {
}
file ファイル {
}
folder フォルダ {
}
frame フレーム {
}
hexagon 六角形 {
}
node ノード {
}
package パッケージ {
}
```

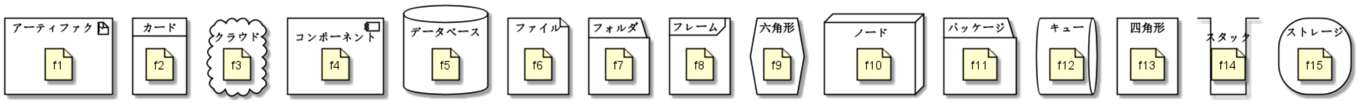
```
queue キュー {
}
rectangle 四角形 {
}
stack スタック {
}
storage ストレージ {
}
@enduml
```

8.8 パッケージと入れ子要素

8.8.1 一階層の例

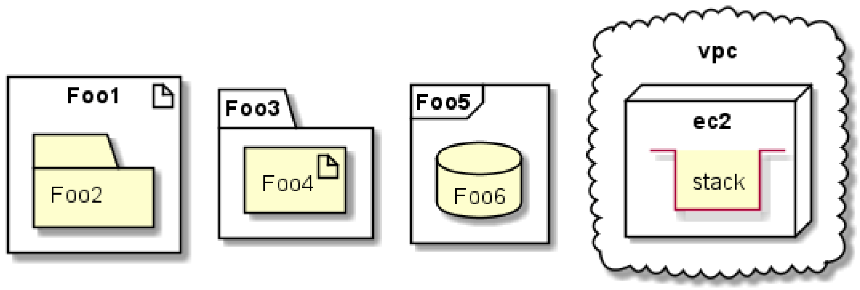
```
@startuml
artifact    artifactVeryLOOOOOOOOOOOOOOOOOOOg    as "アーティファクト" {
file f1
}
card        cardVeryLOOOOOOOOOOOOOOOOOOOg        as "カード" {
file f2
}
cloud       cloudVeryLOOOOOOOOOOOOOOOOOOOg       as "クラウド" {
file f3
}
component   componentVeryLOOOOOOOOOOOOOOOOOOOg   as "コンポーネント" {
file f4
}
database    databaseVeryLOOOOOOOOOOOOOOOOOOOg    as "データベース" {
file f5
}
file        fileVeryLOOOOOOOOOOOOOOOOOOOg        as "ファイル" {
file f6
}
folder      folderVeryLOOOOOOOOOOOOOOOOOOOg      as "フォルダ" {
file f7
}
frame       frameVeryLOOOOOOOOOOOOOOOOOOOg       as "フレーム" {
file f8
}
hexagon     hexagonVeryLOOOOOOOOOOOOOOOOOOOg     as "六角形" {
file f9
}
node        nodeVeryLOOOOOOOOOOOOOOOOOOOg        as "ノード" {
file f10
}
package     packageVeryLOOOOOOOOOOOOOOOOOOOg     as "パッケージ" {
file f11
}
queue       queueVeryLOOOOOOOOOOOOOOOOOOOg       as "キュー" {
file f12
}
rectangle   rectangleVeryLOOOOOOOOOOOOOOOOOOOg   as "四角形" {
file f13
}
stack       stackVeryLOOOOOOOOOOOOOOOOOOOg       as "スタック" {
file f14
```

```
}
storage     storageVeryL0000000000000000000g     as "ストレージ" {
file f15
}
@enduml
```



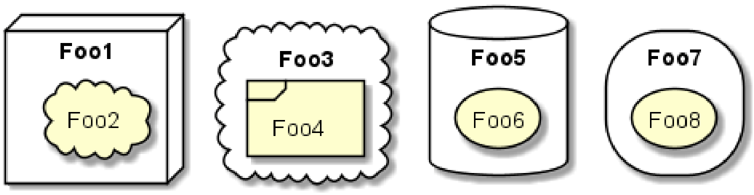
8.8.2 他の例

```
@startuml
artifact Foo1 {
  folder Foo2
}

folder Foo3 {
  artifact Foo4
}

frame Foo5 {
  database Foo6
}

cloud vpc {
  node ec2 {
    stack stack
  }
}

@enduml
```




```
@startuml
node Foo1 {
 cloud Foo2
}

cloud Foo3 {
  frame Foo4
}

database Foo5  {
  storage Foo6
}

storage Foo7 {
  storage Foo8
```

```
}
@enduml
```

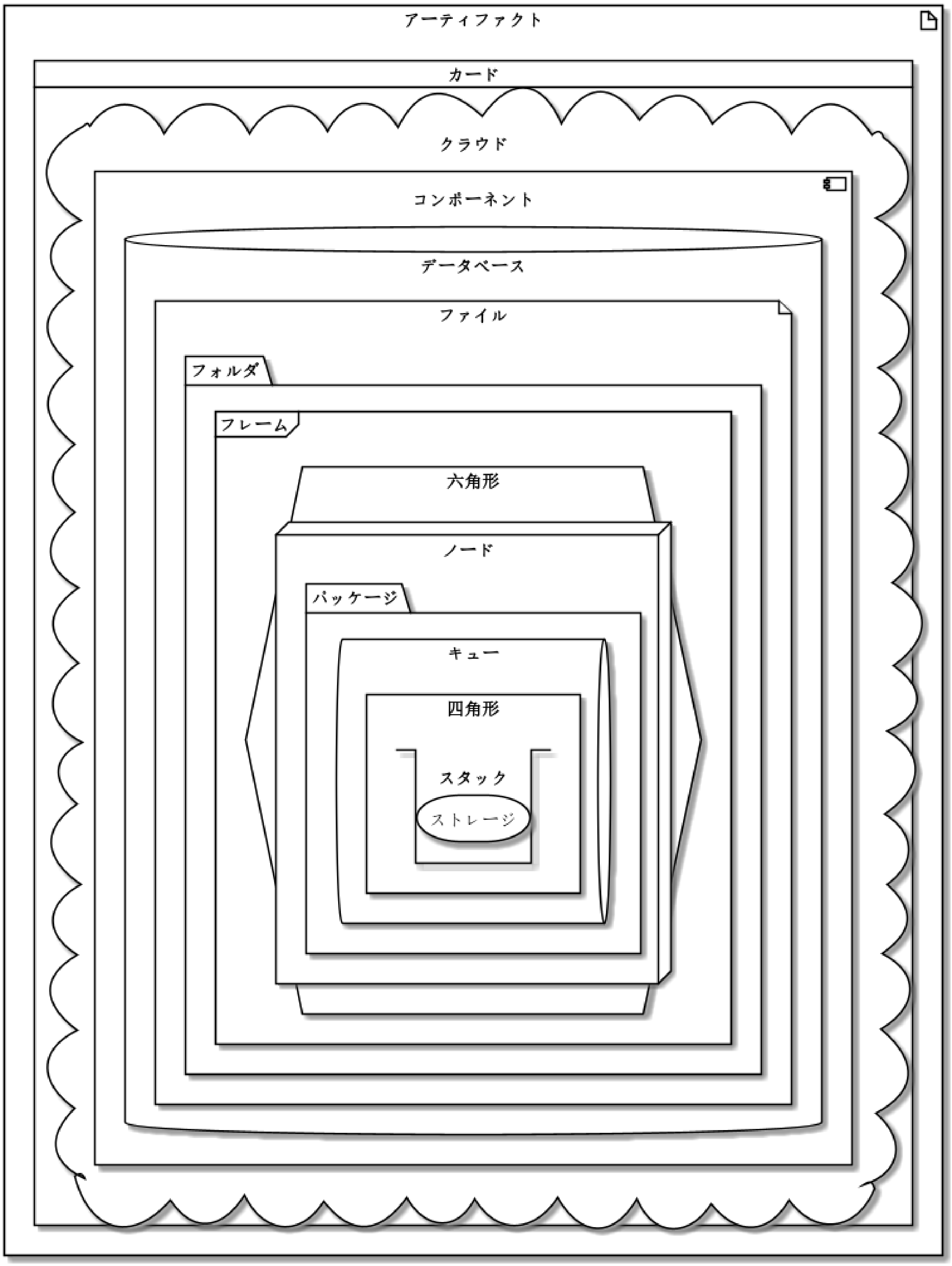


8.8.3 すべて入れ子にした例

すべての入れ子要素の例です:

- アルファベット順:

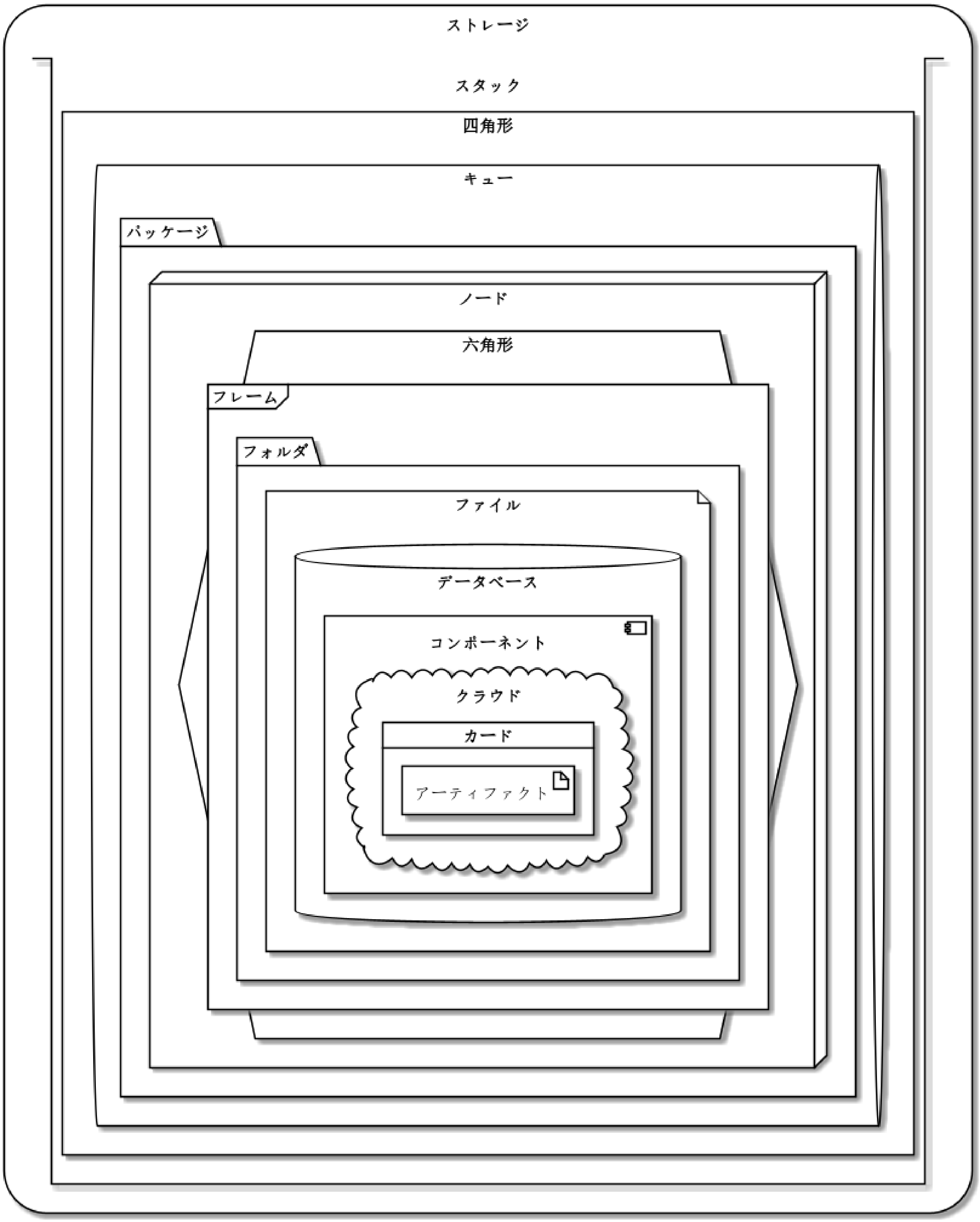
```
@startuml
artifact アーティファクト {
card カード {
cloud クラウド {
component コンポーネント {
database データベース {
file ファイル {
folder フォルダ {
frame フレーム {
hexagon 六角形 {
node ノード {
package パッケージ {
queue キュー {
rectangle 四角形 {
stack スタック {
storage ストレージ {
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
@enduml
```



- アルファベットの逆順

```
@startuml
storage ストレージ {
stack スタック {
rectangle 四角形 {
queue キュー {
```

```
package パッケージ {
node ノード {
hexagon 六角形 {
frame フレーム {
folder フォルダ {
file ファイル {
database データベース {
component コンポーネント {
cloud クラウド {
card カード {
artifact アーティファクト {
}
}
}
}
}
}
}
}
}
}
}
}
}
}
}
@enduml
```

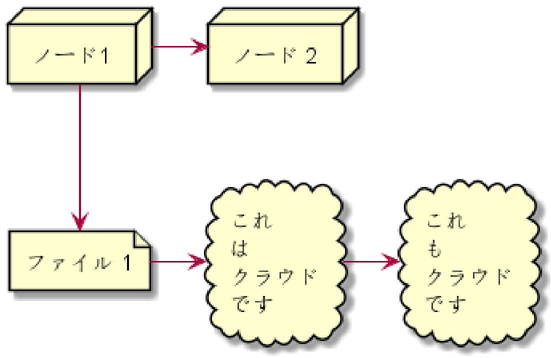


8.9　別名

8.9.1　as による単純な別名

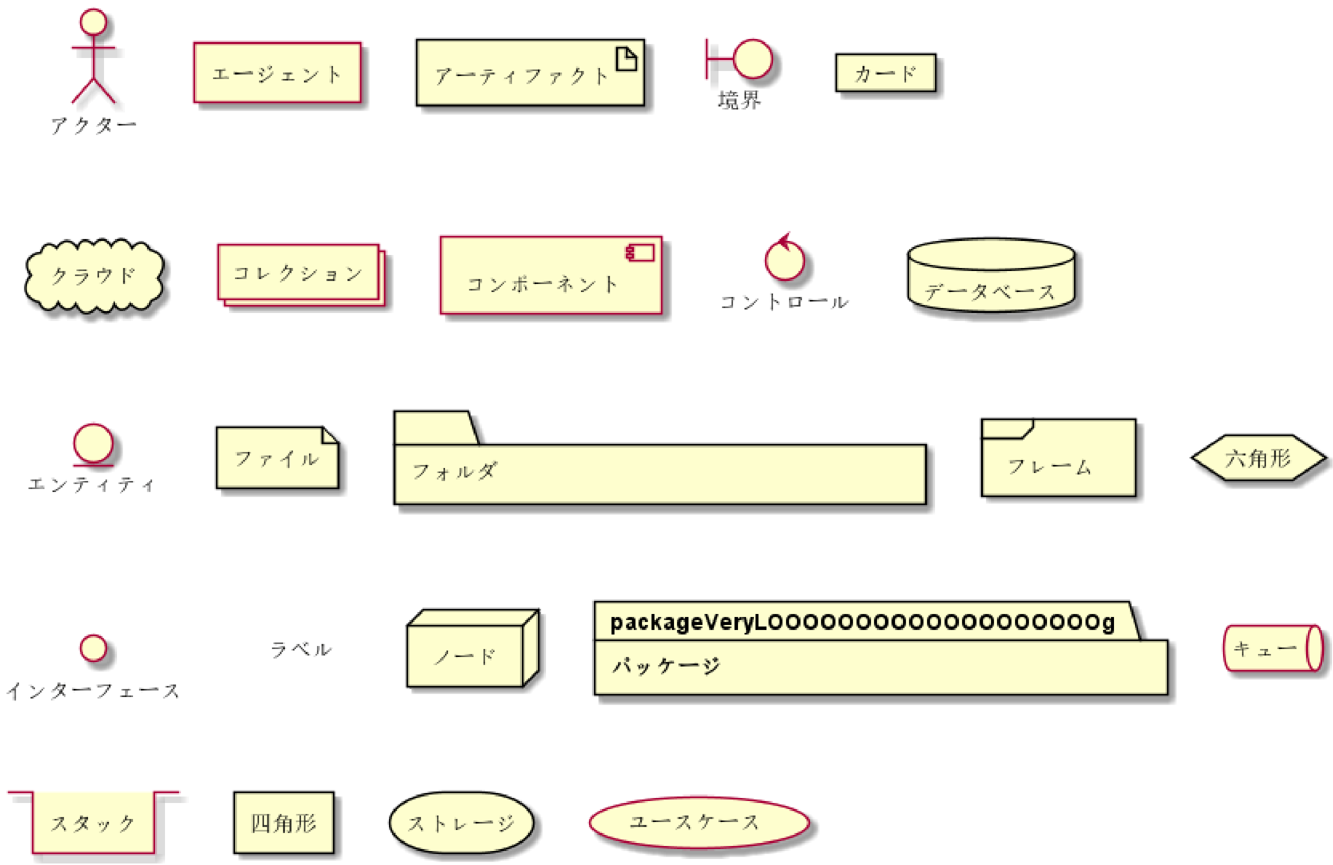
```
@startuml
node ノード1 as n1
node "ノード 2" as n2
```

```
file f1 as "ファイル 1"
cloud c1 as "これ
は
クラウド
です"
cloud c2 [これ
も
クラウド
です]

n1 -> n2
n1 --> f1
f1 -> c1
c1 -> c2
@enduml
```

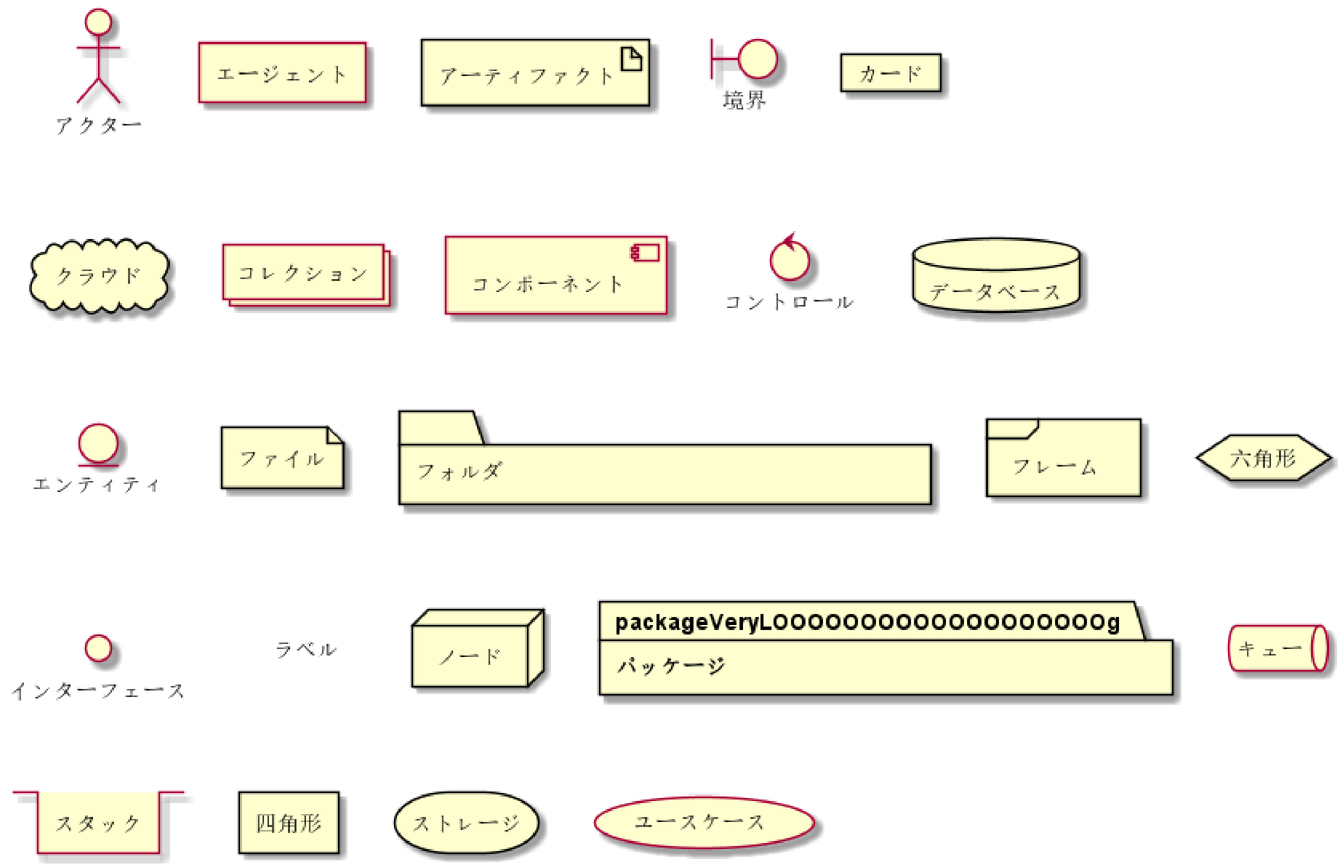


8.9.2 長い別名の例

```
@startuml
actor         "アクター"          as actorVeryL0000000000000000000g
agent         "エージェント"      as agentVeryL0000000000000000000g
artifact      "アーティファクト"  as artifactVeryL0000000000000000000g
boundary      "境界"              as boundaryVeryL0000000000000000000g
card          "カード"            as cardVeryL0000000000000000000g
cloud         "クラウド"          as cloudVeryL0000000000000000000g
collections   "コレクション"      as collectionsVeryL0000000000000000000g
component     "コンポーネント"    as componentVeryL0000000000000000000g
control       "コントロール"      as controlVeryL0000000000000000000g
database      "データベース"      as databaseVeryL0000000000000000000g
entity        "エンティティ"      as entityVeryL0000000000000000000g
file          "ファイル"          as fileVeryL0000000000000000000g
folder        "フォルダ"          as folderVeryL0000000000000000000g
frame         "フレーム"          as frameVeryL0000000000000000000g
hexagon       "六角形"            as hexagonVeryL0000000000000000000g
interface     "インターフェース"  as interfaceVeryL0000000000000000000g
label         "ラベル"            as labelVeryL0000000000000000000g
node          "ノード"            as nodeVeryL0000000000000000000g
package       "パッケージ"        as packageVeryL0000000000000000000g
queue         "キュー"            as queueVeryL0000000000000000000g
stack         "スタック"          as stackVeryL0000000000000000000g
rectangle     "四角形"            as rectangleVeryL0000000000000000000g
storage       "ストレージ"        as storageVeryL0000000000000000000g
usecase       "ユースケース"      as usecaseVeryL0000000000000000000g
@enduml
```



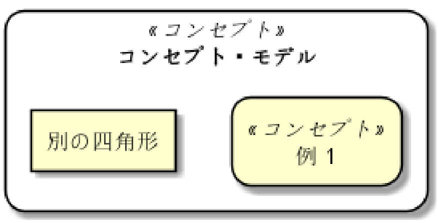

```
@startuml
actor       actorVeryLOOOOOOOOOOOOOOOOOOOg       as "アクター"
agent       agentVeryLOOOOOOOOOOOOOOOOOOOg       as "エージェント"
artifact    artifactVeryLOOOOOOOOOOOOOOOOOOOg    as "アーティファクト"
boundary    boundaryVeryLOOOOOOOOOOOOOOOOOOOg    as "境界"
card        cardVeryLOOOOOOOOOOOOOOOOOOOg        as "カード"
cloud       cloudVeryLOOOOOOOOOOOOOOOOOOOg       as "クラウド"
collections collectionsVeryLOOOOOOOOOOOOOOOOOOOg as "コレクション"
component   componentVeryLOOOOOOOOOOOOOOOOOOOg   as "コンポーネント"
control     controlVeryLOOOOOOOOOOOOOOOOOOOg     as "コントロール"
database    databaseVeryLOOOOOOOOOOOOOOOOOOOg    as "データベース"
entity      entityVeryLOOOOOOOOOOOOOOOOOOOg      as "エンティティ"
file        fileVeryLOOOOOOOOOOOOOOOOOOOg        as "ファイル"
folder      folderVeryLOOOOOOOOOOOOOOOOOOOg      as "フォルダ"
frame       frameVeryLOOOOOOOOOOOOOOOOOOOg       as "フレーム"
hexagon     hexagonVeryLOOOOOOOOOOOOOOOOOOOg     as "六角形"
interface   interfaceVeryLOOOOOOOOOOOOOOOOOOOg   as "インターフェース"
label       labelVeryLOOOOOOOOOOOOOOOOOOOg       as "ラベル"
node        nodeVeryLOOOOOOOOOOOOOOOOOOOg        as "ノード"
package     packageVeryLOOOOOOOOOOOOOOOOOOOg     as "パッケージ"
queue       queueVeryLOOOOOOOOOOOOOOOOOOOg       as "キュー"
stack       stackVeryLOOOOOOOOOOOOOOOOOOOg       as "スタック"
rectangle   rectangleVeryLOOOOOOOOOOOOOOOOOOOg   as "四角形"
storage     storageVeryLOOOOOOOOOOOOOOOOOOOg     as "ストレージ"
usecase     usecaseVeryLOOOOOOOOOOOOOOOOOOOg     as "ユースケース"
@enduml
```



[Ref. QA-12082]

8.10 角に丸みをつける

```
@startuml
skinparam rectangle {
    roundCorner<<コンセプト>> 25
}


rectangle "コンセプト· モデル" <<コンセプト>> {
rectangle "例 1" <<コンセプト>> as ex1
rectangle "別の四角形"
}
@enduml
```

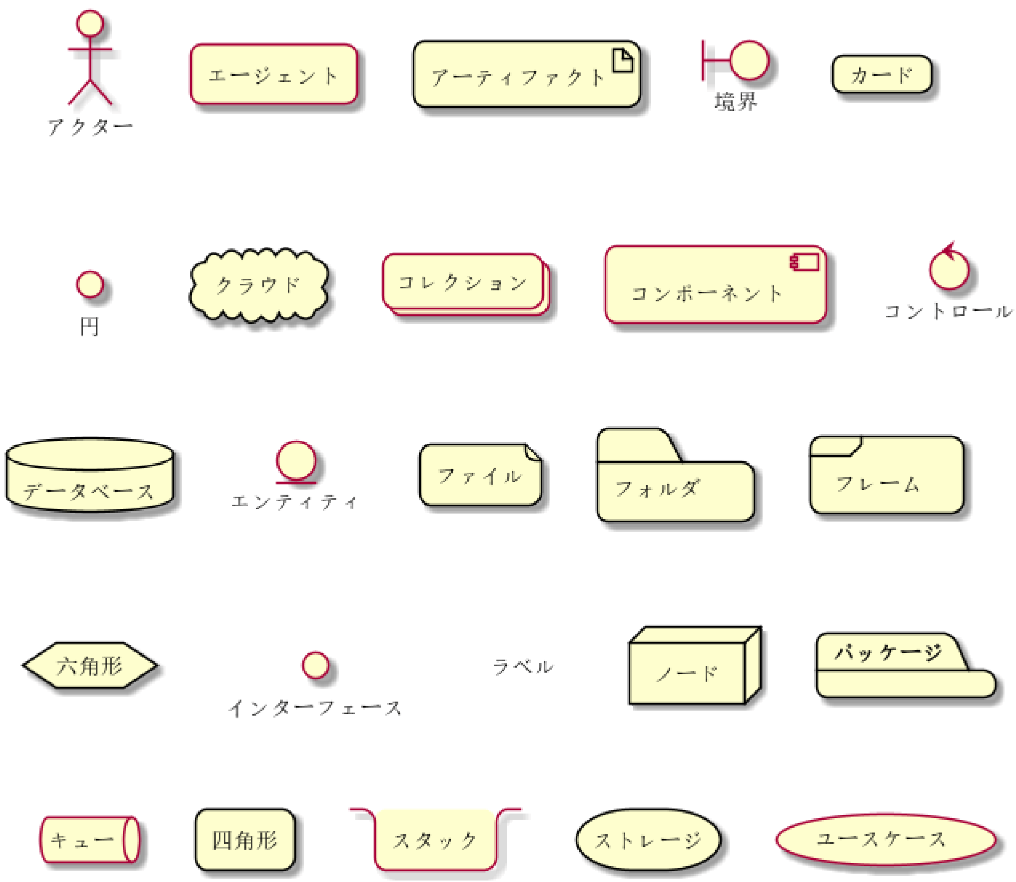


8.11 特有の skinparam

8.11.1 roundCorner

```
@startuml
skinparam roundCorner 15
actor アクター
```

```
agent エージェント
artifact アーティファクト
boundary 境界
card カード
circle 円
cloud クラウド
collections コレクション
component コンポーネント
control コントロール
database データベース
entity エンティティ
file ファイル
folder フォルダ
frame フレーム
hexagon 六角形
interface インターフェース
label ラベル
node ノード
package パッケージ
queue キュー
rectangle 四角形
stack スタック
storage ストレージ
usecase ユースケース
@enduml
```

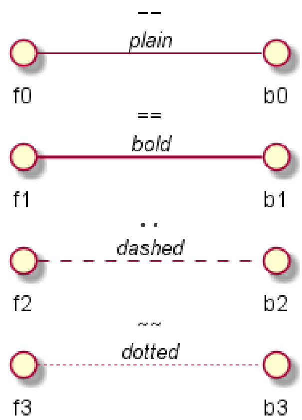


[Ref. QA-5299, QA-6915, QA-11943]

8.12 付録：線の種類の一覧

```
@startuml
```

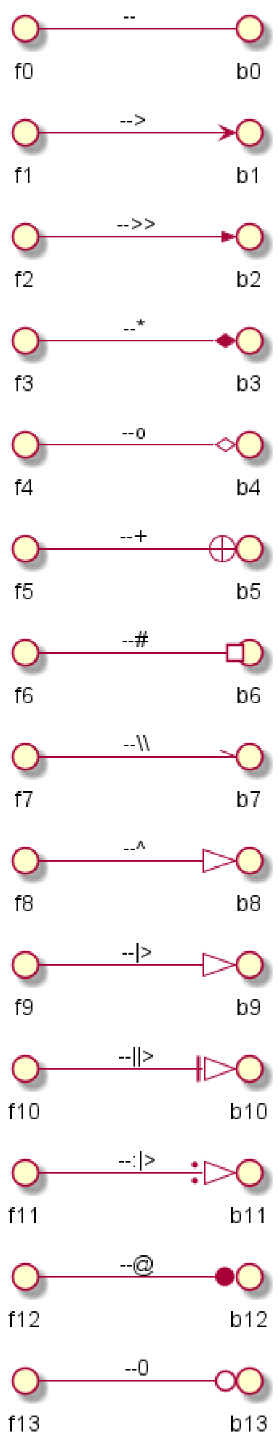
```
left to right direction
skinparam nodesep 5

f3  ~~  b3  : ""~~""\n//dotted//
f2  ..  b2  : ""..""\n//dashed//
f1  ==  b1  : ""==""\n//bold//
f0  --  b0  : ""--""\n//plain//
@enduml
```



8.13 付録：矢印の先端と'0' 矢印の一覧

8.13.1 矢印の先端

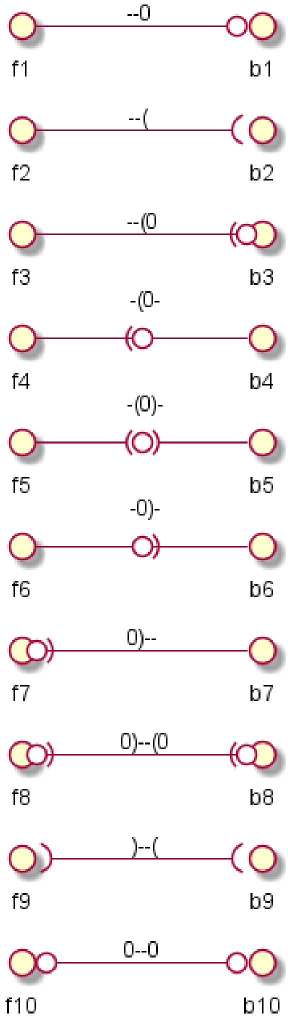
```
@startuml
left to right direction
skinparam nodesep 5

f13 --0   b13 : ""--0""
f12 --@   b12 : ""--@""
f11 --:|> b11 : ""--:|>""
f10 --||> b10 : ""--||>""
f9  --|>  b9  : ""--|>""
f8  --^   b8  : ""--^ ""
f7  --\\  b7  : ""--\\\\""
f6  --#   b6  : ""--# ""
f5  --+   b5  : ""--+ ""
f4  --o   b4  : ""--o ""
f3  --*   b3  : ""--* ""
f2  -->>  b2  : ""-->>""
f1  -->   b1  : ""--> ""
f0  --    b0  : ""--  ""
@enduml
```



8.13.2 丸形の矢印（'0' 矢印）

```
@startuml
left to right direction
skinparam nodesep 5

f10 0--0 b10 : "" 0--0 ""
f9 )--(  b9  : "" )--(""
f8 0)--(0 b8 : "" 0)--(0""
f7 0)--  b7  : "" 0)-- ""
f6 -0)-  b6  : "" -0)-\n ""
f5 -(0)- b5  : "" -(0)-\n""
```

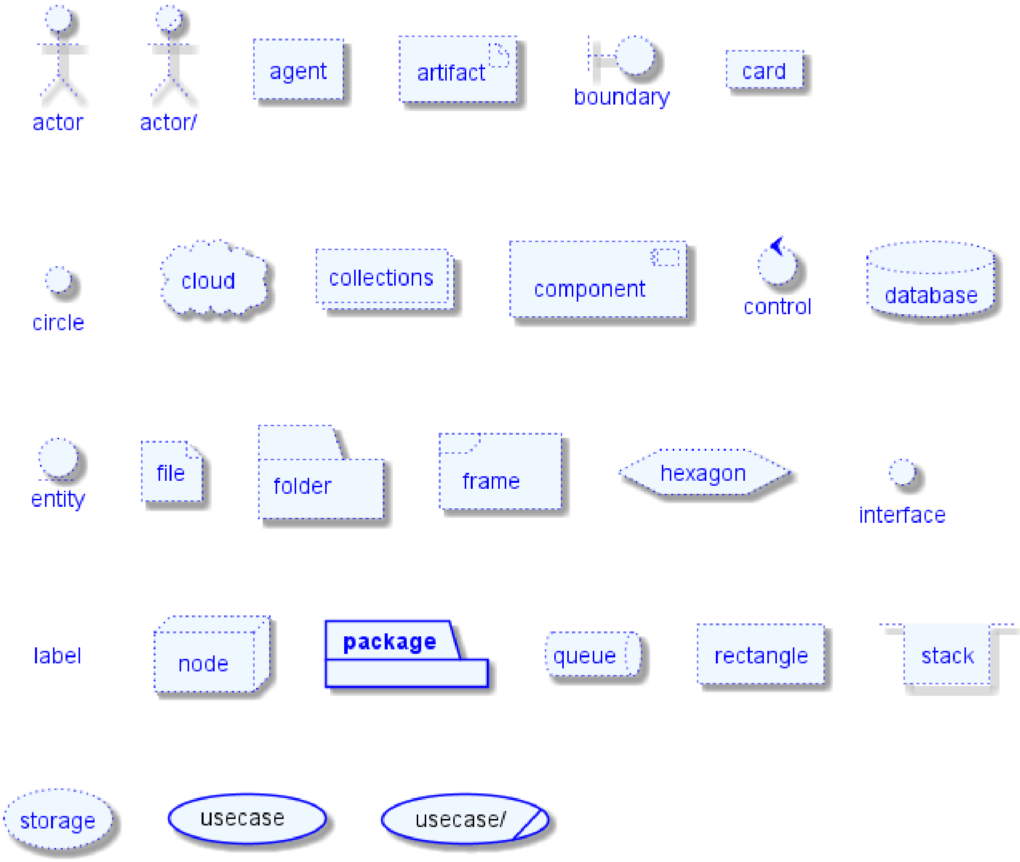
```
f4 -(0-  b4  : "" -(0-\n ""
f3 --(0  b3  : "" --(0 ""
f2 --(   b2  : "" --(  ""
f1 --0   b1  : "" --0  ""
@enduml
```



8.14 Appendix: Test of inline style on all element

```
@startuml
actor actor #aliceblue;line:blue;line.dotted;text:blue
actor/ "actor/" #aliceblue;line:blue;line.dotted;text:blue
agent agent #aliceblue;line:blue;line.dotted;text:blue
artifact artifact #aliceblue;line:blue;line.dotted;text:blue
boundary boundary #aliceblue;line:blue;line.dotted;text:blue
card card #aliceblue;line:blue;line.dotted;text:blue
circle circle #aliceblue;line:blue;line.dotted;text:blue
cloud cloud #aliceblue;line:blue;line.dotted;text:blue
collections collections #aliceblue;line:blue;line.dotted;text:blue
component component #aliceblue;line:blue;line.dotted;text:blue
control control #aliceblue;line:blue;line.dotted;text:blue
database database #aliceblue;line:blue;line.dotted;text:blue
entity entity #aliceblue;line:blue;line.dotted;text:blue
file file #aliceblue;line:blue;line.dotted;text:blue
folder folder #aliceblue;line:blue;line.dotted;text:blue
frame frame #aliceblue;line:blue;line.dotted;text:blue
```

```
hexagon hexagon #aliceblue;line:blue;line.dotted;text:blue
interface interface #aliceblue;line:blue;line.dotted;text:blue
label label #aliceblue;line:blue;line.dotted;text:blue
node node #aliceblue;line:blue;line.dotted;text:blue
package package #aliceblue;line:blue;line.dotted;text:blue
queue queue #aliceblue;line:blue;line.dotted;text:blue
rectangle rectangle #aliceblue;line:blue;line.dotted;text:blue
stack stack #aliceblue;line:blue;line.dotted;text:blue
storage storage #aliceblue;line:blue;line.dotted;text:blue
usecase usecase #aliceblue;line:blue;line.dotted;text:blue
usecase/ "usecase/" #aliceblue;line:blue;line.dotted;text:blue
@enduml
```



9 ステート図

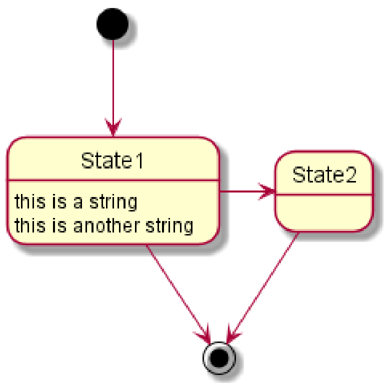
状態遷移図 (ステート図) とは、システムの振る舞いを抽象化して表現するために使われます。This behavior is represented as a series of events that can occur in one or more possible states.

9.1 簡単なステート

ステート図の始点と終点は、[*] で示します。

矢印は、--> で示します。

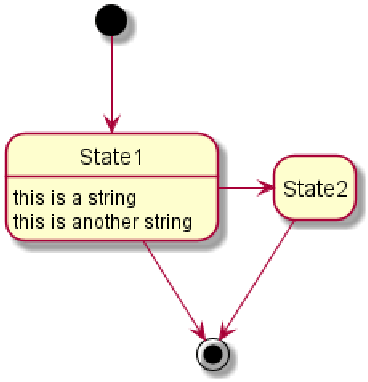
```
@startuml

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

State1 -> State2
State2 --> [*]

@enduml
```



9.2 ステートの表現を変える

hide empty description を使用し、ステート枠をシンプルにできます。

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

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

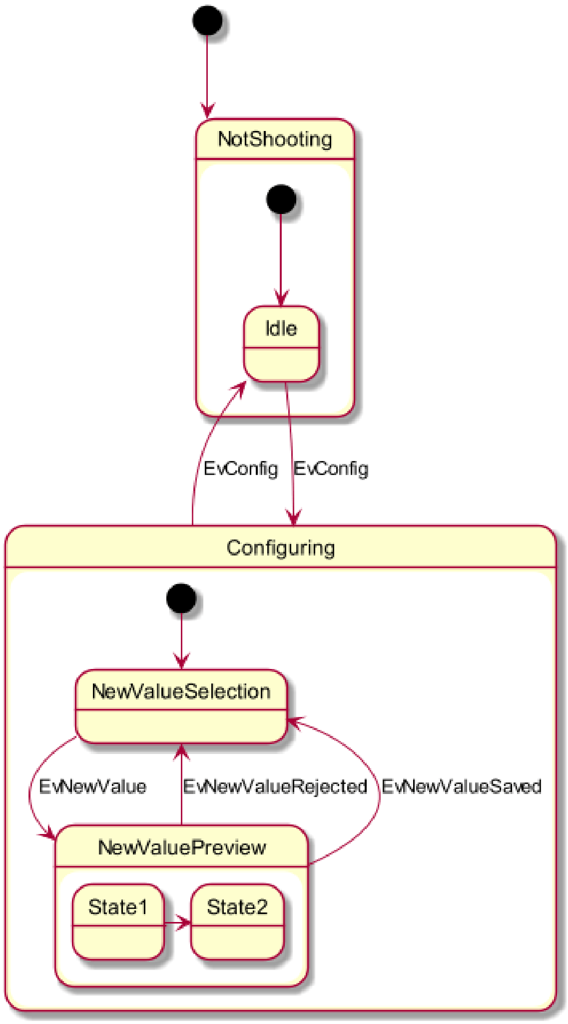


9.3 合成状態

状態は合成することもできます。キーワード `state` と中括弧を使用して定義します。

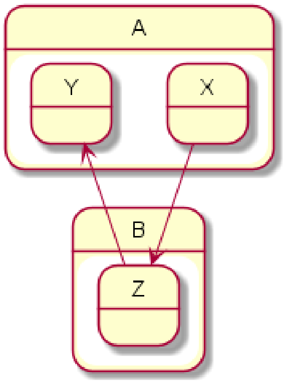
9.3.1 内部サブ状態

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

state NotShooting {
  [*] --> Idle
  Idle --> Configuring : EvConfig
  Configuring --> Idle : EvConfig
}

state Configuring {
  [*] --> NewValueSelection
  NewValueSelection --> NewValuePreview : EvNewValue
  NewValuePreview --> NewValueSelection : EvNewValueRejected
  NewValuePreview --> NewValueSelection : EvNewValueSaved

  state NewValuePreview {
     State1 -> State2
  }

}
@enduml
```



9.3.2 サブ状態からサブ状態へ

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

state B {
  state Z {
  }
}

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

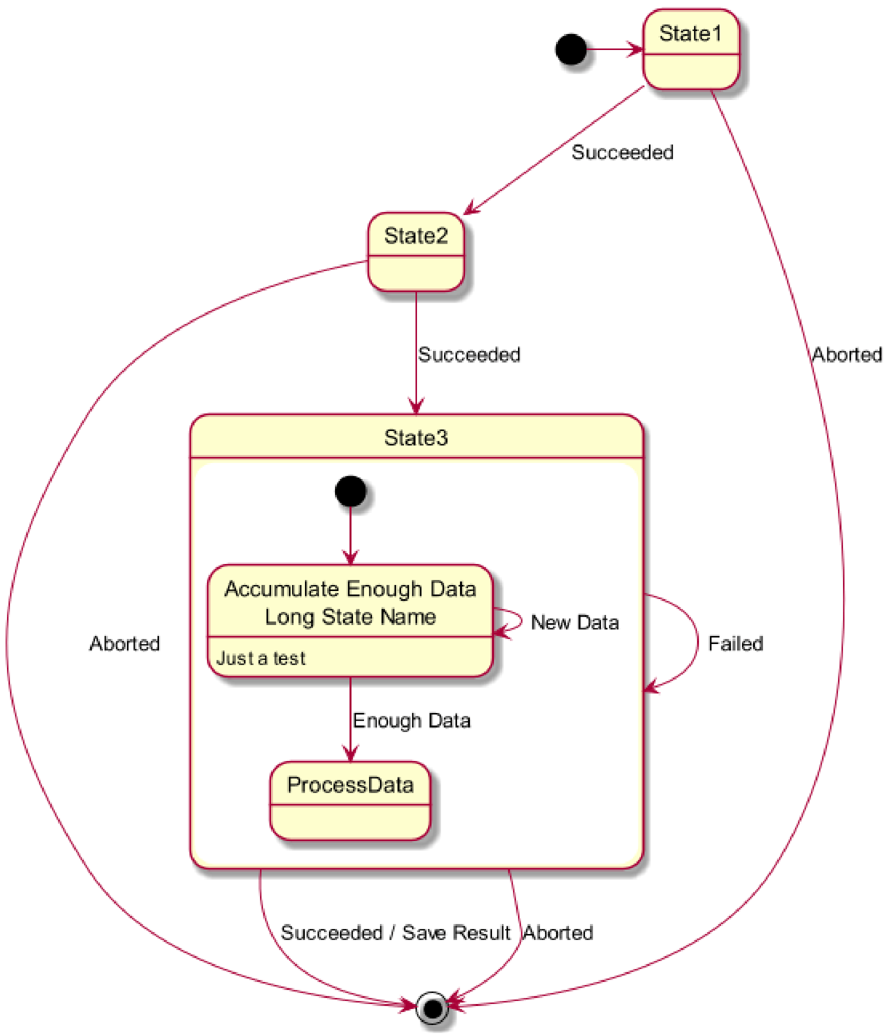


[Ref. QA-3300]

9.4 長い名前

キーワード state によって、状態についての長めの記述を使用することができます。

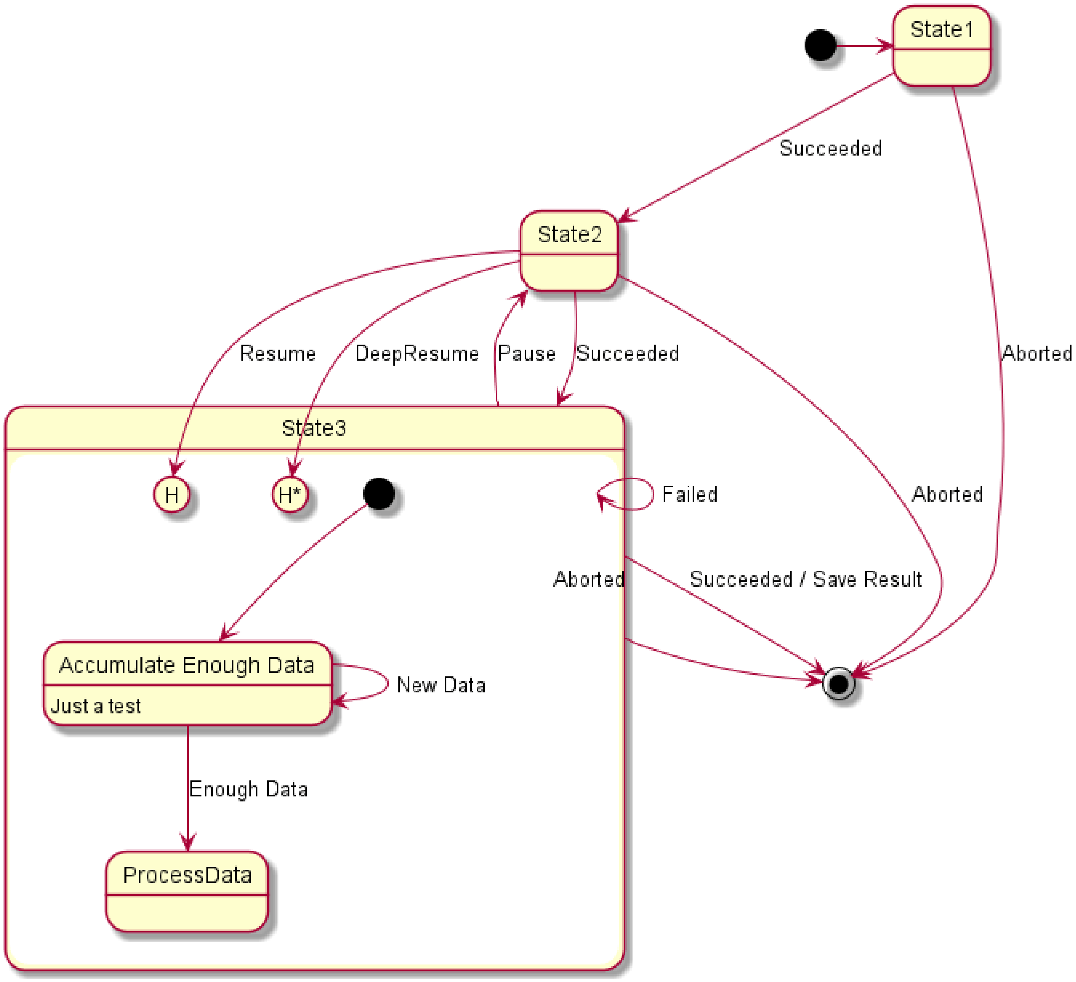
```
@startuml
scale 600 width

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

@enduml
```



9.5 履歴

[H] でサブ状態の履歴、[H*] でサブ状態の深い履歴 (deep history) を定義します。

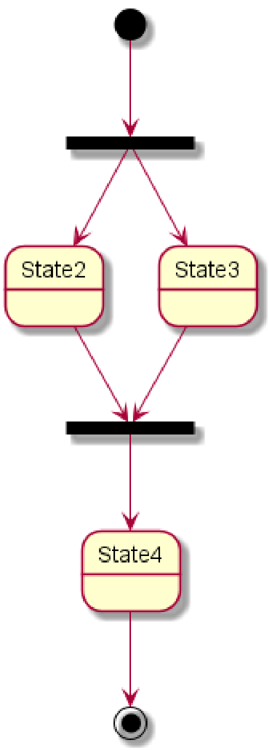
```
@startuml
[*] -> State1
State1 --> State2 : Succeeded
State1 --> [*] : Aborted
State2 --> State3 : Succeeded
State2 --> [*] : Aborted
state State3 {
  state "Accumulate Enough Data" as long1
  long1 : Just a test
  [*] --> long1
  long1 --> long1 : New Data
  long1 --> ProcessData : Enough Data
  State2 --> [H]: Resume
}
State3 --> State2 : Pause
State2 --> State3[H*]: DeepResume
State3 --> State3 : Failed
State3 --> [*] : Succeeded / Save Result
State3 --> [*] : Aborted
@enduml
```



9.6 フォーク (非同期実行)

<<fork>> と <<join>> stereotypes を使うことで、フォークノードとジョインノードを表せます。

```
@startuml

state fork_state <<fork>>
[*] --> fork_state
fork_state --> State2
fork_state --> State3

state join_state <<join>>
State2 --> join_state
State3 --> join_state
join_state --> State4
State4 --> [*]

@enduml
```

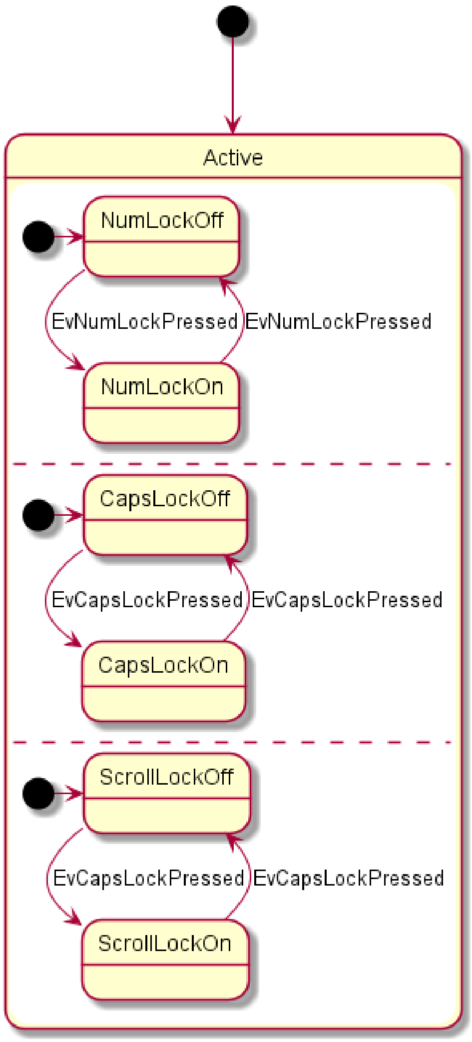


9.7 同時状態

記号--または || で分離することで、合成状態の中に同時状態を定義することができます。

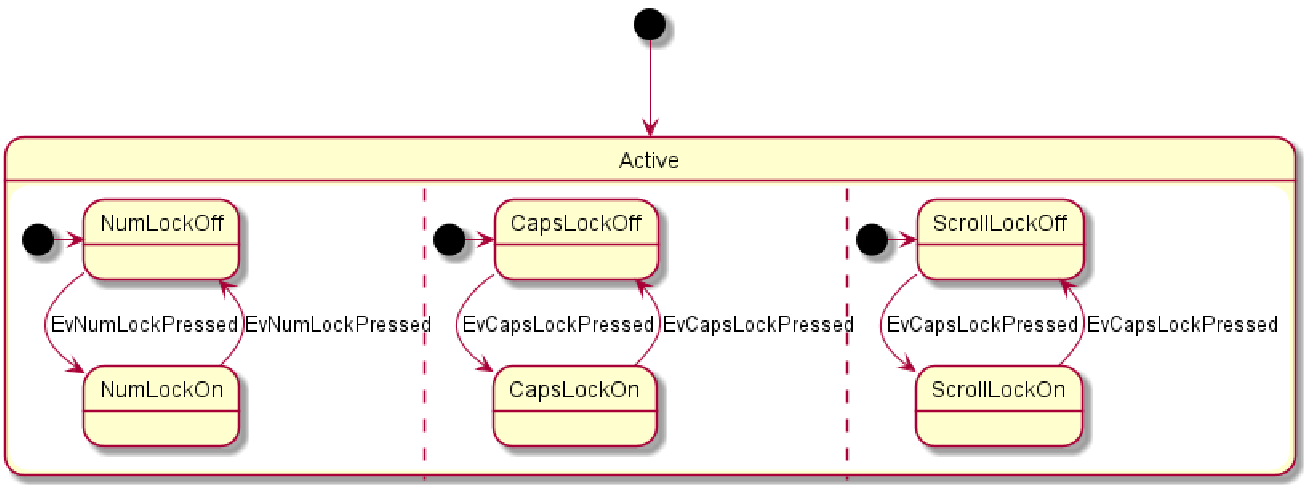
9.7.1 水平セパレーター--

```
@startuml
[*] --> Active

state Active {
  [*] -> NumLockOff
  NumLockOff --> NumLockOn : EvNumLockPressed
  NumLockOn --> NumLockOff : EvNumLockPressed
  --
  [*] -> CapsLockOff
  CapsLockOff --> CapsLockOn : EvCapsLockPressed
  CapsLockOn --> CapsLockOff : EvCapsLockPressed
  --
  [*] -> ScrollLockOff
  ScrollLockOff --> ScrollLockOn : EvCapsLockPressed
  ScrollLockOn --> ScrollLockOff : EvCapsLockPressed
}

@enduml
```



9.7.2 垂直セパレータ ||

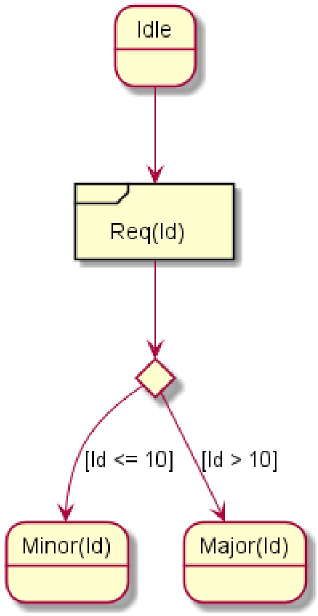
```
@startuml
[*] --> Active

state Active {
  [*] -> NumLockOff
  NumLockOff --> NumLockOn : EvNumLockPressed
  NumLockOn --> NumLockOff : EvNumLockPressed
  ||
  [*] -> CapsLockOff
  CapsLockOff --> CapsLockOn : EvCapsLockPressed
  CapsLockOn --> CapsLockOff : EvCapsLockPressed
  ||
  [*] -> ScrollLockOff
  ScrollLockOff --> ScrollLockOn : EvCapsLockPressed
  ScrollLockOn --> ScrollLockOff : EvCapsLockPressed
}

@enduml
```



9.8 条件

<<choice>> ステレオタイプで条件付きの状態を表すことができます。

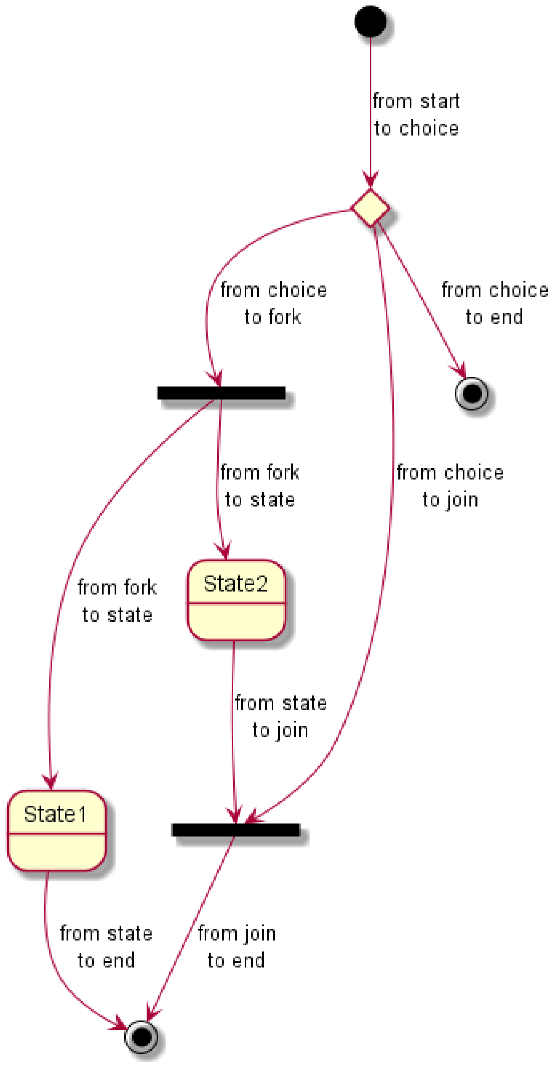
```
@startuml
state "Req(Id)" as ReqId <<sdlreceive>>
state "Minor(Id)" as MinorId
state "Major(Id)" as MajorId

state c <<choice>>

Idle --> ReqId
ReqId --> c
c --> MinorId : [Id <= 10]
c --> MajorId : [Id > 10]
@enduml
```



9.9 全ステレオタイプの例 (choice, fork, join, end)

```
@startuml
state choice1 <<choice>>
state fork1   <<fork>>
```

```
state join2   <<join>>
state end3    <<end>>

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

fork1   ---> State1 : from fork\nto state
fork1   --> State2  : from fork\nto state

State2  --> join2   : from state\nto join
State1  --> [*]     : from state\nto end

join2   --> [*]     : from join\nto end
@enduml
```



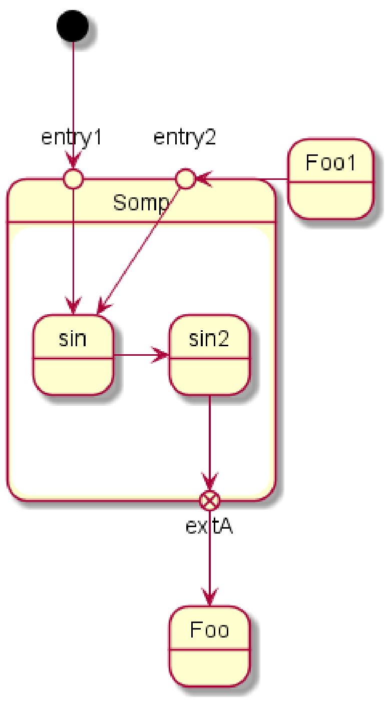
[Ref. QA-404 and QA-1159]

9.10 入場点と退場点

<<entryPoint>> ステレオタイプで入場点、<<exitPoint>> ステレオタイプで退場点を追加することができます。

```
@startuml
```

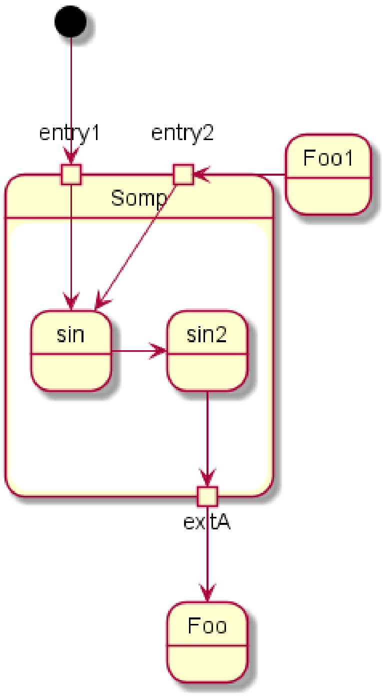
```
state Somp {
  state entry1 <<entryPoint>>
  state entry2 <<entryPoint>>
  state sin
  entry1 --> sin
  entry2 -> sin
  sin -> sin2
  sin2 --> exitA <<exitPoint>>
}

[*] --> entry1
exitA --> Foo
Foo1 -> entry2
@enduml
```



9.11 入力ピンと出力ピン

<<inputPin>> ステレオタイプで入力ピン、<<outputPin>> ステレオタイプで出力ピンを追加することができます。

```
@startuml
state Somp {
  state entry1 <<inputPin>>
  state entry2 <<inputPin>>
  state sin
  entry1 --> sin
  entry2 -> sin
  sin -> sin2
  sin2 --> exitA <<outputPin>>
}

[*] --> entry1
exitA --> Foo
Foo1 -> entry2
@enduml
```

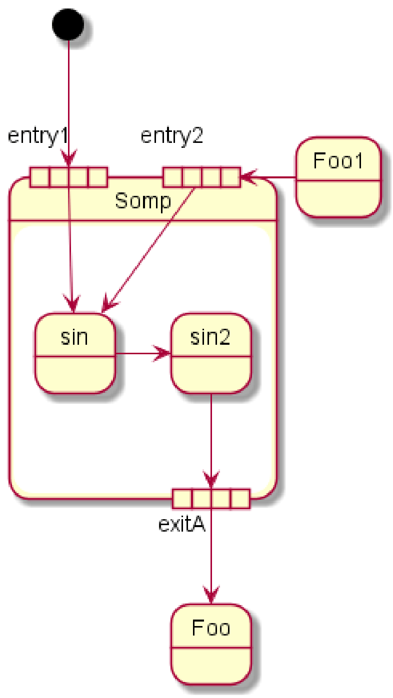


[Ref. QA-4309]

9.12 展開

<<expansionInput>> と <<expansionOutput>> ステレオタイプにより、展開 (expansion) を追加することができます。

```
@startuml
state Somp {
  state entry1 <<expansionInput>>
  state entry2 <<expansionInput>>
  state sin
  entry1 --> sin
  entry2 -> sin
  sin -> sin2
  sin2 --> exitA <<expansionOutput>>
}

[*] --> entry1
exitA --> Foo
Foo1 -> entry2
@enduml
```



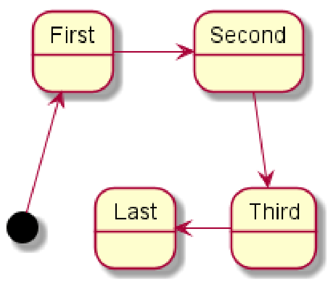
[Ref. QA-4309]

9.13 矢印の方向

記号-> を水平矢印として使用でき、以下の構文を使用することで、矢印の方向を指定することができます。

- `-down->` or `-->`
- `-right->` or `->` (デフォルトの矢印)
- `-left->`
- `-up->`

```
@startuml

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

@enduml
```



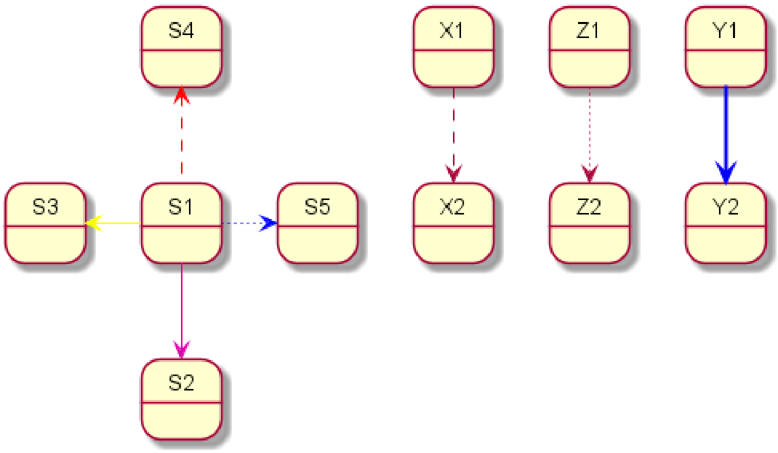
方向を示す単語の、最初の文字だけ（例：`-down-`の代わりに`-d-`）、または 2 文字（`-do-`）を使用することで、矢印の記述を短くすることができます。

この機能を乱用しないよう注意しなくてはいけません：通常、*Graphviz* は微調整なしでよい結果をもたらしてくれます。

9.14 線の色とスタイルを変更する

線の色とスタイルを変更することができます。

```
@startuml
State S1
State S2
S1 -[#DD00AA]-> S2
S1 -left[#yellow]-> S3
S1 -up[#red,dashed]-> S4
S1 -right[dotted,#blue]-> S5

X1 -[dashed]-> X2
Z1 -[dotted]-> Z2
Y1 -[#blue,bold]-> Y2
@enduml
```



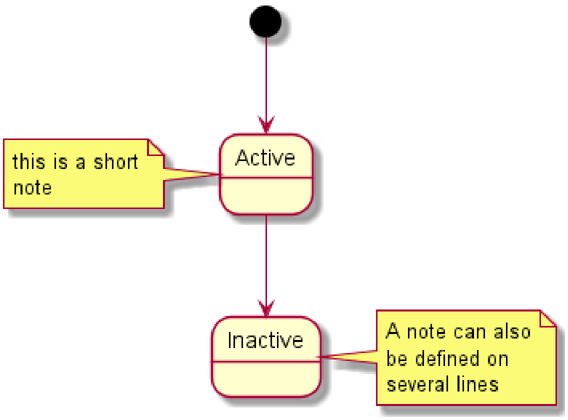
[Ref. Incubation: Change line color in state diagrams]

9.15 注釈

キーワード note left of, note right of, note top of, note bottom of を使用して注釈を定義することができます。

また、複数行の注釈を定義することもできます。

```
@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
```



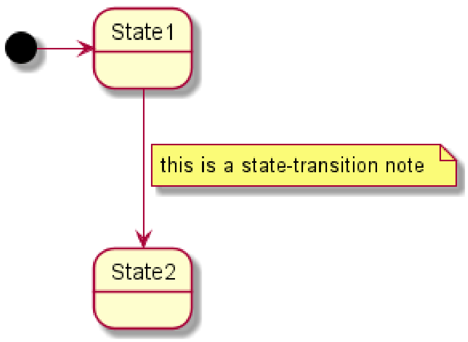
さらに、状態にひもづかない注釈を定義できます。

```
@startuml

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

@enduml
```

9.16 リンクへの注釈

note on link キーワードで、状態遷移(リンク)に注釈を追加することができます。

```
@startuml
[*] -> State1
State1 --> State2
note on link
  this is a state-transition note
end note
@enduml
```

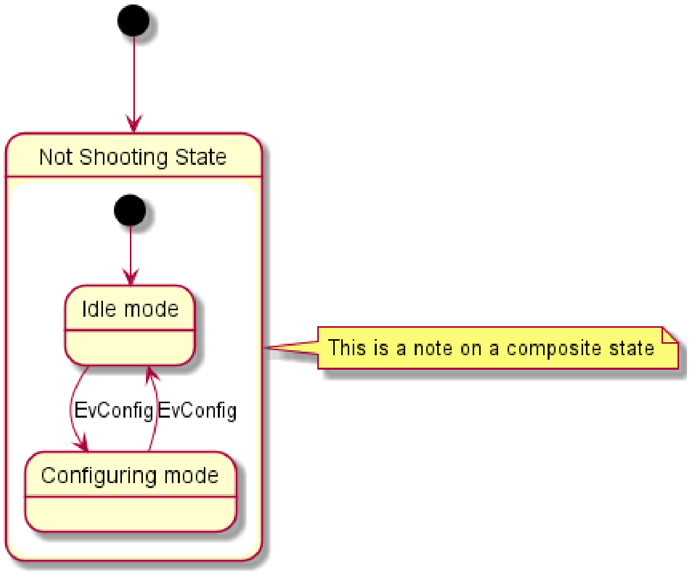


9.17 その他の注釈

合成状態にも注釈をつけることができます。

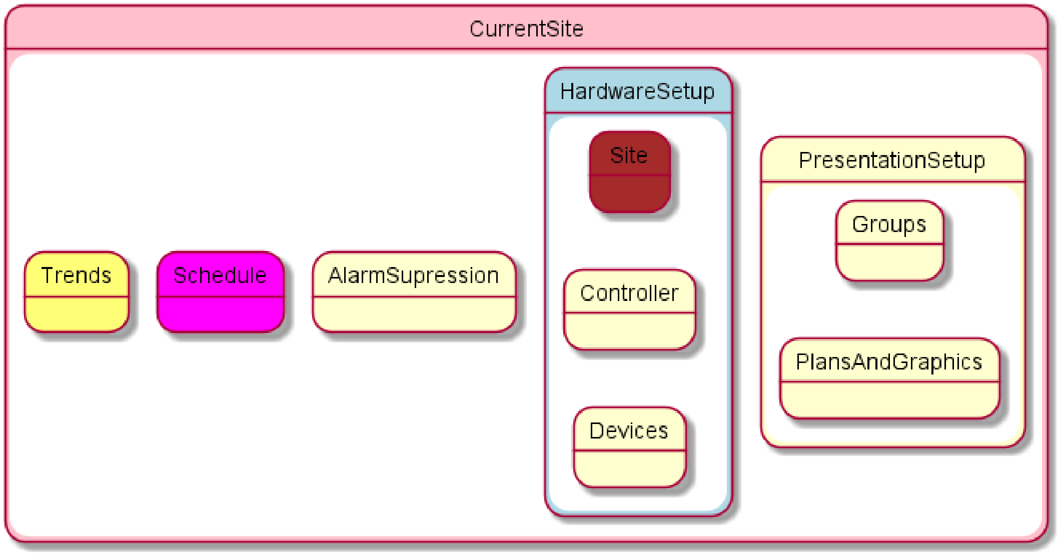
```
@startuml

[*] --> NotShooting

state "Not Shooting State" as NotShooting {
```

```
  state "Idle mode" as Idle
  state "Configuring mode" as Configuring
  [*] --> Idle
  Idle --> Configuring : EvConfig
  Configuring --> Idle : EvConfig
}

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

@enduml
```



9.18 インライン色指定

```
@startuml
state CurrentSite #pink {
    state HardwareSetup #lightblue {
       state Site #brown
        Site -[hidden]-> Controller
        Controller -[hidden]-> Devices
    }
    state PresentationSetup{
        Groups -[hidden]-> PlansAndGraphics
    }
    state Trends #FFFF77
    state Schedule #magenta
    state AlarmSupression
}
@enduml
```



[Ref. QA-1812]

9.19 見栄え

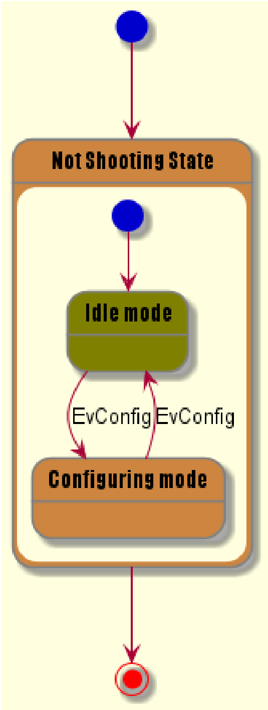
ダイアグラムの色やフォントを変更するには skinparam コマンドを使用します。

このコマンドは以下の場面で使用できます。

- ダイアグラム定義内で他のコマンドを同様に。
- インクルードされたファイル内。
- 設定ファイルのコマンドライン内や ANT タスク内。

定型化した状態に、特定の色とフォントを定義することができます。

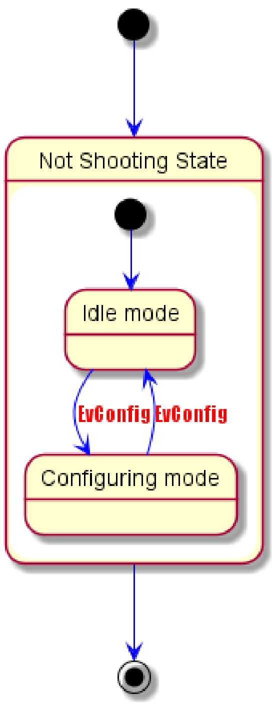
```
@startuml
skinparam backgroundColor LightYellow
skinparam state {
  StartColor MediumBlue
  EndColor Red
  BackgroundColor Peru
  BackgroundColor<<Warning>> Olive
  BorderColor Gray
  FontName Impact
}

[*] --> NotShooting

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

NotShooting --> [*]
@enduml
```



9.20 スタイル変更

スタイルを変更できます。

```
@startuml

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


[*] --> NotShooting

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

NotShooting --> [*]
@enduml
```



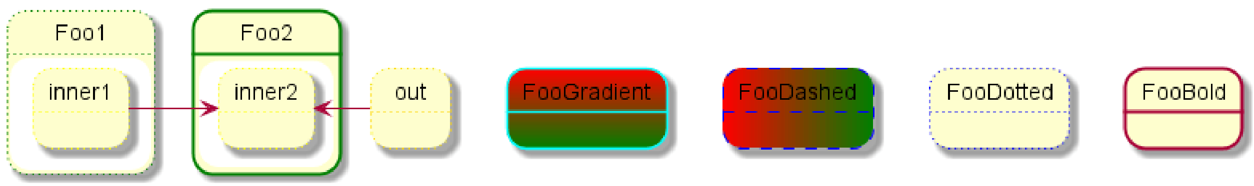
9.21 Change state color and style (inline style)

You can change the color or style of individual state using the following notation:

- `#color ##[style]color`

With background color first (`#color`), then line style and line color (`##[style]color`).

```
@startuml
state FooGradient #red-green ##00FFFF
state FooDashed #red|green ##[dashed]blue {
}
state FooDotted ##[dotted]blue {
}
state FooBold ##[bold] {
}
state Foo1 ##[dotted]green {
state inner1 ##[dotted]yellow
}

state out ##[dotted]gold

state Foo2 ##[bold]green {
state inner2 ##[dotted]yellow
}
inner1 -> inner2
out -> inner2
@enduml
```

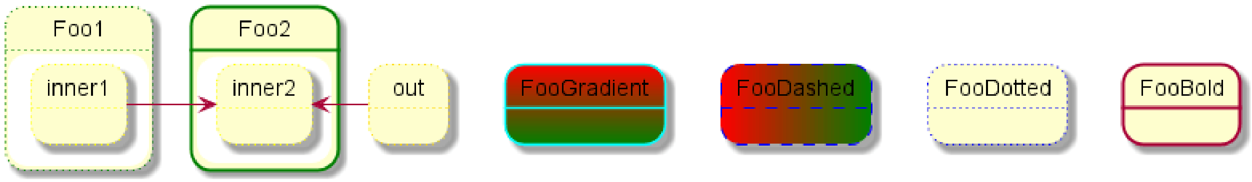


[Ref. QA-1487]

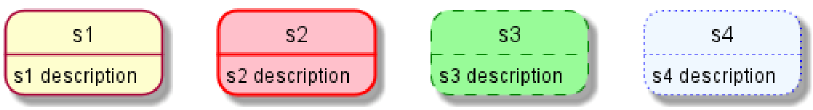
- `#color;line:color;line.[bold|dashed|dotted];text:color`

TODO: FIXME `text:color` seems not to be taken into account **TODO:** FIXME

```
@startuml
@startuml
state FooGradient #red-green;line:00FFFF
state FooDashed #red|green;line.dashed;line:blue {
}
state FooDotted #line.dotted;line:blue {
}
state FooBold #line.bold {
}
state Foo1 #line.dotted;line:green {
state inner1 #line.dotted;line:yellow
}

state out #line.dotted;line:gold

state Foo2 #line.bold;line:green {
state inner2 #line.dotted;line:yellow
}
inner1 -> inner2
out -> inner2
@enduml
@enduml
```




```
@startuml
state s1 : s1 description
state s2 #pink;line:red;line.bold;text:red : s2 description
state s3 #palegreen;line:green;line.dashed;text:green : s3 description
state s4 #aliceblue;line:blue;line.dotted;text:blue   : s4 description
@enduml
```



[Adapted from QA-3770]

10 タイミング図

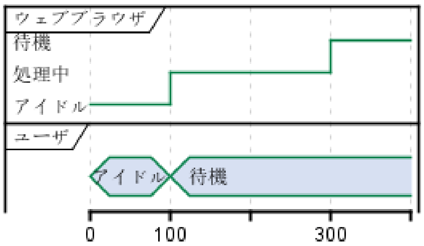
現在、このダイアグラムは提案段階です。将来的に変更されるかもしれません。 新しいシンタックス案の提案を歓迎します！よりよい形を模索するのに、あなたからの意見や提案が役に立ちます！!

10.1 ライフライン

ライフラインは、concise か robust で定義できます。concise は状態ライフラインを、robust は汎用値ライフラインを、それぞれ作成します。

@ と is を用いて、状態の変化を記述できます。

```
@startuml
robust "ウェブブラウザ" as WB
concise "ユーザ" as WU

@0
WU is アイドル
WB is アイドル

@100
WU is 待機
WB is 処理中

@300
WB is 待機
@enduml
```

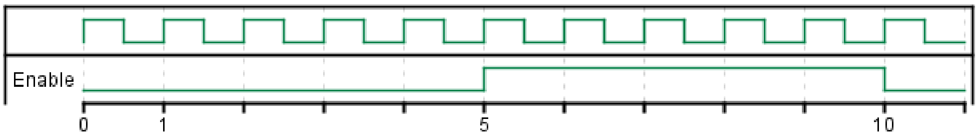


10.2 Binary and Clock

It's also possible to have binary and clock signal, using the following keywords:

- binary
- clock

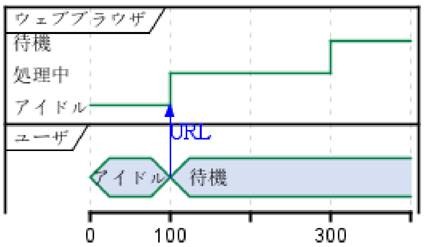
```
@startuml
clock clk with period 1
binary "Enable" as EN

@0
EN is low

@5
EN is high

@10
EN is low
@enduml
```



10.3 メッセージ（相互作用）

メッセージは、矢印構文を使います。

```
@startuml
robust "ウェブブラウザ" as WB
concise "ユーザ" as WU

@0
WU is アイドル
WB is アイドル

@100
WU -> WB : URL
WU is 待機
WB is 処理中

@300
WB is 待機
@enduml
```

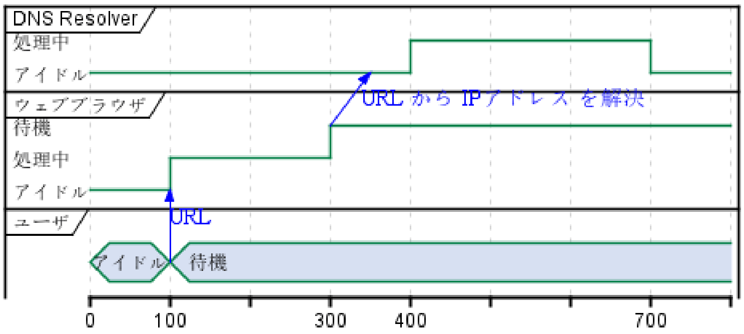


10.4 相対時間での指定

@ で時間を指定するとき、相対的な時間の指定の仕方ができます。

```
@startuml
robust "DNS Resolver" as DNS
robust "ウェブブラウザ" as WB
concise "ユーザ" as WU

@0
WU is アイドル
WB is アイドル
DNS is アイドル

@+100
WU -> WB : URL
WU is 待機
WB is 処理中

@+200
WB is 待機
WB -> DNS@+50 : URL から IPアドレス を解決
```

```
@+100
DNS is 処理中

@+300
DNS is アイドル
@enduml
```

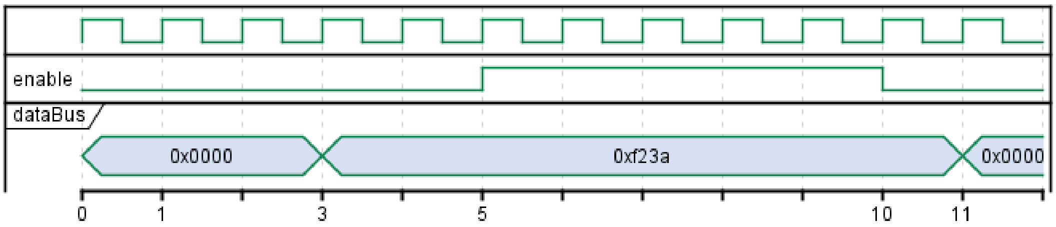


10.5 Anchor Points

Instead of using absolute or relative time on an absolute time you can define a time as an anchor point by using the `as` keyword and starting the name with a `:`.

```
@XX as :<anchor point name>
```

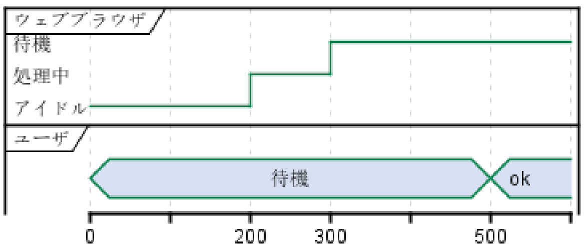
```
@startuml
clock clk with period 1
binary "enable" as EN
concise "dataBus" as db

@0 as :start
@5 as :en_high
@10 as :en_low


@:start
EN is low
db is "0x0000"

@:en_high
EN is high

@:en_low
EN is low

@:en_high-2
db is "0xf23a"

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



10.6 インスタンス指向

時系列順での定義ではなく、インスタンス毎 (ライフライン毎) に定義できます。

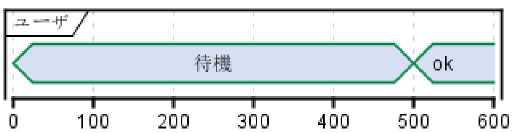
```
@startuml
robust "ウェブブラウザ" as WB
concise "ユーザ" as WU

@WB
0 is アイドル
+200 is 処理中
+100 is 待機

@WU
0 is 待機
+500 is ok
@enduml
```



10.7 スケールの設定

スケール (メモリの数値の表示) を指定できます。以下の例では、「メモリを 100 ずつ表示、1 メモリの幅を 50px にする」設定になります。

```
@startuml
concise "ユーザ" as WU
scale 100 as 50 pixels

@WU
0 is 待機
+500 is ok
@enduml
```

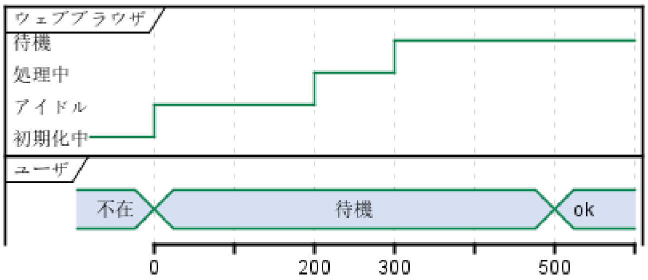


10.8 初期状態

「初期状態」を設定できます。

```
@startuml
```

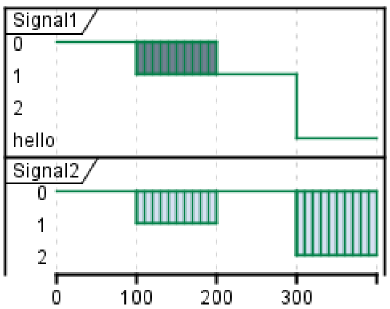
```
robust "ウェブブラウザ" as WB
concise "ユーザ" as WU

WB is 初期化中
WU is 不在

@WB
0 is アイドル
+200 is 処理中
+100 is 待機

@WU
0 is 待機
+500 is ok
@enduml
```



10.9 複雑な状態

信号をいくつかの不定状態とすることができます。

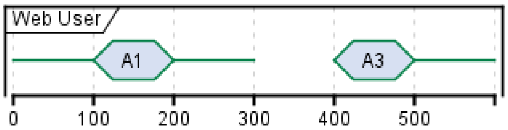
```
@startuml
robust "Signal1" as S1
robust "Signal2" as S2
S1 has 0,1,2,hello
S2 has 0,1,2
@0
S1 is 0
S2 is 0
@100
S1 is {0,1} #SlateGrey
S2 is {0,1}
@200
S1 is 1
S2 is 0
@300
S1 is hello
S2 is {0,2}
@enduml
```



10.10 状態の非表示

いくつかの状態を非表示にすることもできます。

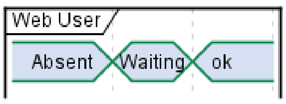
```
@startuml
concise "Web User" as WU

@0
WU is {-}

@100
WU is A1

@200
WU is {-}

@300
WU is {hidden}

@400
WU is A3

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



10.11 Hide time axis

It is possible to hide time axis.

```
@startuml
hide time-axis
concise "Web User" as WU

WU is Absent

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

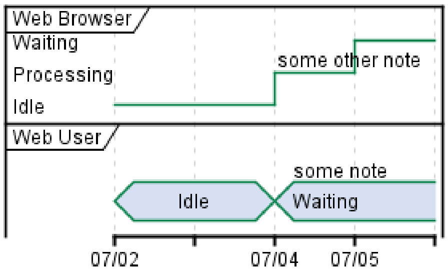


10.12 Using Time and Date

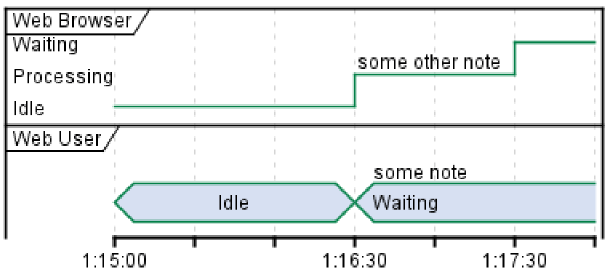
It is possible to use time or date.

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

@2019/07/02
WU is Idle
```

```
WB is Idle

@2019/07/04
WU is Waiting : some note
WB is Processing : some other note

@2019/07/05
WB is Waiting
@enduml
```




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

@1:15:00
WU is Idle
WB is Idle

@1:16:30
WU is Waiting : some note
WB is Processing : some other note

@1:17:30
WB is Waiting
@enduml
```

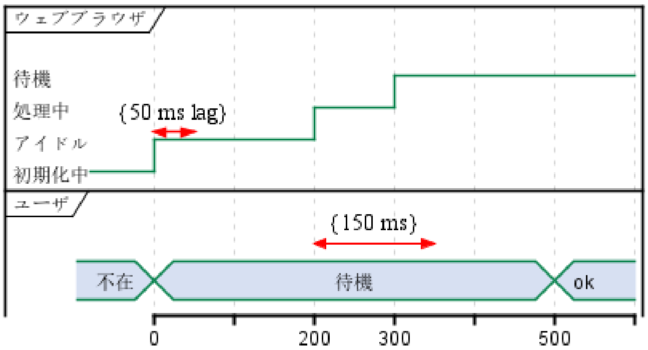


10.13 時間定規（ time constraint ）の追加

タイムラインのメモリとは別に、時間の尺度を示す矢印を表示することができます。

```
@startuml
robust "ウェブブラウザ" as WB
concise "ユーザ" as WU

WB is 初期化中
WU is 不在

@WB
0 is アイドル
+200 is 処理中
```

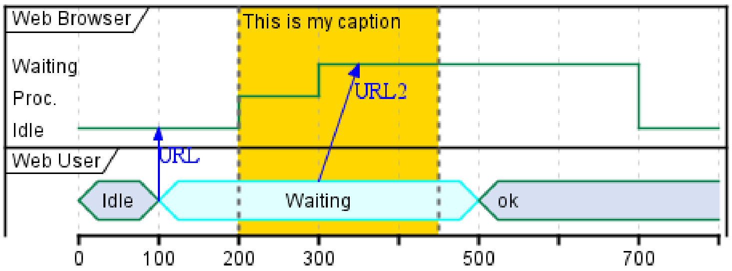
```
+100 is 待機
WB@0 <-> @50 : {50 ms lag}

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



10.14 Highlighted period

You can higlight a part of diagram.

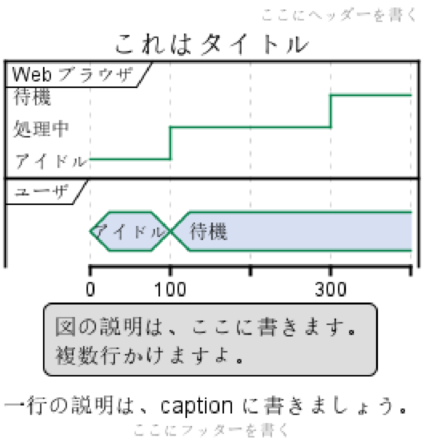
```
@startuml
robust "Web Browser" as WB
concise "Web User" as WU

@0
WU is Idle
WB is Idle

@100
WU -> WB : URL
WU is Waiting #LightCyan;line:Aqua

@200
WB is Proc.

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

@+200
WU is ok

@+200
WB is Idle

highlight 200 to 450 #Gold;line:DimGrey : This is my caption
@enduml
```



10.15 タイトルなどを追加する

(他の UML ダイアグラムと同様に) タイトル、ヘッダー / フッター、説明文、キャプションを書くことができます。

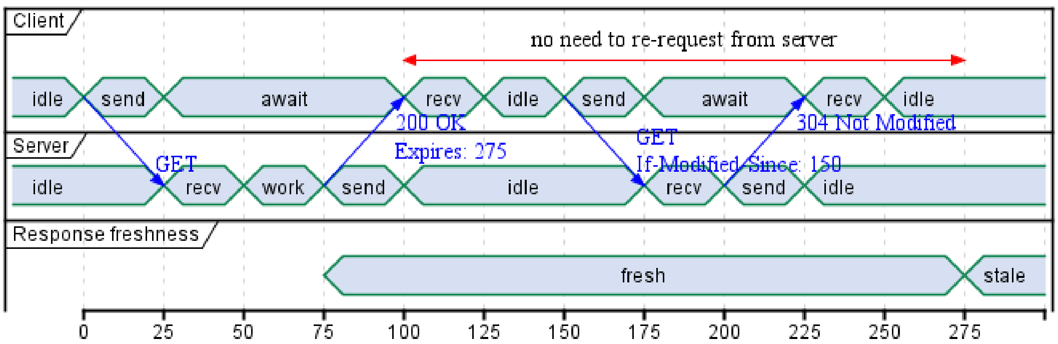
```
@startuml
Title これはタイトル
header: ここにヘッダーを書く
footer: ここにフッターを書く
legend
図の説明は、ここに書きます。
複数行かけますよ。
end legend
caption 一行の説明は、caption に書きましょう。

robust "Web ブラウザ" as WB
concise "ユーザ" as WU

@0
WU is アイドル
WB is アイドル

@100
WU is 待機
WB is 処理中

@300
WB is 待機
@enduml
```



10.16 Complete example

Thanks to Adam Rosien for this example.

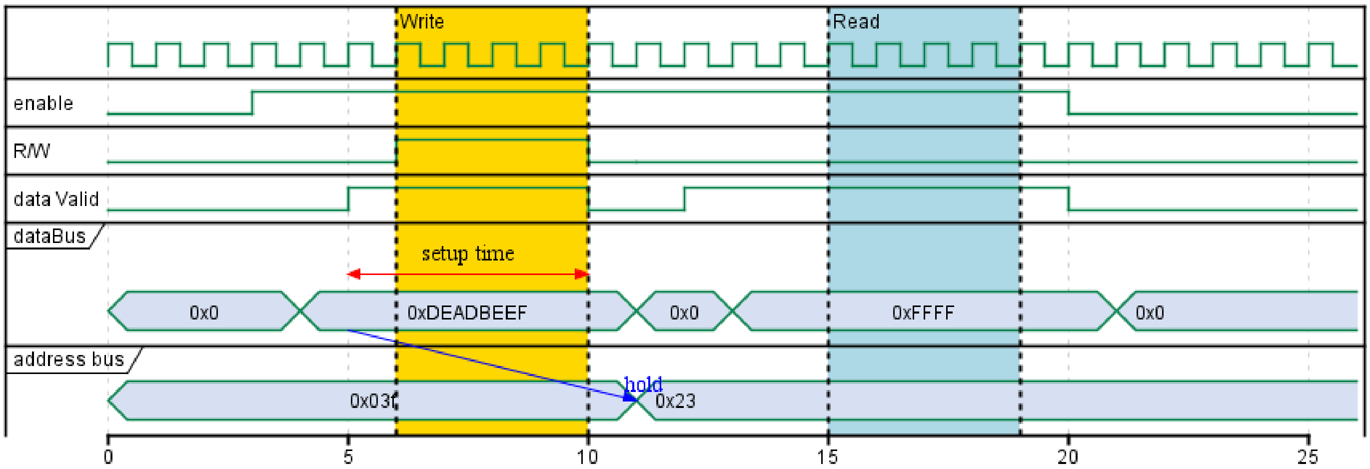
```
@startuml
concise "Client" as Client
concise "Server" as Server
concise "Response freshness" as Cache

Server is idle
Client is idle

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

@Server
25 is recv
+25 is work
+25 is send
Server -> Client@+25 : 200 OK\nExpires: 275
+25 is idle
+75 is recv
+25 is send
Server -> Client@+25 : 304 Not Modified
+25 is idle

@Cache
75 is fresh
+200 is stale
@enduml
```



10.17 Digital Example

```
@startuml
scale 5 as 150 pixels

clock clk with period 1
binary "enable" as en
binary "R/W" as rw
binary "data Valid" as dv
```

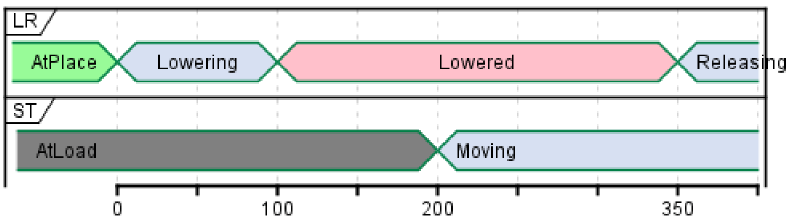
```
concise "dataBus" as db
concise "address bus" as addr

@6 as :write_beg
@10 as :write_end

@15 as :read_beg
@19 as :read_end


@0
en is low
db is "0x0"
addr is "0x03f"
rw is low
dv is 0

@:write_beg-3
 en is high
@:write_beg-2
 db is "0xDEADBEEF"
@:write_beg-1
dv is 1
@:write_beg
rw is high


@:write_end
rw is low
dv is low
@:write_end+1
rw is low
db is "0x0"
addr is "0x23"

@12
dv is high
@13
db is "0xFFFF"

@20
en is low
dv is low
@21
db is "0x0"

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

db@:write_beg-1 <-> @:write_end : setup time
db@:write_beg-1 -> addr@:write_end+1 : hold
@enduml
```



10.18 Adding color

You can add color.

```
@startuml
concise "LR" as LR
concise "ST" as ST

LR is AtPlace #palegreen
ST is AtLoad #gray

@LR
0 is Lowering
100 is Lowered #pink
350 is Releasing

@ST
200 is Moving
@enduml
```



[Ref. QA-5776]

11 JSON データを表示する

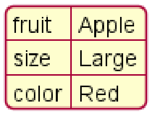
JSON 形式は、様々なソフトウエアで使われています。

PlantUML を使って JSON データを可視化することができます。

この機能を使うには、

- `@startjson` キーワードで開始し、
- `@endjson` キーワードで終了する必要があります。

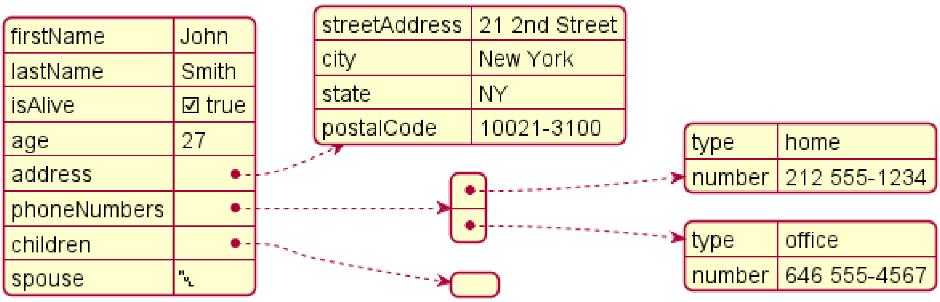
```
@startjson
{
   "fruit":"Apple",
   "size":"Large",
   "color":"Red"
}
@endjson
```



11.1 複雑な例

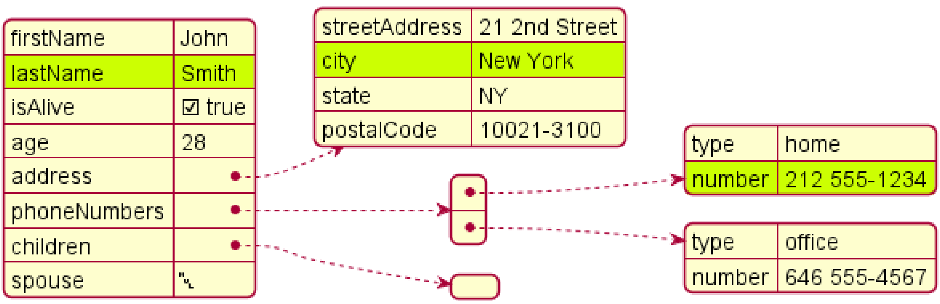
複雑な JSON 構造を使用することができます。

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



11.2 一部をハイライトする

```
@startjson
#highlight "lastName"
#highlight "address" / "city"
#highlight "phoneNumbers" / "0" / "number"
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 28,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
@endjson
```

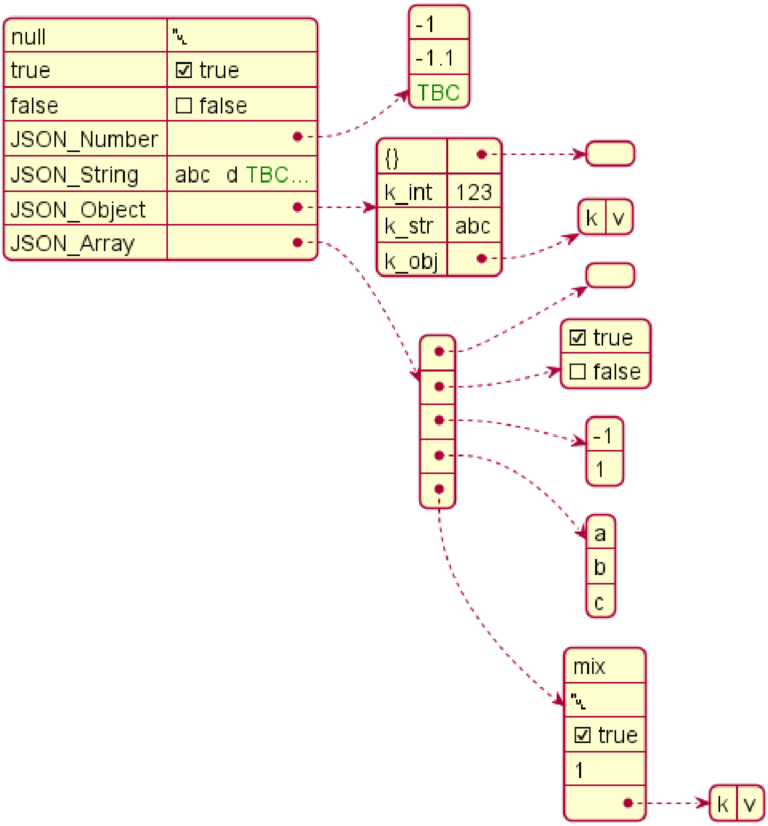


11.3 JSON の基本要素

11.3.1 すべての JSON 基本要素の例

```
@startjson
```

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

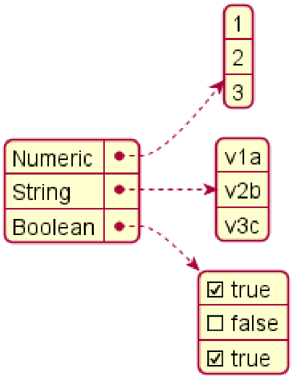


11.4 JSON 配列またはテーブル

11.4.1 配列型

```
@startjson
{
"Numeric": [1, 2, 3],
"String ": ["v1a", "v2b", "v3c"],
"Boolean": [true, false, true]
}
```

```
@endjson
```



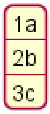
11.4.2 シンプルな配列またはテーブル

11.4.3 Number 配列

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

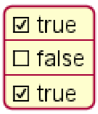
11.4.4 String 配列

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



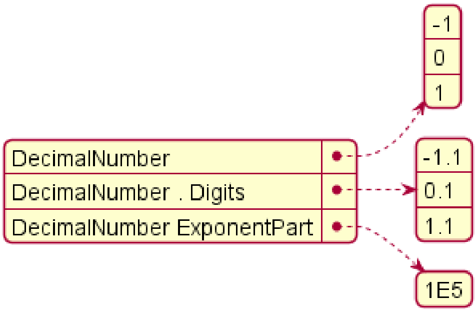
11.4.5 Boolean 配列

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



11.5 number 型

```
@startjson
{
"DecimalNumber": [-1, 0, 1],
"DecimalNumber . Digits": [-1.1, 0.1, 1.1],
"DecimalNumber ExponentPart": [1E5]
}
@endjson
```

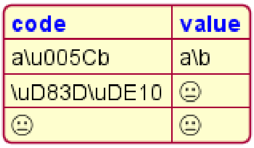


11.6 string 型

11.6.1 Unicode

JSON では Unicode を直接記述するか、のような形式でエスケープして記述することができます。

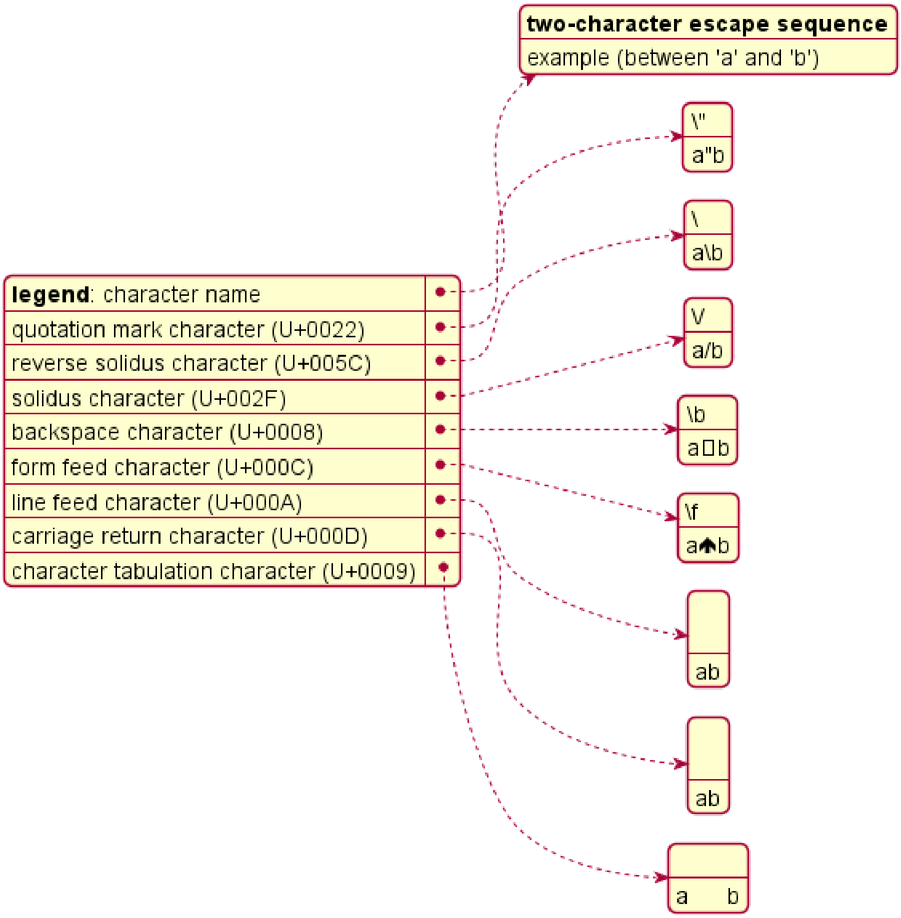
```
@startjson
{
  "<color:blue><b>code":  "<color:blue><b>value",
  "a\\u005Cb":            "a\u005Cb",
  "\\uD83D\\uDE10":       "\uD83D\uDE10",
  " ":                   " "
}
@endjson
```



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

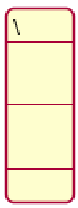
11.6.2 2 文字のエスケープシーケンス

```
@startjson
{
 "**legend**: character name":               ["**two-character escape sequence**", "example (between
 "quotation mark character (U+0022)":        ["\\\"", "a\"b"],
 "reverse solidus character (U+005C)":       ["\\\\", "a\\b"],
 "solidus character (U+002F)":               ["\\\/", "a\/b"],
 "backspace character (U+0008)":             ["\\b", "a\bb"],
 "form feed character (U+000C)":             ["\\f", "a\fb"],
 "line feed character (U+000A)":             ["\\n", "a\nb"],
 "carriage return character (U+000D)":       ["\\r", "a\rb"],
 "character tabulation character (U+0009)":  ["\\t", "a\tb"]
}
@endjson
```



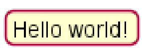
TODO: FIXME FIXME or not , on the same item as management in PlantUML **TODO:** FIXME

```
@startjson
[
"\\\\",
"\\n",
"\\r",
"\\t"
]
@endjson
```



11.7 最小の JSON の例

```
@startjson
"Hello world!"
@endjson
```




```
@startjson
```

```
42
@endjson
```

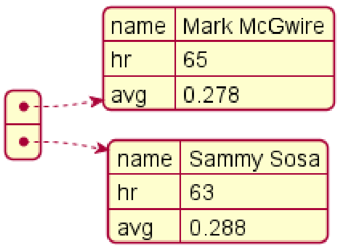

```
@startjson
true
@endjson
```

(Examples come from STD 90 - Examples)

11.8 スタイルを使用する

11.8.1 スタイル無し（デフォルト）

```
@startjson
[
  {
    "name": "Mark McGwire",
    "hr":   65,
    "avg":  0.278
  },
  {
    "name": "Sammy Sosa",
    "hr":   63,
    "avg":  0.288
  }
]
@endjson
```

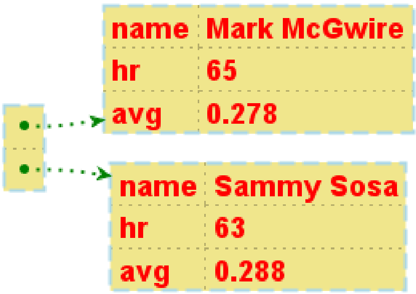


11.8.2 スタイル有り

スタイルを指定して、要素の見た目を変更することができます。

```
@startjson
<style>
jsonDiagram {
  node {
    BackGroundColor Khaki
    LineColor lightblue
    FontName Helvetica
    FontColor red
    FontSize 18
    FontStyle bold
    RoundCorner 0
    LineThickness 2
    LineStyle 10;5
    separator {
```

```
      LineThickness 0.5
      LineColor black
      LineStyle 1;5
    }
  }
  arrow {
    BackGroundColor lightblue
    LineColor green
    LineThickness 2
    LineStyle 2;5
  }
}
</style>
[
  {
    "name": "Mark McGwire",
    "hr":   65,
    "avg":  0.278
  },
  {
    "name": "Sammy Sosa",
    "hr":   63,
    "avg":  0.288
  }
]
@endjson
```



[Adapted from QA-13123]

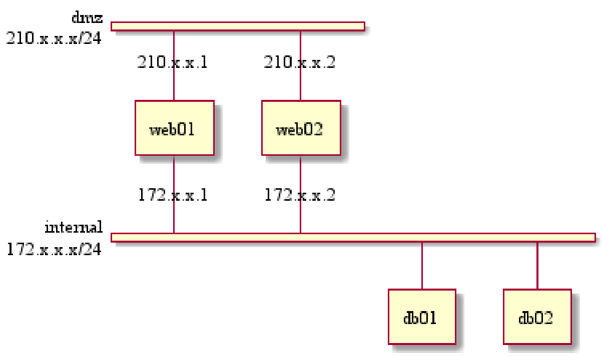
12 ネットワーク図 (nwdiag)

nwdiag は、小宮健 (Takeshi KOMIYA) によって作られた、ネットワーク図を生成するためのツールです。

その文法はわかりやすくシンプルなので、PlantUML にも組み込まれました。以下に、公式サイトのサンプルと同等の図を載せています。

12.1 シンプルなネットワーク図

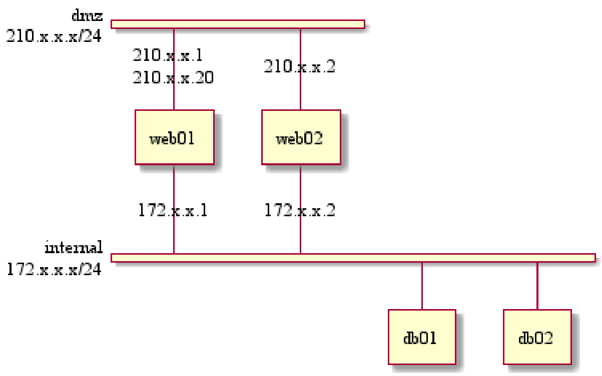
```
@startuml
nwdiag {
  network dmz {
      address = "210.x.x.x/24"

      web01 [address = "210.x.x.1"];
      web02 [address = "210.x.x.2"];
  }
  network internal {
      address = "172.x.x.x/24";

      web01 [address = "172.x.x.1"];
      web02 [address = "172.x.x.2"];
      db01;
      db02;
  }
}
@enduml
```



12.2 複数のアドレスを定義するケース

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

      // set multiple addresses (using comma)
      web01 [address = "210.x.x.1, 210.x.x.20"];
      web02 [address = "210.x.x.2"];
  }
  network internal {
      address = "172.x.x.x/24";

      web01 [address = "172.x.x.1"];
      web02 [address = "172.x.x.2"];
      db01;
```

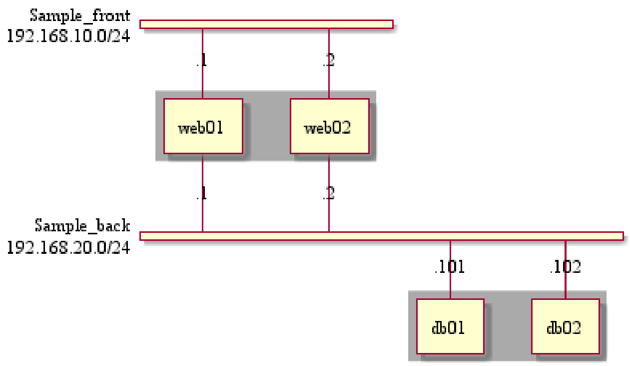
```
    db02;
  }
}
@enduml
```



12.3　ノードのグルーピング

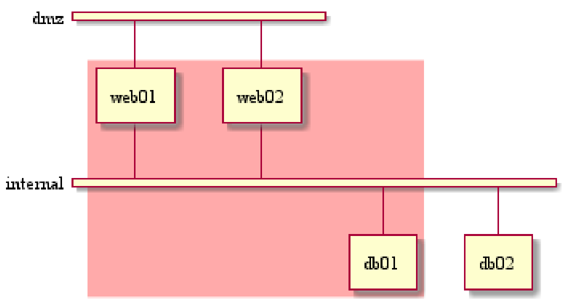
12.3.1　ネットワーク定義の中でグループを定義

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

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

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



12.3.2 ネットワーク定義の外でグループを定義

```
@startuml
nwdiag {
  // define group outside of network definitions
  group {
    color = "#FFAAAA";

    web01;
    web02;
    db01;
  }

  network dmz {
    web01;
    web02;
  }
  network internal {
    web01;
    web02;
    db01;
    db02;
  }
}
@enduml
```

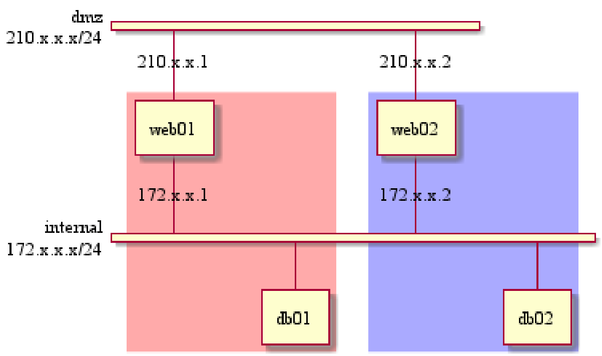


12.3.3 一つのネットワークに複数のグループを定義

12.3.4 グループが 2 つの例

```
@startuml
nwdiag {
  group {
```

```
    color = "#FFaaaa";
    web01;
    db01;
  }
  group {
    color = "#aaaaFF";
    web02;
    db02;
  }
  network dmz {
      address = "210.x.x.x/24"

      web01 [address = "210.x.x.1"];
      web02 [address = "210.x.x.2"];
  }
  network internal {
      address = "172.x.x.x/24";

      web01 [address = "172.x.x.1"];
      web02 [address = "172.x.x.2"];
      db01 ;
      db02 ;
  }
}
@enduml
```

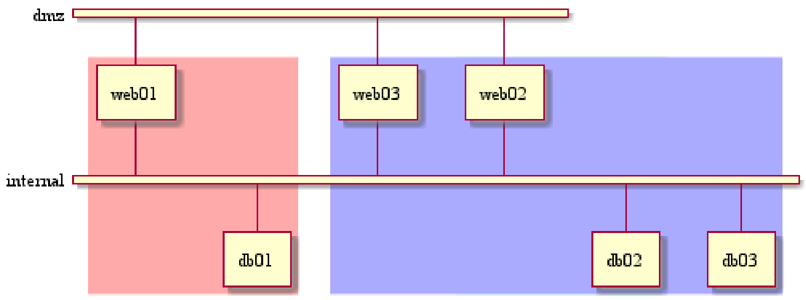


[Ref. QA-12663]

12.3.5 グループが 3 つの例

```
@startuml
nwdiag {
  group {
    color = "#FFaaaa";
    web01;
    db01;
  }
  group {
    color = "#aaFFaa";
    web02;
    db02;
  }
  group {
    color = "#aaaaFF";
    web03;
    db03;
```

```
  }

  network dmz {
      web01;
      web02;
      web03;
  }
  network internal {
      web01;
      db01 ;
      web02;
      db02 ;
      web03;
      db03;
  }
}
@enduml
```



[Ref. QA-13138]

12.4 ネットワークとグループに対する拡張文法

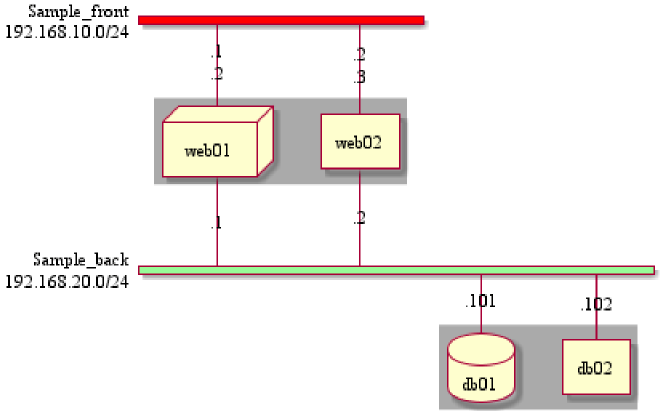
12.4.1 ネットワーク

ネットワーク、ネットワーク要素に対して、次の項目が設定可能です：

- アドレス (コンマ, 区切り)
- 色
- 説明
- 形状

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

    // define group
    group web {
      web01 [address = ".1, .2", shape = "node"]
      web02 [address = ".2, .3"]
    }
  }
  network Sample_back {
    address = "192.168.20.0/24"
    color = "palegreen"
    web01 [address = ".1"]
```

```
    web02 [address = ".2"]
    db01 [address = ".101", shape = database ]
    db02 [address = ".102"]

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

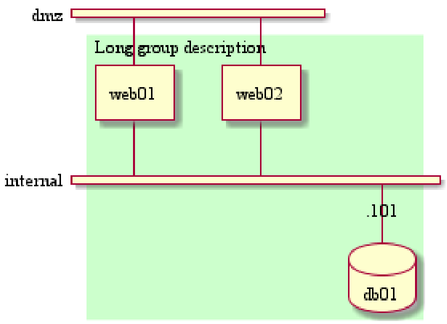


12.4.2　グループ

グループに対して、次の項目が設定可能です：

- 色
- 説明

```
@startuml
nwdiag {
  group {
    color = "#CCFFCC";
    description = "Long group description";

    web01;
    web02;
    db01;
  }

  network dmz {
    web01;
    web02;
  }
  network internal {
    web01;
    web02;
    db01 [address = ".101", shape = database];
  }
}
@enduml
```



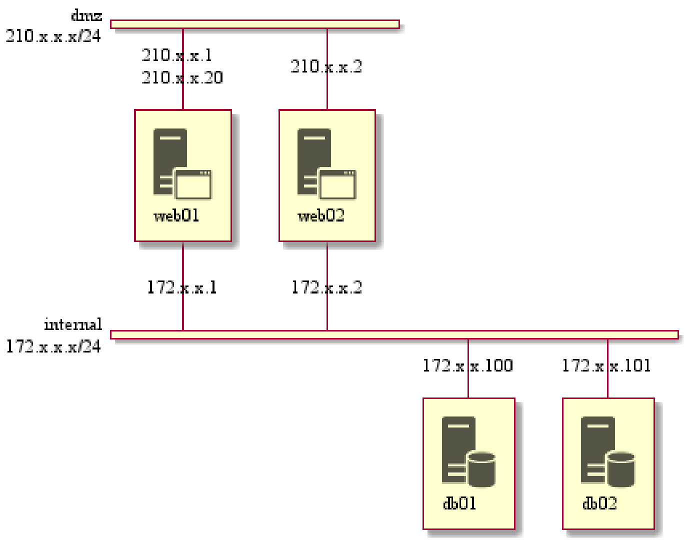
[Ref. QA-12056]

12.5 スプライトの使用

標準ライブラリやその他のライブラリに含まれる、あらゆるスプライト (アイコン) を使用できます。

スプライトは <$sprite> の記法を使います。で改行できます。また、その他の Creole 記法も使用できます。

```
@startuml
!include <office/Servers/application_server>
!include <office/Servers/database_server>

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

      // set multiple addresses (using comma)
      web01 [address = "210.x.x.1, 210.x.x.20",  description = "<$application_server>\n web01"]
      web02 [address = "210.x.x.2",  description = "<$application_server>\n web02"];
  }
  network internal {
      address = "172.x.x.x/24";

      web01 [address = "172.x.x.1"];
      web02 [address = "172.x.x.2"];
      db01 [address = "172.x.x.100",  description = "<$database_server>\n db01"];
      db02 [address = "172.x.x.101",  description = "<$database_server>\n db02"];
  }
}
@enduml
```



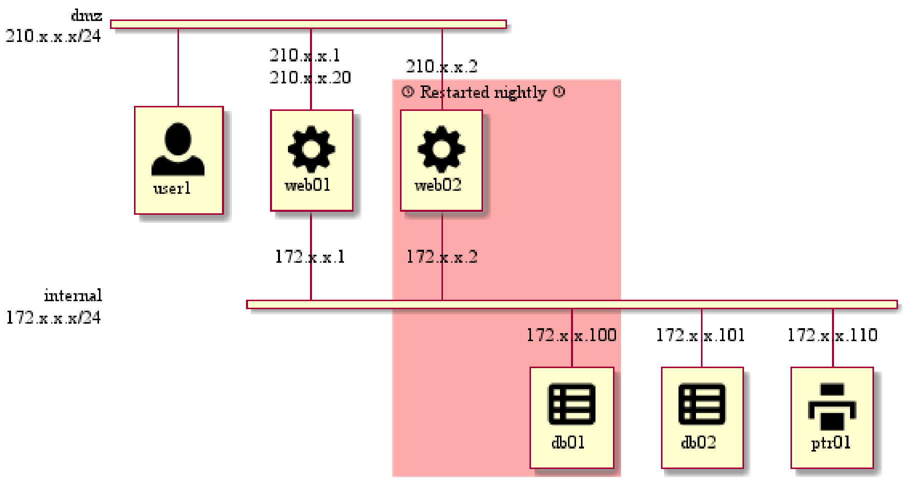
[Ref. QA-11862]

12.6 OpenIconic の使用

ネットワークまたはノードの説明で、OpenIconic のアイコンを使用することもできます。

`<&icon>` の記法でアイコンを表示します。`<&icon*n>` でサイズを n 倍にします。で改行できます：

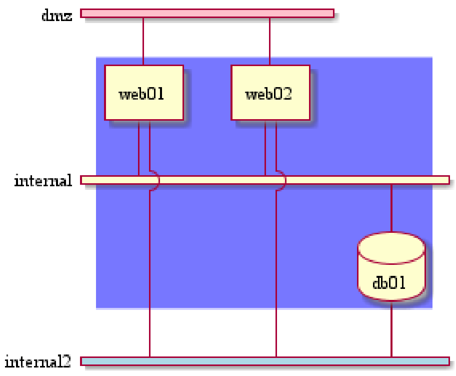
```
@startuml

nwdiag {
  group nightly {
    color = "#FFAAAA";
    description = "<&clock> Restarted nightly <&clock>";
    web02;
    db01;
  }
  network dmz {
      address = "210.x.x.x/24"

      user [description = "<&person*4.5>\n user1"];
      // set multiple addresses (using comma)
      web01 [address = "210.x.x.1, 210.x.x.20",  description = "<&cog*4>\nweb01"]
      web02 [address = "210.x.x.2",  description = "<&cog*4>\nweb02"];

  }
  network internal {
      address = "172.x.x.x/24";

      web01 [address = "172.x.x.1"];
      web02 [address = "172.x.x.2"];
      db01 [address = "172.x.x.100",  description = "<&spreadsheet*4>\n db01"];
      db02 [address = "172.x.x.101",  description = "<&spreadsheet*4>\n db02"];
      ptr  [address = "172.x.x.110",  description = "<&print*4>\n ptr01"];
  }
}
@enduml
```



12.7 複数のネットワークに一つのノードを定義

異なる二つ以上のネットワークに同一のノードを使用することができます。この場合、*nwdiag* ではネットワークの上をジャンプする線が描画されます。

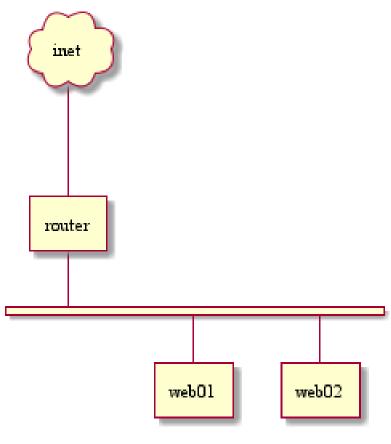
```
@startuml
nwdiag {
  // define group at outside network definitions
  group {
    color = "#7777FF";

    web01;
    web02;
    db01;
  }

  network dmz {
    color = "pink"

    web01;
    web02;
  }

  network internal {
    web01;
    web02;
    db01 [shape = database ];
  }

  network internal2 {
    color = "LightBlue";

    web01;
    web02;
    db01;
  }

}
@enduml
```



12.8 ピア接続

ピア接続は、二つのノード間を単純につなぐものです。この場合は、横長の「バス線」ネットワークは使用されません。

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

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

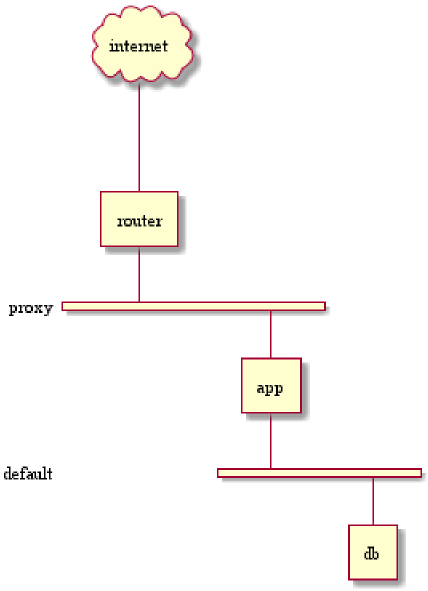


12.9 ピア接続とグループ

12.9.1 グループ無し

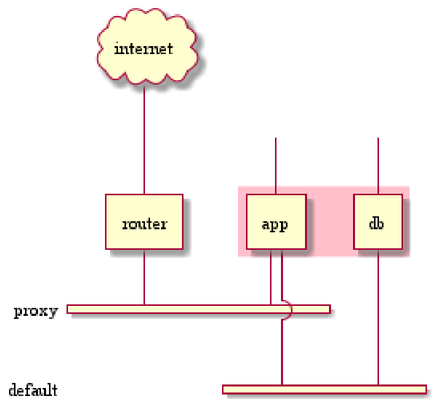
```
@startuml
nwdiag {
    internet [ shape = cloud];
    internet -- router;
```

```
    network proxy {
        router;
        app;
    }
    network default {
     app;
        db;
    }
}
@enduml
```



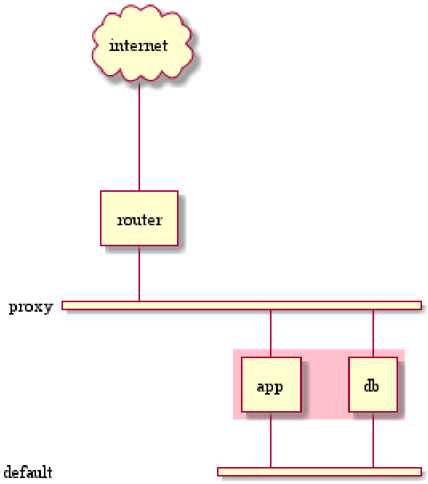
12.9.2 1 番目にグループを記述

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

    group {
      color = "pink";
      app;
      db;
    }

    network proxy {
        router;
        app;
    }

    network default {
     app;
        db;
    }
}
@enduml
```



12.9.3 2 番目にグループを記述

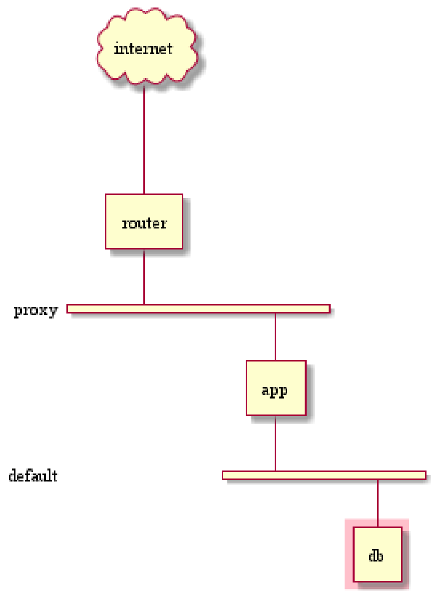
```
@startuml
nwdiag {
    internet [ shape = cloud];
    internet -- router;

    network proxy {
        router;
        app;
    }

    group {
      color = "pink";
      app;
      db;
    }

    network default {
     app;
        db;
    }
}
@enduml
```



TODO: FIXME proxy から'db' へ線が引かれてしまいます。('db' は'default network' にしかつながっていないはずです。) [グループ無しの例を参照]

12.9.4 3 番目にグループを記述

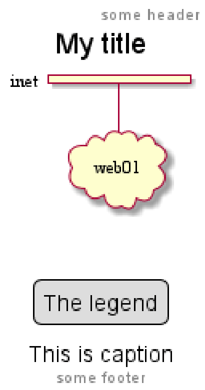
```
@startuml
nwdiag {
    internet [ shape = cloud];
    internet -- router;

    network proxy {
        router;
        app;
    }
    network default {
     app;
        db;
    }
    group {
      color = "pink";
      app;
      db;
    }
}
@enduml
```



TODO: FIXME *[Ref. Issue#408 and QA-12655]* **TODO:** Not totally fixed

12.10 ネットワーク図にタイトル、ヘッダ、フッタ、キャプション、凡例を追加する

```
@startuml

header some header

footer some footer

title My title

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

legend
The legend
end legend

caption This is caption
@enduml
```



[Ref. QA-11303 and Common commands]

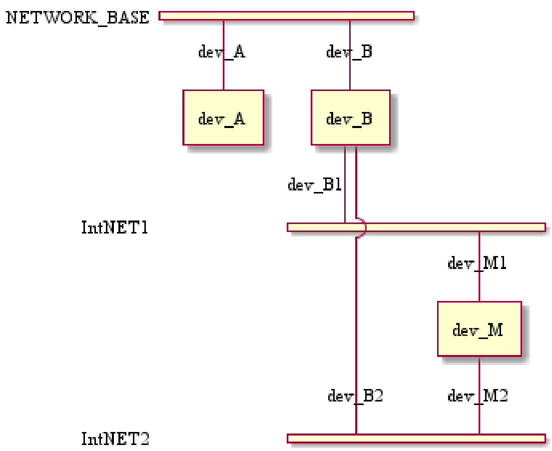
12.11　ネットワークの幅の変更

ネットワークの幅を変更することができます。一部の（もしくはすべての）ネットワークの幅を full（最大幅）に設定して揃えることができます。

可能な組合せの例を示します：

- 無し

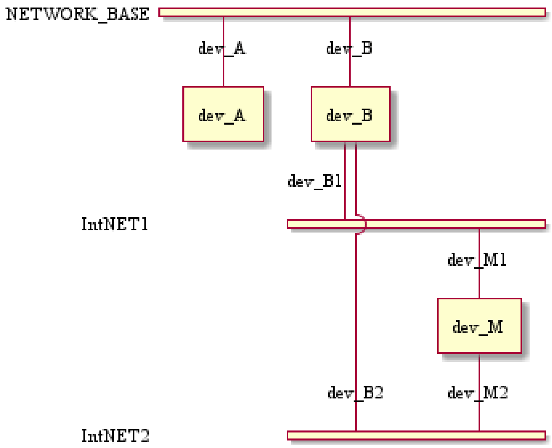
```
@startuml
nwdiag {
  network NETWORK_BASE {
   dev_A    [address = "dev_A" ]
   dev_B [address = "dev_B" ]
  }
  network IntNET1 {
   dev_B [address = "dev_B1" ]
   dev_M [address = "dev_M1" ]
  }
  network IntNET2 {
   dev_B [address = "dev_B2" ]
   dev_M [address = "dev_M2" ]
 }
}
@enduml
```



- １番目のみ

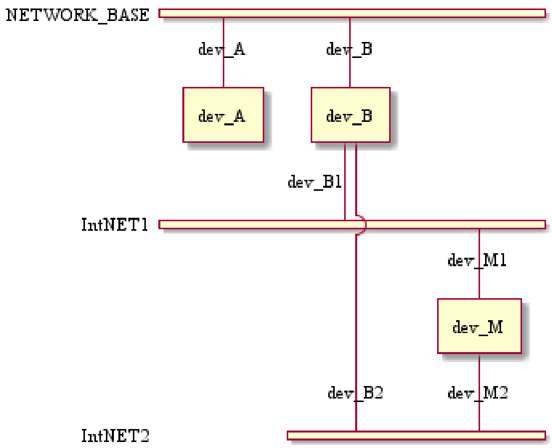
```
@startuml
```

```
nwdiag {
  network NETWORK_BASE {
   width = full
   dev_A   [address = "dev_A" ]
   dev_B [address = "dev_B" ]
  }
  network IntNET1 {
   dev_B [address = "dev_B1" ]
   dev_M [address = "dev_M1" ]
  }
  network IntNET2 {
   dev_B [address = "dev_B2" ]
   dev_M [address = "dev_M2" ]
 }
}
@enduml
```



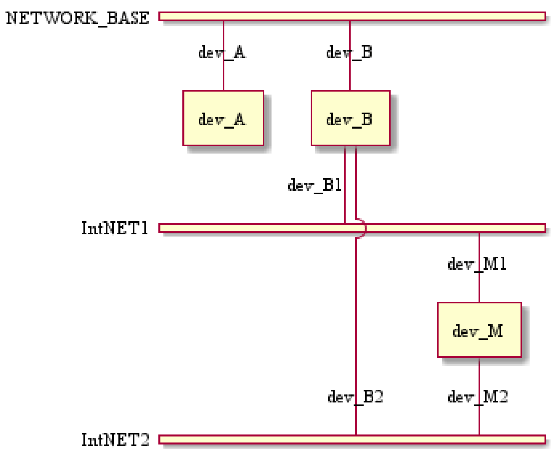
- １番目と２番目

```
@startuml
nwdiag {
  network NETWORK_BASE {
   width = full
   dev_A   [address = "dev_A" ]
   dev_B [address = "dev_B" ]
  }
  network IntNET1 {
   width = full
   dev_B [address = "dev_B1" ]
   dev_M [address = "dev_M1" ]
  }
  network IntNET2 {
   dev_B [address = "dev_B2" ]
   dev_M [address = "dev_M2" ]
 }
}
@enduml
```



- すべてに最大幅を指定

```
@startuml
nwdiag {
  network NETWORK_BASE {
   width = full
   dev_A   [address = "dev_A" ]
   dev_B [address = "dev_B" ]
  }
  network IntNET1 {
   width = full
   dev_B [address = "dev_B1" ]
   dev_M [address = "dev_M1" ]
  }
  network IntNET2 {
   width = full
   dev_B [address = "dev_B2" ]
   dev_M [address = "dev_M2" ]
 }
}
@enduml
```

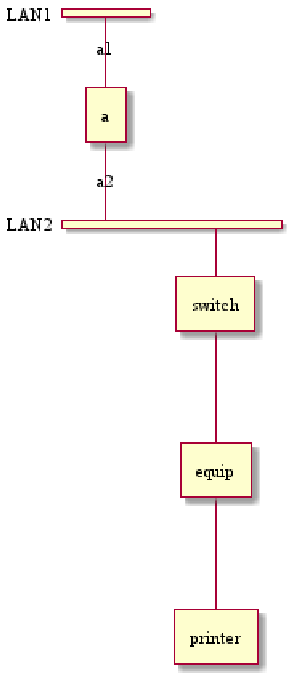


12.12　その他の内部ネットワーク

その他の内部ネットワーク（TCP/IP、USB、SERIAL...）を定義することもできます。

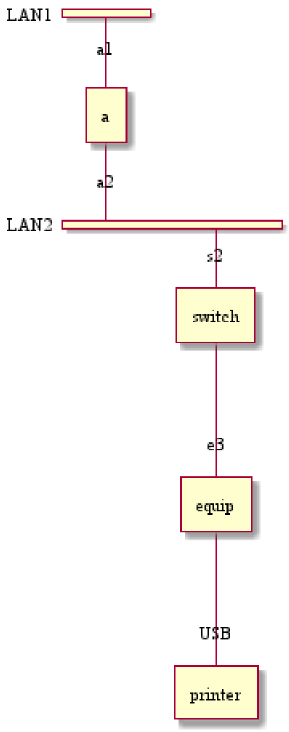
- アドレスまたは種類の指定無し

```
@startuml
nwdiag {
  network LAN1 {
     a [address = "a1"];
  }
  network LAN2 {
     a [address = "a2"];
     switch;
  }
  switch -- equip;
  equip -- printer;
}
@enduml
```



- アドレスまたは種類の指定有り

```
@startuml
nwdiag {
  network LAN1 {
     a [address = "a1"];
  }
  network LAN2 {
     a [address = "a2"];
     switch [address = "s2"];
  }
  switch -- equip;
  equip [address = "e3"];
  equip -- printer;
  printer [address = "USB"];
}
@enduml
```



[Ref. QA-12824]

13 Salt（ワイヤフレームによる GUI 設計ツール）

Salt はグラフィカルインタフェースの設計（ウェブサイトワイヤーフレーム、ページの大まかな設計）を助ける PlantUML のサブプロジェクトです。

このツールの用途は、簡単なサンプルのウィンドウについて議論することです。

キーワード @startsalt、または、@startuml と次の行に続くキーワード salt の、いずれかを使用することができます。

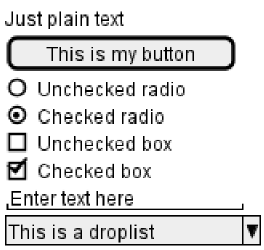
13.1 基本のウィジェット

ウィンドウは中括弧で始めて中括弧で閉じなければなりません。

次のように定義できます。

- ボタンは [と] で括ります。
- ラジオボタンは (と) で括ります。
- チェックボックスは [と] で括ります。
- テキスト領域は " で括ります。
- ドロップリストは ^ で括ります。

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



13.2 罫線の使用

表は括弧 { で開始すれば自動的に作成されます。そして、列を分割するには | を使う必要があります。

例：

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

行や列の罫線を表示したいときは、括弧で開始した直後に、以下のように定義された 1 文字を使用してください。:

文字	結果
#	全ての縦横の罫線を表示する
!	全ての縦線を表示する
-	全ての横線を表示する
+	外枠を表示する

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

13.3 グループボックス [^]

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

[Ref. QA-5840]

13.4 セパレータの使用 [..、==、~~、–]

いくつかの横線をセパレータとして使用することができます。

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

13.5 木構造ウィジェット [T]

木構造を作るには、{T で開始して階層を示すために + を使用する必要があります。

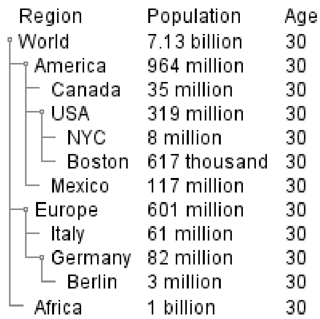
```
@startsalt
{
{T
 + World
 ++ America
 +++ Canada
 +++ USA
 ++++ New York
 ++++ Boston
 +++ Mexico
 ++ Europe
 +++ Italy
 +++ Germany
 ++++ Berlin
 ++ Africa
}
}
@endsalt
```

13.6 木構造と表 [T]

木構造と表を組み合わせることができます。

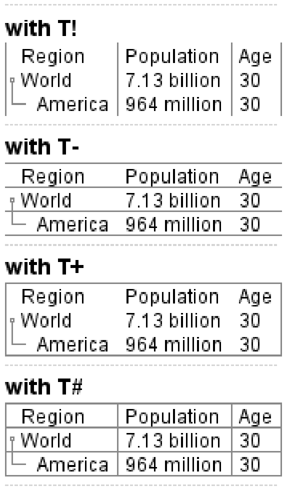
```
@startsalt
{
{T
+Region        | Population    | Age
+ World        | 7.13 billion  | 30
++ America     | 964 million   | 30
+++ Canada     | 35 million    | 30
+++ USA        | 319 million   | 30
++++ NYC       | 8 million     | 30
++++ Boston    | 617 thousand  | 30
+++ Mexico     | 117 million   | 30
++ Europe      | 601 million   | 30
+++ Italy      | 61 million    | 30
+++ Germany    | 82 million    | 30
++++ Berlin    | 3 million     | 30
```

```
++ Africa       | 1 billion     | 30
}
}
@endsalt
```



罫線を追加すると次のようになります。

```
@startsalt
{
..
== with T!
{T!
+Region         | Population    | Age
+ World         | 7.13 billion  | 30
++ America      | 964 million   | 30
}
..
== with T-
{T-
+Region         | Population    | Age
+ World         | 7.13 billion  | 30
++ America      | 964 million   | 30
}
..
== with T+
{T+
+Region         | Population    | Age
+ World         | 7.13 billion  | 30
++ America      | 964 million   | 30
}
..
== with T#
{T#
+Region         | Population    | Age
+ World         | 7.13 billion  | 30
++ America      | 964 million   | 30
}
..
}
@endsalt
```

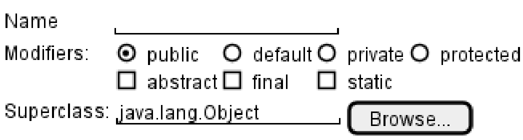


[Ref. QA-1265]

13.7 括弧で括る [{、}]

定義中に、新しい括弧で括ることによりサブ要素を定義することができます。

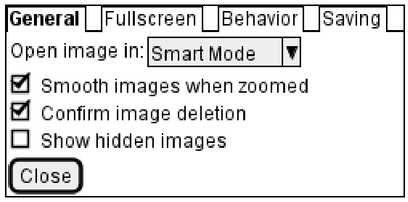
```
@startsalt
{
Name          | "                 "
Modifiers:    | { (X) public | () default | () private | () protected
                [] abstract | [] final   | [] static }
Superclass:   | { "java.lang.Object " | [Browse...] }
}
@endsalt
```



13.8 タブの追加 [/]

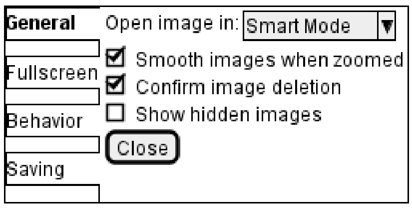
{/ 表記を使用してタブを追加することができます。HTML コードを使用してテキストを太字にできることに注意してください。

```
@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
```



タブは垂直方向にも配向できます：

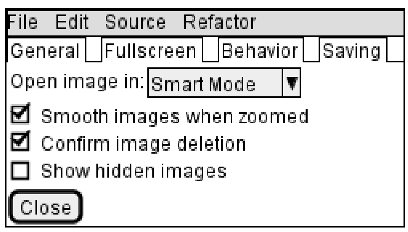
```
@startsalt
{+
{/ <b>General
Fullscreen
Behavior
Saving } |
{
{ Open image in: | ^Smart Mode^ }
[X] Smooth images when zoomed
[X] Confirm image deletion
[ ] Show hidden images
[Close]
}
}
@endsalt
```



13.9 メニューの使用 [*]

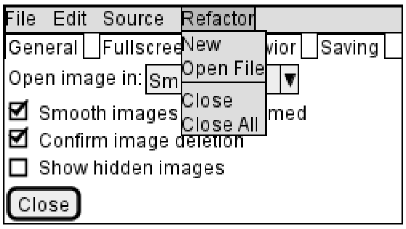
{* 表記でメニューを追加することができます。

```
@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
```



メニューを開くことも可能です：

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



13.10　テーブル（上級）

テーブルのための 2 つの特別な表記を使用することができます。

- `*` は左のセルとの結合となります
- `.` は空のセルとなります

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

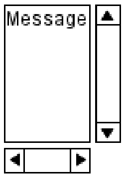
	Column 2	Column 3
Row header 1	value 1	value 2
Row header 2	A long cell	

13.11　スクロールバー [S、SI、S-]

次の例のように、`{S` でスクロールバーを使用できます：

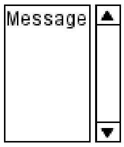
- `{S`：水平方向、垂直方向のスクロールバー

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



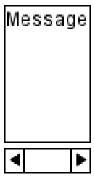
- `{SI`：垂直方向のスクロールバーのみ

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



- `{S-`：水平方向のスクロールバーのみ

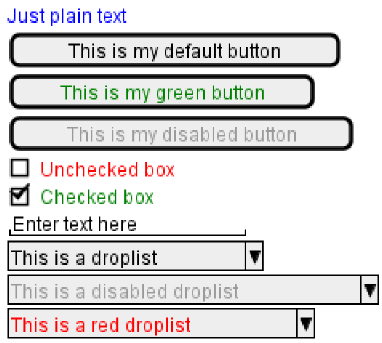
```
@startsalt
{S-
Message
.
.
.
.
}
@endsalt
```



13.12　色

ウィジェットのテキストの色を変えることができます。

```
@startsalt
{
  <color:Blue>Just plain text
  [This is my default button]
  [<color:green>This is my green button]
  [<color:#9a9a9a>This is my disabled button]
  []  <color:red>Unchecked box
  [X] <color:green>Checked box
  "Enter text here   "
  ^This is a droplist^
  ^<color:#9a9a9a>This is a disabled droplist^
  ^<color:red>This is a red droplist^
}
@endsalt
```

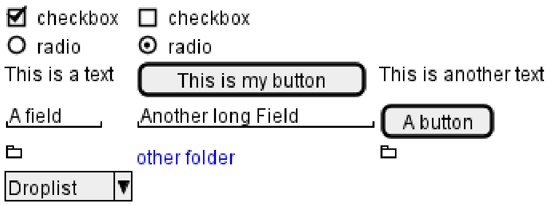


[Ref. QA-12177]

13.13 疑似スプライト [«、»]

<< と >> を使って疑似スプライト (スプライトのような画像) を定義し、それ以降で画像を使用することができます。

```
@startsalt
 {
 [X] checkbox|[] checkbox
 () radio | (X) radio
 This is a text|[This is my button]|This is another text
 "A field"|"Another long Field"|[A button]
 <<folder
 ............
 .XXXXX......
 .X...X......
 .XXXXXXXXXX.
 .X........X.
 .X........X.
 .X........X.
 .X........X.
 .XXXXXXXXXX.
 ............
 >>|<color:blue>other folder|<<folder>>
^Droplist^
}
@endsalt
```



[Ref. QA-5849]

13.14 OpenIconic

OpenIconic はとても素晴らしいオープンソースのアイコンセットです。これらのアイコンは、creole parser に統合されていますので、簡単に使うことができます。このような構文で使用できます。: <&ICON_NAME>

```
@startsalt
{
  Login<&person> | "MyName   "
  Password<&key> | "****     "
```

```
  [Cancel <&circle-x>] | [OK <&account-login>]
}
@endsalt
```

全アイコンのリストは OpenIconic Website にあります。もしくは、次の特別なダイアグラムを使用してください:

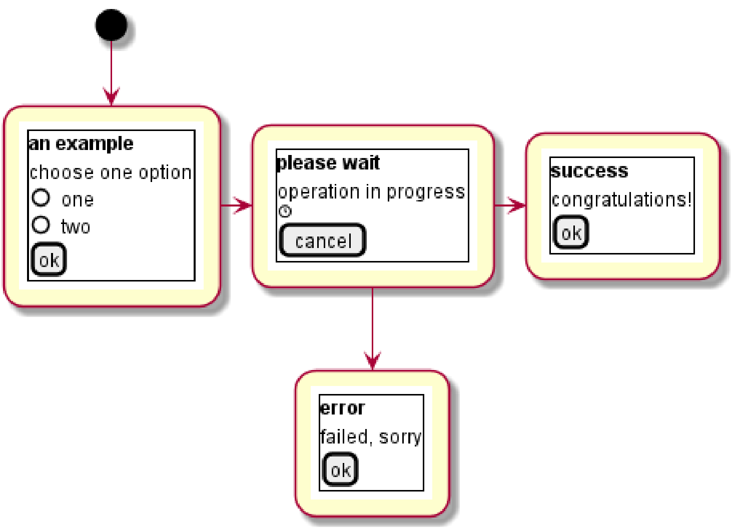
```
@startuml
listopeniconic
@enduml
```

13.15 Salt をアクティビティ図の上に表示する

こちらの説明を参照してください。

```
@startuml
(*) --> "
{{
salt
{+
<b>an example
choose one option
()one
()two
[ok]
}
}}
" as choose

choose -right-> "
{{
```

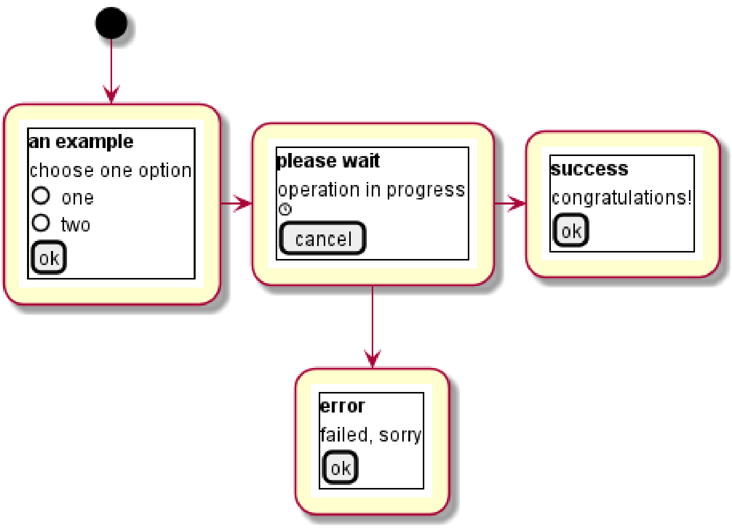
```
salt
{+
<b>please wait
operation in progress
<&clock>
[cancel]
}
}}
" as wait
wait -right-> "
{{
salt
{+
<b>success
congratulations!
[ok]
}
}}
" as success

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



マクロ定義と組み合わせることもできます。

```
@startuml
!unquoted procedure SALT($x)
"{{
salt
%invoke_procedure("_"+$x)
}}" as $x
!endprocedure
```

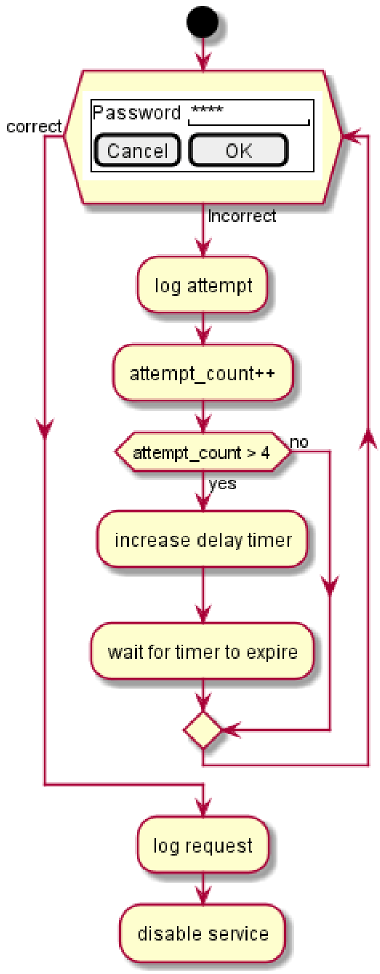
```
!procedure _choose()
{+
<b>an example
choose one option
()one
()two
[ok]
}
!endprocedure

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

!procedure _success()
{+
<b>success
congratulations!
[ok]
}
!endprocedure

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

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



13.16 Salt をアクティビティ図の条件分岐の上に表示する

salt をアクティビティ図の条件分岐の上に表示することができます。

```
@startuml
start
while (\n{{\nsalt\n{+\nPassword | "****     "\n[Cancel] | [  OK   ]}\n}}\n) is (Incorrect)
  :log attempt;
  :attempt_count++;
  if (attempt_count > 4) then (yes)
    :increase delay timer;
    :wait for timer to expire;
  else (no)
  endif
endwhile (correct)
:log request;
:disable service;
@enduml
```



[Ref. QA-8547]

14 アーキテクチャ図

現在、このダイアグラムは提案段階です。将来的に変更されるかもしれません。

新しいシンタックス案の提案を歓迎します！よりよい形を模索するのに、あなたからの意見や提案が役に立ちます!!

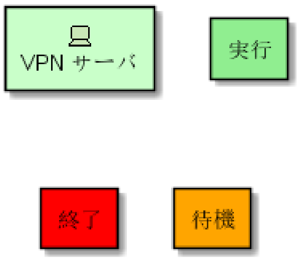
14.1 アーキテクチャの要素

各要素は archimate で定義します。

ステレオタイプとして、アイコンを使うことができます。使用できるアイコンの一覧は、[# 使用できるアイコン一覧 | こちらを参照してください。]

HTML のカラーネームを使って、色の変更ができます。また、いくつかのキーワード (Business, Application, Motivation, Strategy, Technology, Physical, Implementation) を使うこともできます。

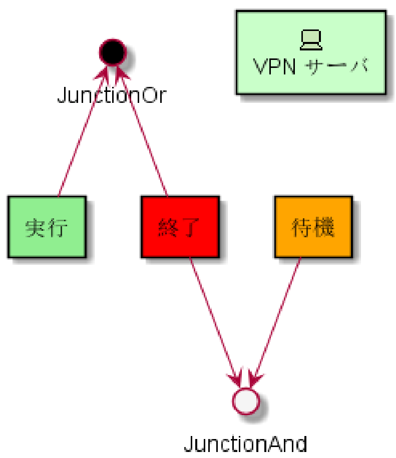
```
@startuml
archimate #Technology "VPN サーバ" as vpnServerA <<technology-device>>

rectangle 実行 #lightgreen
rectangle 終了 #red
rectangle 待機 #orange
@enduml
```



14.2 ジャンクション

プリプロセス機能を使って circle を定義し、使用してください。

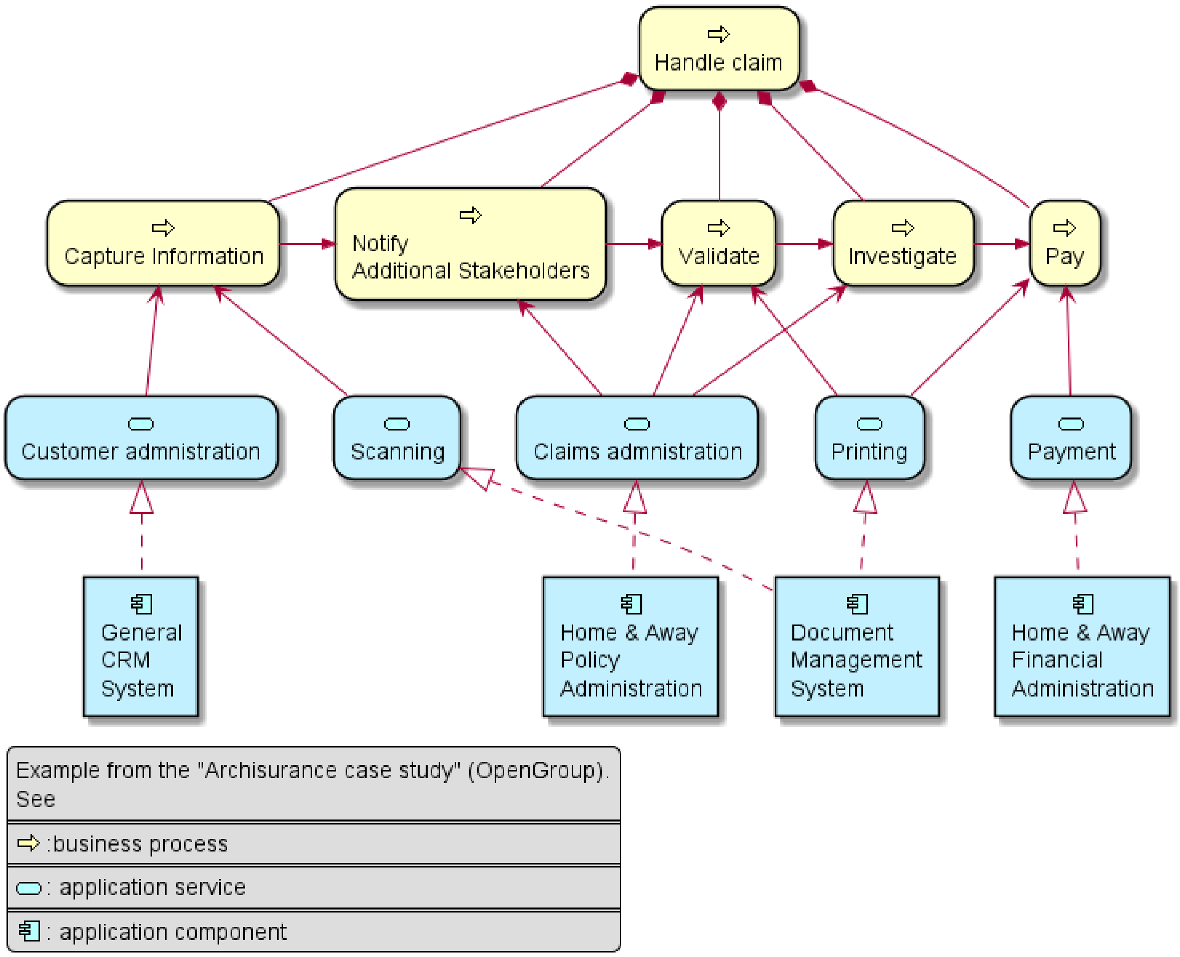
```
@startuml
!define Junction_Or circle #black
!define Junction_And circle #whitesmoke

Junction_And JunctionAnd
Junction_Or JunctionOr

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

rectangle 実行 #lightgreen
rectangle 終了 #red
rectangle 待機 #orange
実行 -up-> JunctionOr
終了 -up-> JunctionOr
終了 -down-> JunctionAnd
待機 -down-> JunctionAnd
@enduml
```



14.3 例 1

```
@startuml
skinparam rectangle<<behavior>> {
roundCorner 25
}
sprite $bProcess jar:archimate/business-process
sprite $aService jar:archimate/application-service
sprite $aComponent jar:archimate/application-component

rectangle "Handle claim"  as HC <<$bProcess>><<behavior>> #Business
rectangle "Capture Information"  as CI <<$bProcess>><<behavior>> #Business
rectangle "Notify\nAdditional Stakeholders" as NAS <<$bProcess>><<behavior>> #Business
rectangle "Validate" as V <<$bProcess>><<behavior>> #Business
rectangle "Investigate" as I <<$bProcess>><<behavior>> #Business
rectangle "Pay" as P <<$bProcess>><<behavior>> #Business

HC *-down- CI
HC *-down- NAS
HC *-down- V
HC *-down- I
HC *-down- P

CI -right->> NAS
NAS -right->> V
V -right->> I
I -right->> P

rectangle "Scanning" as scanning <<$aService>><<behavior>> #Application
rectangle "Customer admnistration" as customerAdministration <<$aService>><<behavior>> #Application
rectangle "Claims admnistration" as claimsAdministration <<$aService>><<behavior>> #Application
rectangle Printing <<$aService>><<behavior>> #Application
rectangle Payment <<$aService>><<behavior>> #Application

scanning -up-> CI
customerAdministration  -up-> CI
claimsAdministration -up-> NAS
claimsAdministration -up-> V
claimsAdministration -up-> I
Payment -up-> P

Printing -up-> V
Printing -up-> P
```

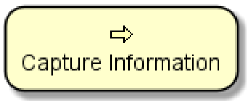
```
rectangle "Document\nManagement\nSystem" as DMS <<$aComponent>> #Application
rectangle "General\nCRM\nSystem" as CRM <<$aComponent>>  #Application
rectangle "Home & Away\nPolicy\nAdministration" as HAPA <<$aComponent>> #Application
rectangle "Home & Away\nFinancial\nAdministration" as HFPA <<$aComponent>>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

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

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



14.4 例 2

```
@startuml
```

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



14.5 使用できるアイコン一覧

アーキテクチャ図で使用できるアイコンの一覧は、次のコードで表示することができます。

```
@startuml
listsprite
@enduml
```

14.6 ArchiMate マクロ

14.6.1 Archimate マクロとライブラリ

Archimate マクロの一覧は Archimate-PlantUML で定義されています。このマクロはアーキテクチャ図の作成を簡単にしてくれます。Archimate は PlantUML の標準ライブラリにネイティブに存在します。

14.6.2 Archimate 要素

マクロを使用した ArchiMate 要素の生成は次のように行います：`Category_ElementName(nameOfTheElement, "description")`

例：

- Motivation カテゴリに含まれる「ステークホルダー」要素を定義する場合、次のように記述します：`Motivation_Stakeholder(StakeholderElement, "Stakeholder Description")`:

```
@startuml
!include <archimate/Archimate>
Motivation_Stakeholder(StakeholderElement, "Stakeholder Description")
@enduml
```

- 「ビジネスサービス」要素を定義する場合は、`Business_Service(BService, "Business Service")`：

```
@startuml
!include <archimate/Archimate>
Business_Service(BService, "Business Service")
@enduml
```

14.6.3 Archimate の関係 (relationship)

Archimate の関係は、次のように定義します：`Rel_RelationType(fromElement, toElement, "description")` また、次のように、２つの要素の方向を定義します：`Rel_RelationType_Direction(fromElement, toElement, "description")`

次の `RelationTypes` がサポートされています：

- Access
- Aggregation
- Assignment
- Association
- Composition
- Flow
- Influence
- Realization
- Serving
- Specialization
- Triggering

次の `Directions` がサポートされています：

- Up
- Down
- Left
- Right

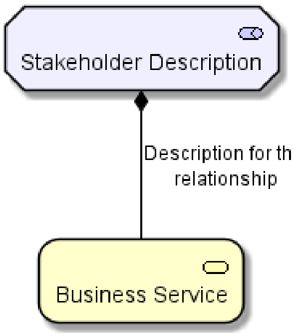
例：

- 上で定義した「ステークホルダー」と「ビジネスサービス」に対して、コンポジション関係を定義する場合、次のように記述します

`Rel_Composition(StakeholderElement, BService, "Description for the relationship")`

```
@startuml
!include <archimate/Archimate>
Motivation_Stakeholder(StakeholderElement, "Stakeholder Description")
```

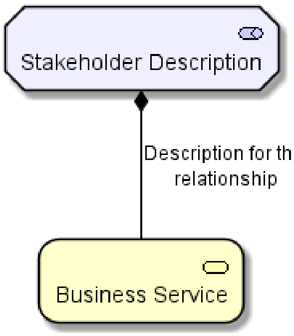
```
Business_Service(BService, "Business Service")
Rel_Composition(StakeholderElement, BService, "Description for the relationship")
@enduml
```



- Unordered List ItemTo orient the two elements in top - down position, the syntax will be

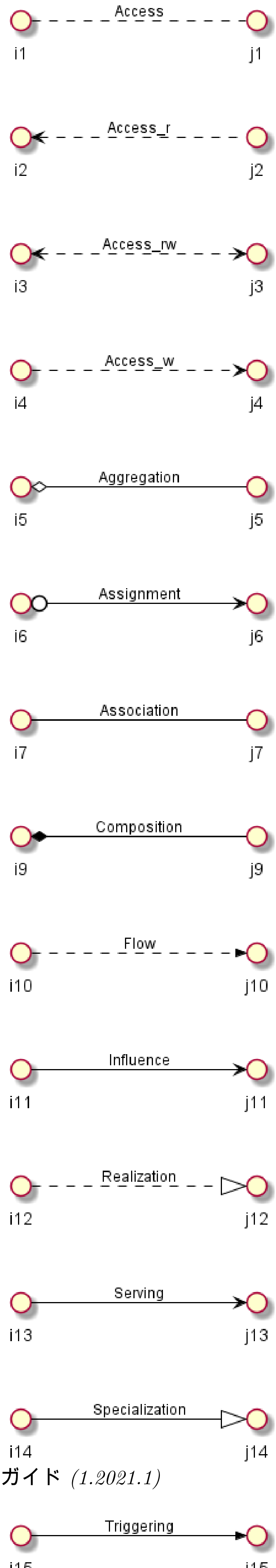
```
Rel_Composition_Down(StakeholderElement, BService, "Description for the relationship")
```

```
@startuml
!include <archimate/Archimate>
Motivation_Stakeholder(StakeholderElement, "Stakeholder Description")
Business_Service(BService, "Business Service")
Rel_Composition_Down(StakeholderElement, BService, "Description for the relationship")
@enduml
```



14.6.4 付録：すべての Archimate RelationType の例

```
@startuml
left to right direction
!include <archimate/Archimate>
Rel_Triggering(i15, j15, Triggering)
Rel_Specialization(i14, j14, Specialization)
Rel_Serving(i13, j13, Serving)
Rel_Realization(i12, j12, Realization)
Rel_Influence(i11, j11, Influence)
Rel_Flow(i10, j10, Flow)
Rel_Composition(i9, j9, Composition)
'Rel_Association_dir(i8, j8, Association_dir)
Rel_Association(i7, j7, Association)
Rel_Assignment(i6, j6, Assignment)
Rel_Aggregation(i5, j5, Aggregation)
Rel_Access_w(i4, j4, Access_w)
Rel_Access_rw(i3, j3, Access_rw)
Rel_Access_r(i2, j2, Access_r)
Rel_Access(i1, j1, Access)
@enduml
```


Access
i1
j1
Access_r
i2
j2
Access_rw
i3
j3
Access_w
i4
j4
Aggregation
i5
j5
Assignment
i6
j6
Association
i7
j7
Composition
i9
j9
Flow
i10
j10
Influence
i11
j11
Realization
i12
j12
Serving
i13
j13
Specialization
i14
j14
Triggering

15 ガントチャート

ガントチャートは、SVC 文型 (主語-動詞-補語) の単純な文によって、自然な英語を書くように記述できます。

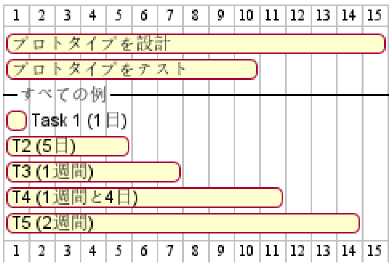
15.1 タスクの定義

タスクは角カッコで定義します。

15.1.1 期間

タスクの期間は、動詞 last で指定します。

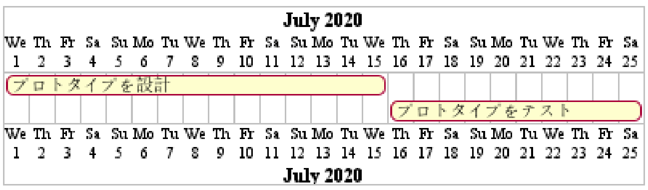
```
@startgantt
[プロトタイプを設計] lasts 15 days
[プロトタイプをテスト] lasts 10 days
-- すべての例 --
[Task 1 (1日)] lasts 1 day
[T2 (5日)] lasts 5 days
[T3 (1週間)] lasts 1 week
[T4 (1週間と4日)] lasts 1 week and 4 days
[T5 (2週間)] lasts 2 weeks
@endgantt
```



15.1.2 開始

タスクの開始は、動詞 start で指定します。

```
@startuml
[プロトタイプを設計] lasts 15 days
[プロトタイプをテスト] lasts 10 days

Project starts 2020-07-01
[プロトタイプを設計] starts 2020-07-01
[プロトタイプをテスト] starts 2020-07-16
@enduml
```

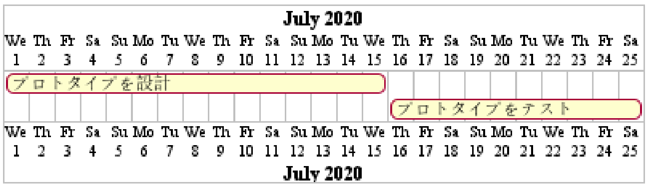


15.1.3 終了

タスクの終了は、動詞 end で指定します。

```
@startuml
[プロトタイプを設計] lasts 15 days
[プロトタイプをテスト] lasts 10 days

Project starts 2020-07-01
```

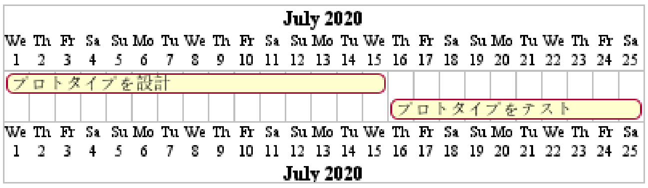
```
[プロトタイプを設計] ends 2020-07-15
[プロトタイプをテスト] ends 2020-07-25

@enduml
```



15.1.4 開始と終了

開始と終了の両方を日付で指定することも可能です。

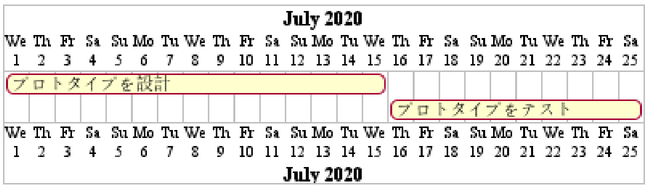
```
@startuml

Project starts 2020-07-01
[プロトタイプを設計] starts 2020-07-01
[プロトタイプをテスト] starts 2020-07-16
[プロトタイプを設計] ends 2020-07-15
[プロトタイプをテスト] ends 2020-07-25

@enduml
```



15.2 AND 接続詞を使った 1 行宣言

and でつなぐことで、宣言を 1 行にできます。

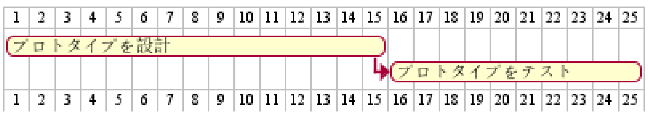
```
@startuml
Project starts 2020-07-01
[プロトタイプを設計] starts 2020-07-01 and ends 2020-07-15
[プロトタイプをテスト] starts 2020-07-16 and lasts 10 days
@enduml
```



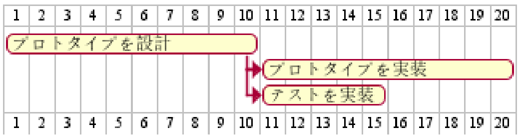
15.3 依存関係

タスク間の依存関係を定義することができます。

```
@startgantt
[プロトタイプを設計] lasts 15 days
[プロトタイプをテスト] lasts 10 days
[プロトタイプをテスト] starts at [プロトタイプを設計]'s end
@endgantt
```



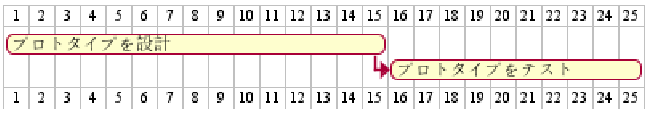
```
@startgantt
[プロトタイプを設計] lasts 10 days
[プロトタイプを実装] lasts 10 days
[テストを実装] lasts 5 days
[プロトタイプを実装] starts at [プロトタイプを設計]'s end
[テストを実装] starts at [プロトタイプを実装]'s start
@endgantt
```



15.4　短い名前

as キーワードを使用して、タスクに短い名前を定義できます。

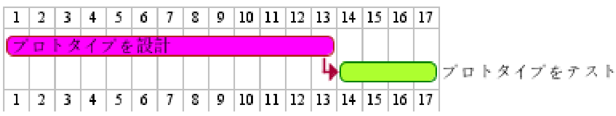
```
@startgantt
[プロトタイプを設計] as [D] lasts 15 days
[プロトタイプをテスト] as [T] lasts 10 days
[T] starts at [D]'s end
@endgantt
```



15.5　色のカスタマイズ

is colored in で、色のカスタマイズができます。

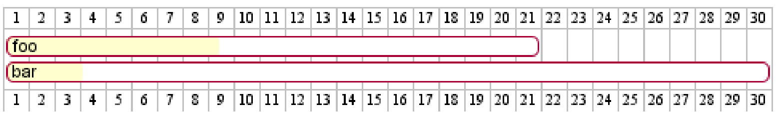
```
@startgantt
[プロトタイプを設計] lasts 13 days
[プロトタイプをテスト] lasts 4 days
[プロトタイプをテスト] starts at [プロトタイプを設計]'s end
[プロトタイプを設計] is colored in Fuchsia/FireBrick
[プロトタイプをテスト] is colored in GreenYellow/Green
@endgantt
```



15.6　進捗状況

タスクの進捗状況を示すことができます。

```
@startgantt
[foo] lasts 21 days
[foo] is 40% completed
[bar] lasts 30 days and is 10% complete
@endgantt
```



15.7　マイルストーン

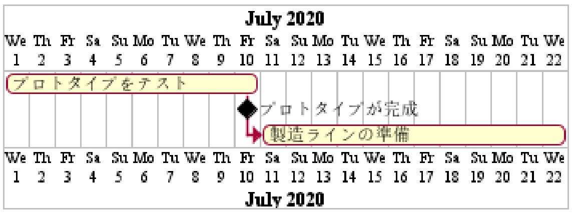
happens でマイルストーンを定義できます。

15.7.1 依存関係に基づくマイルストーン

```
@startgantt
[プロトタイプをテスト] lasts 10 days
[プロトタイプが完成] happens at [プロトタイプをテスト]'s end
[製造ラインの準備] lasts 12 days
[製造ラインの準備] starts at [プロトタイプをテスト]'s end
@endgantt
```

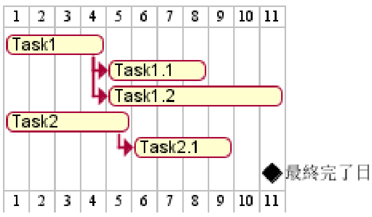
15.7.2 日付指定のマイルストーン

```
@startgantt
Project starts 2020-07-01
[プロトタイプをテスト] lasts 10 days
[プロトタイプが完成] happens 2020-07-10
[製造ラインの準備] lasts 12 days
[製造ラインの準備] starts at [プロトタイプをテスト]'s end
@endgantt
```



15.7.3 全タスク完了のマイルストーン

```
@startgantt
[Task1] lasts 4 days
then [Task1.1] lasts 4 days
[Task1.2] starts at [Task1]'s end and lasts 7 days

[Task2] lasts 5 days
then [Task2.1] lasts 4 days

[最終完了日] happens at [Task1.1]'s end
[最終完了日] happens at [Task1.2]'s end
[最終完了日] happens at [Task2.1]'s end

@endgantt
```

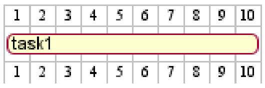


[Ref. QA-10764]

15.8 ハイパーリンク

タスクにハイパーリンクを追加することもできます。

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



15.9 日付の表示

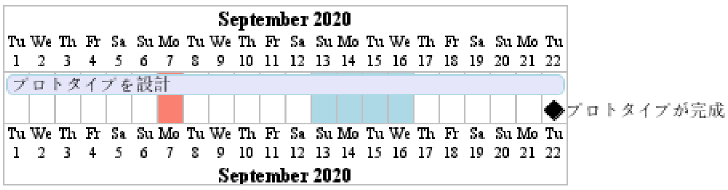
プロジェクトの開始日時に、特定の日付を指定できます。デフォルトでは、一番最初のタスクが指定した日付から開始します。

```
@startgantt
Project starts the 20th of september 2017
[プロトタイプを設計] as [タスク1] lasts 13 days
[タスク1] is colored in Lavender/LightBlue
@endgantt
```

15.10 日付の色

特定の日に色を付けることができます。

```
@startgantt
Project starts the 2020/09/01

2020/09/07 is colored in salmon
2020/09/13 to 2020/09/16 are colored in lightblue

[プロトタイプを設計] as [TASK1] lasts 22 days
[TASK1] is colored in Lavender/LightBlue
[プロトタイプが完成] happens at [TASK1]'s end
@endgantt
```



15.11 スケールの変更

長期にわたるプロジェクトの場合、次のいずれかのパラメータを使用してスケールを変更することができます。

- printscale
- ganttscale
- projectscale

次のいずれかの値を使用します。

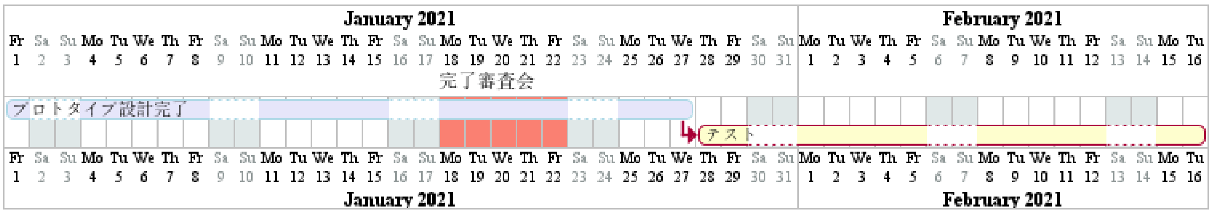
- daily - 日 *(デフォルト)*
- weekly - 週

- monthly - 月

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

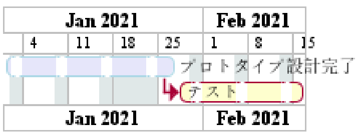
15.11.1 daily（デフォルト）

```
@startuml
saturday are closed
sunday are closed

Project starts the 1st of january 2021
[プロトタイプ設計完了] as [TASK1] lasts 19 days
[TASK1] is colored in Lavender/LightBlue
[テスト] lasts 14 days
[TASK1]->[テスト]

2021-01-18 to 2021-01-22 are named [完了審査会]
2021-01-18 to 2021-01-22 are colored in salmon
@enduml
```

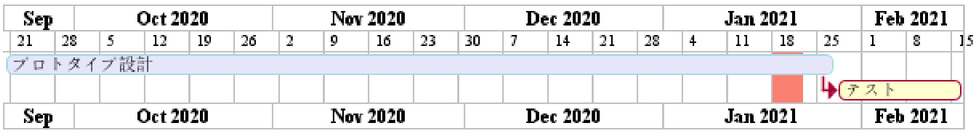


15.11.2 weekly

```
@startuml
printscale weekly
saturday are closed
sunday are closed

Project starts the 1st of january 2021
[プロトタイプ設計完了] as [TASK1] lasts 19 days
[TASK1] is colored in Lavender/LightBlue
[テスト] lasts 14 days
[TASK1]->[テスト]

2021-01-18 to 2021-01-22 are named [完了審査会]
2021-01-18 to 2021-01-22 are colored in salmon
@enduml
```

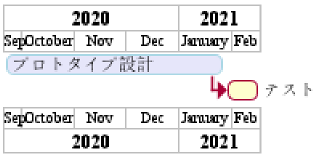



```
@startgantt
printscale weekly
Project starts the 20th of september 2020
[プロトタイプ設計] as [TASK1] lasts 130 days
[TASK1] is colored in Lavender/LightBlue
[テスト] lasts 20 days
[TASK1]->[テスト]

2021-01-18 to 2021-01-22 are named [完了審査会]
2021-01-18 to 2021-01-22 are colored in salmon
@endgantt
```



15.11.3 monthly

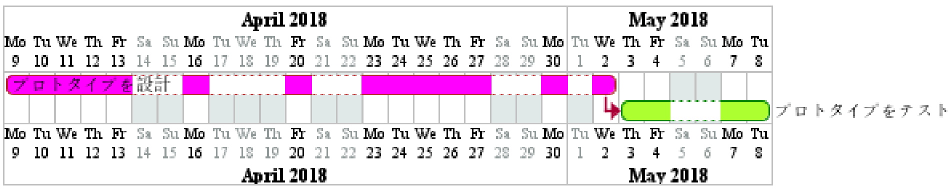
```
@startgantt
projectscale monthly
Project starts the 20th of september 2020
[プロトタイプ設計] as [TASK1] lasts 130 days
[TASK1] is colored in Lavender/LightBlue
[テスト] lasts 20 days
[TASK1]->[テスト]

2021-01-18 to 2021-01-22 are named [完了審査会]
2021-01-18 to 2021-01-22 are colored in salmon
@endgantt
```



15.12 休業日

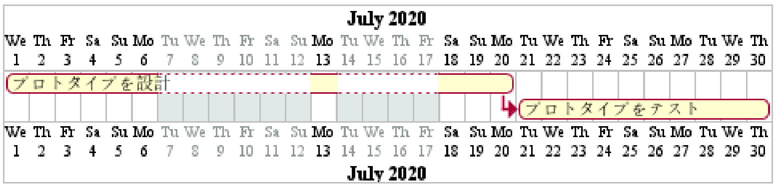
特定の曜日・日付を休業日に指定できます。

```
@startuml
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
[プロトタイプを設計] lasts 14 days
[プロトタイプをテスト] lasts 4 days
[プロトタイプをテスト] starts at [プロトタイプを設計]'s end
[プロトタイプを設計] is colored in Fuchsia/FireBrick
[プロトタイプをテスト] is colored in GreenYellow/Green
@enduml
```



さらに、休業期間中の特定の日だけを開業日にすることができます。

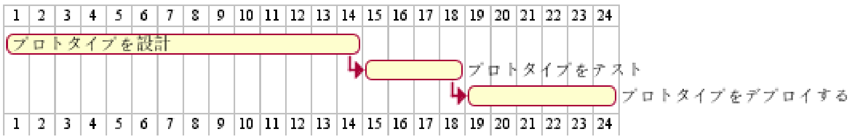
```
@startgantt
2020-07-07 to 2020-07-17 is closed
2020-07-13 is open

Project starts the 2020-07-01
[プロトタイプを設計] lasts 10 days
Then [プロトタイプをテスト] lasts 10 days
@endgantt
```



15.13 簡単なタスク継承

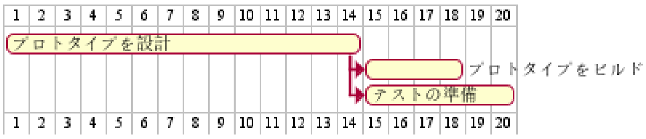
then を使えば、連続したタスクを表すことができます。

```
@startgantt
[プロトタイプを設計] lasts 14 days
then [プロトタイプをテスト] lasts 4 days
then [プロトタイプをデプロイする] lasts 6 days
@enduml
```



矢印-> を使っても表せます。

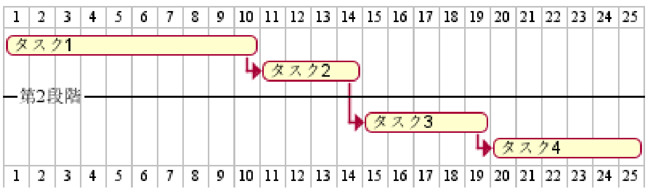
```
@startgantt
[プロトタイプを設計] lasts 14 days
[プロトタイプをビルド] lasts 4 days
[テストの準備] lasts 6 days
[プロトタイプを設計] -> [プロトタイプをビルド]
[プロトタイプを設計] -> [テストの準備]
@endgantt
```



15.14 区切り線

--を使って、タスクを区切ることができます。

```
@startgantt
[タスク1] lasts 10 days
then [タスク2] lasts 4 days
-- 第2段階 --
then [タスク3] lasts 5 days
then [タスク4] lasts 6 days
@endgantt
```

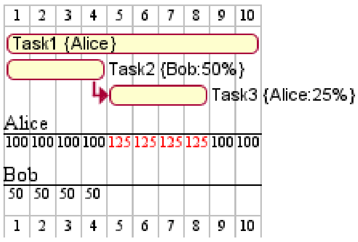


15.15 リソースを使用する

on キーワードとリソース名を波括弧に入れて記述することで、リソースを使用するタスクを表すことができます。

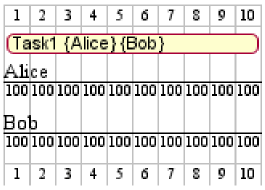
```
@startgantt
```

```
[Task1] on {Alice} lasts 10 days
[Task2] on {Bob:50%} lasts 2 days
then [Task3] on {Alice:25%} lasts 1 days
@endgantt
```



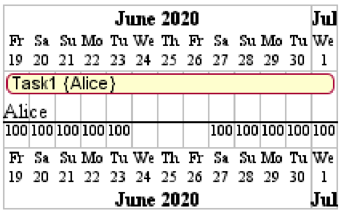
タスクには複数のリソースを割り当てることができます。

```
@startgantt
[Task1] on {Alice} {Bob} lasts 20 days
@endgantt
```



リソースを特定の日だけ割り当てないことができます。

```
@startgantt
project starts on 2020-06-19
[Task1] on {Alice} lasts 10 days
{Alice} is off on 2020-06-24 to 2020-06-26
@endgantt
```

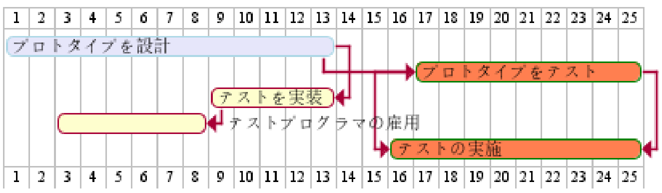


15.16 複雑な例

1 つのタスクに対して、and で同時に複数の設定を行えます。

依存関係に遅延日数を加えることもできます。

```
@startgantt
[プロトタイプを設計] lasts 13 days and is colored in Lavender/LightBlue
[プロトタイプをテスト] lasts 9 days and is colored in Coral/Green and starts 3 days after [プロトタイ
[テストを実装] lasts 5 days and ends at [プロトタイプを設計]'s end
[テストプログラマの雇用] lasts 6 days and ends at [テストを実装]'s start
[テストの実施] is colored in Coral/Green
[テストの実施] starts 1 day before [プロトタイプをテスト]'s start and ends at [プロトタイプをテスト]
@endgantt
```

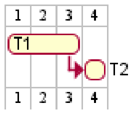


15.17 コメント

共通コマンドのページに書かれている通り、 blockquote Everything that starts with `simple quote '` is a comment.

You can also put comments on several lines using `/'` to start and `'/` to end. blockquote *(*つまり、コメント行は、*(*空白文字を除くと*)* シングルクォーテーション `'` で始まっている必要があります。*)*

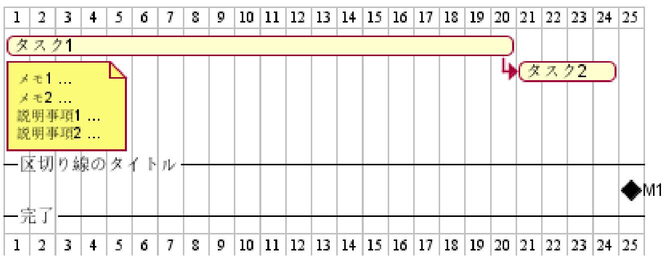
```
@startgantt
' これはコメントです

[T1] lasts 3 days

/' このコメントは
複数行にわたっています '/

[T2] starts at [T1]'s end and lasts 1 day
@endgantt
```



15.18 スタイルの使用

15.18.1 スタイル無し(デフォルト)

```
@startuml
[タスク1] lasts 20 days
note bottom
  メモ1 ...
  メモ2 ...
  説明事項1 ...
  説明事項2 ...
end note
[タスク2] lasts 4 days
[タスク1] -> [タスク2]
-- 区切り線のタイトル --
[M1] happens on 5 days after [タスク1]'s end
-- 完了 --
@enduml
```

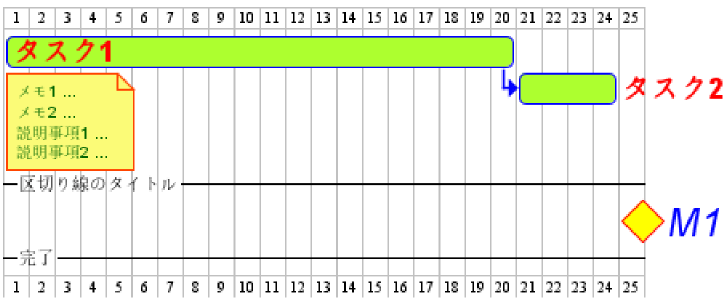


15.18.2 スタイル有り

スタイルを指定して、要素の表示を変更することができます。

```
@startuml
<style>
ganttDiagram {
task {
FontName Helvetica
FontColor red
FontSize 18
```

```
FontStyle bold
BackGroundColor GreenYellow
LineColor blue
}
milestone {
FontColor blue
FontSize 25
FontStyle italic
BackGroundColor yellow
LineColor red
}
note {
FontColor DarkGreen
FontSize 10
LineColor OrangeRed
}
arrow {
FontName Helvetica
FontColor red
FontSize 18
FontStyle bold
BackGroundColor GreenYellow
LineColor blue
}
separator {
LineColor red
BackGroundColor green
FontSize 16
FontStyle bold
FontColor purple
}
}
</style>
[タスク1] lasts 20 days
note bottom
  メモ1 ...
  メモ2 ...
  説明事項1 ...
  説明事項2 ...
end note
[タスク2] lasts 4 days
[タスク1] -> [タスク2]
-- 区切り線のタイトル --
[M1] happens on 5 days after [タスク1]'s end
-- 完了 --
@enduml
```

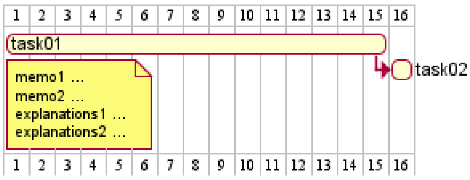


[Ref. QA-10835, QA-12045, QA-11877 and PR-438]

TODO: TODO *Awaiting style for Separator and all style for Arrow (thickness...)*

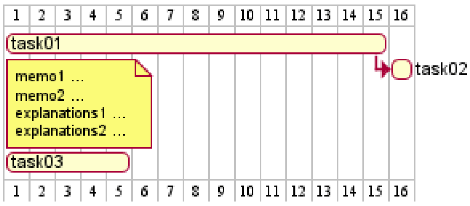
15.19　ノートの追加

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

[task01] -> [task02]

@endgantt
```



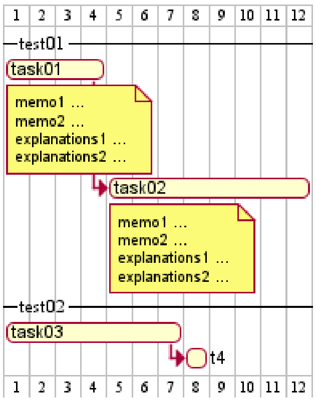
期間の重なりがある場合の例。

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

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

@endgantt
```




```
@startgantt

-- test01 --

[task01] lasts 4 days
note bottom
'note left
memo1 ...
memo2 ...
explanations1 ...
explanations2 ...
end note

[task02] lasts 8 days
[task01] -> [task02]
note bottom
```

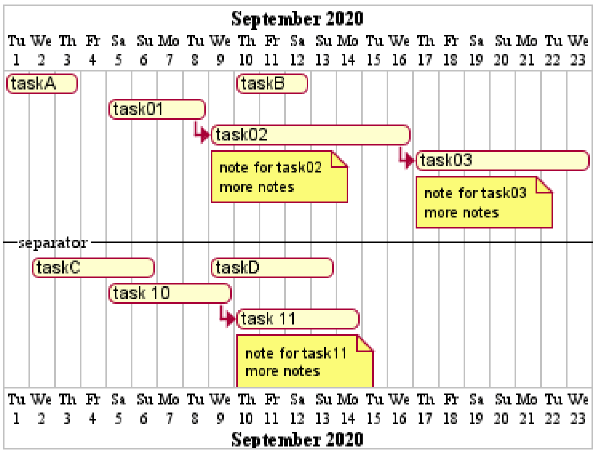
```
'note left
memo1 ...
memo2 ...
explanations1 ...
explanations2 ...
end note
-- test02 --

[task03] as [t3] lasts 7 days
[t3] -> [t4]
@endgantt
```



TODO: DONE *Thanks for correction (of #386 on v1.2020.18) when overlapping*

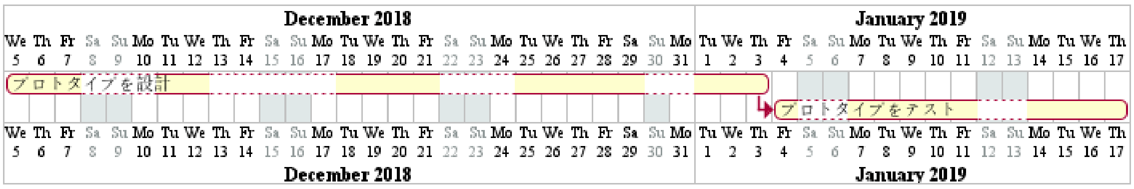
```
@startgantt

Project starts 2020-09-01

[taskA] starts 2020-09-01 and lasts 3 days
[taskB] starts 2020-09-10 and lasts 3 days
[taskB] displays on same row as [taskA]

[task01] starts 2020-09-05 and lasts 4 days

then [task02] lasts 8 days
note bottom
  note for task02
  more notes
end note

then [task03] lasts 7 days
note bottom
  note for task03
  more notes
end note

-- separator --

[taskC] starts 2020-09-02 and lasts 5 days
[taskD] starts 2020-09-09 and lasts 5 days
[taskD] displays on same row as [taskC]

[task 10] starts 2020-09-05 and lasts 5 days
then [task 11] lasts 5 days
note bottom
  note for task11
```

```
  more notes
end note
@endgantt
```



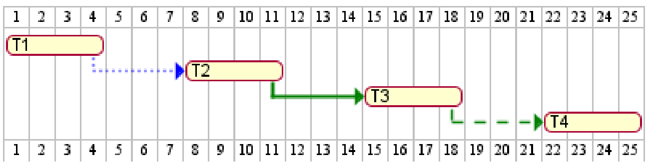
15.20　タスクの中断

```
@startgantt
Project starts the 5th of december 2018
saturday are closed
sunday are closed
2018/12/29 is opened
[プロトタイプを設計] lasts 17 days
[プロトタイプを設計] pauses on 2018/12/13
[プロトタイプを設計] pauses on 2018/12/14
[プロトタイプを設計] pauses on monday
[プロトタイプをテスト] starts at [プロトタイプを設計]'s end and lasts 2 weeks
@endgantt
```



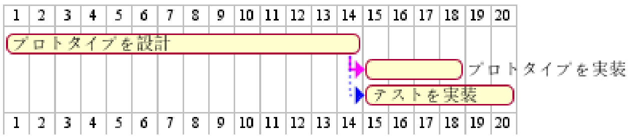
15.21　リンクの色の変更

```
@startgantt
[T1] lasts 4 days
[T2] lasts 4 days and starts 3 days after [T1]'s end with blue dotted link
[T3] lasts 4 days and starts 3 days after [T2]'s end with green bold link
[T4] lasts 4 days and starts 3 days after [T3]'s end with green dashed link
@endgantt
```



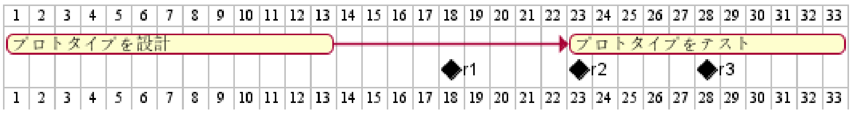

```
@startuml
Links are colored in blue
[プロトタイプを設計] lasts 14 days
[プロトタイプを実装] lasts 4 days
[テストを実装] lasts 6 days
```

```
[プロトタイプを設計] -[#FF00FF]-> [プロトタイプを実装]
[プロトタイプを設計] -[dotted]-> [テストを実装]
@enduml
```



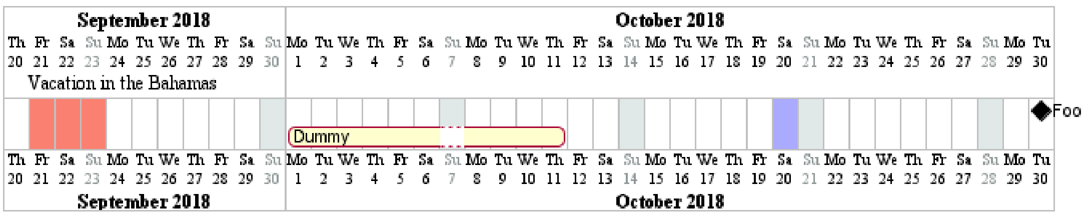
15.22　タスクやマイルストーンを同じ行に表示する

```
@startgantt
[プロトタイプを設計] lasts 13 days
[プロトタイプをテスト] lasts 4 days and 1 week
[プロトタイプをテスト] starts 1 week and 2 days after [プロトタイプを設計]'s end
[プロトタイプをテスト] displays on same row as [プロトタイプを設計]
[r1] happens on 5 days after [プロトタイプを設計]'s end
[r2] happens on 5 days after [r1]'s end
[r3] happens on 5 days after [r2]'s end
[r2] displays on same row as [r1]
[r3] displays on same row as [r1]
@endgantt
```



15.23　今日に色を付ける

```
@startgantt
Project starts the 20th of september 2018
sunday are close
2018/09/21 to 2018/09/23 are colored in salmon
2018/09/21 to 2018/09/30 are named [Vacation in the Bahamas]

today is 30 days after start and is colored in #AAF
[Foo] happens 40 days after start
[Dummy] lasts 10 days and starts 10 days after start

@endgantt
```



15.24　2 つのマイルストーンに挟まれたタスク

```
@startgantt
project starts on 2020-07-01
[開始] happens 2020-07-03
[終了] happens 2020-07-13
[プロトタイプを設計] occurs from [開始] to [終了]
@endgantt
```

15.25 Grammar and verbal form

Verbal form	Example
[T] starts	
[M] happens	

15.26 ガントチャートにタイトル、ヘッダ、フッタ、キャプション、凡例を追加する

```
@startuml

header これはヘッダです

footer これはフッタです

title これはタイトルです

[プロトタイプ設計] lasts 13 days

legend
これは凡例です
end legend

caption これはキャプションです

@enduml
```

(参考：共通コマンド)

15.27 下部のボックスの除去

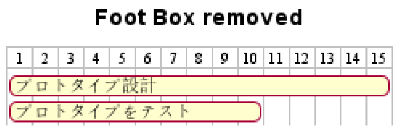
`hide footbox` キーワードを使用して、ガントチャート下部のボックスを取り除くことができます。(シーケンス図と同様)

例：

- 日単位 (プロジェクト開始日の指定無し)

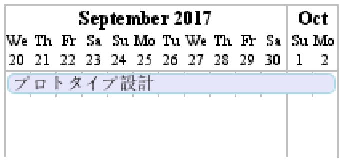
```
@startgantt

hide footbox
title Foot Box removed

[プロトタイプ設計] lasts 15 days
[プロトタイプをテスト] lasts 10 days
```

```
@endgantt
```



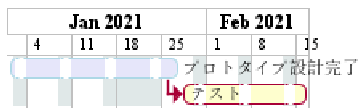
- 日単位

```
@startgantt

Project starts the 20th of september 2017
[プロトタイプ設計] as [TASK1] lasts 13 days
[TASK1] is colored in Lavender/LightBlue

hide footbox
@endgantt
```



- 週単位

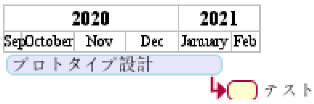
```
@startgantt
hide footbox

printscale weekly
saturday are closed
sunday are closed

Project starts the 1st of january 2021
[プロトタイプ設計完了] as [TASK1] lasts 19 days
[TASK1] is colored in Lavender/LightBlue
[テスト] lasts 14 days
[TASK1]->[テスト]

2021-01-18 to 2021-01-22 are named [完了審査会]
2021-01-18 to 2021-01-22 are colored in salmon
@endgantt
```



- 月単位

```
@startgantt

hide footbox

projectscale monthly
Project starts the 20th of september 2020
[プロトタイプ設計] as [TASK1] lasts 130 days
[TASK1] is colored in Lavender/LightBlue
[テスト] lasts 20 days
[TASK1]->[テスト]

2021-01-18 to 2021-01-22 are named [完了審査会]
2021-01-18 to 2021-01-22 are colored in salmon
```

```
@endgantt
```



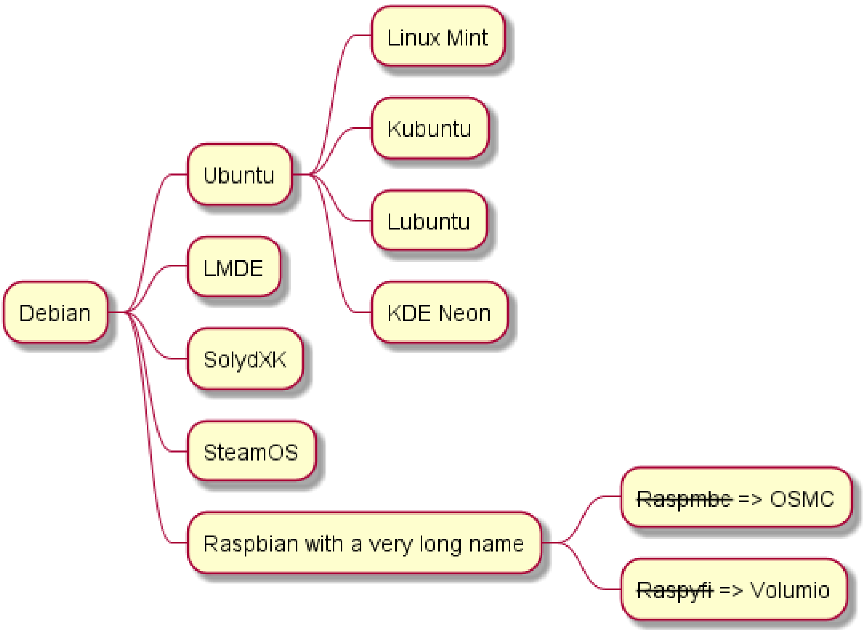
16　マインドマップ

マインドマップはまだベータ版です。文法は予告なく変更するかもしれません。

16.1　OrgMode の文法

OrgMode と互換性のある文法です。

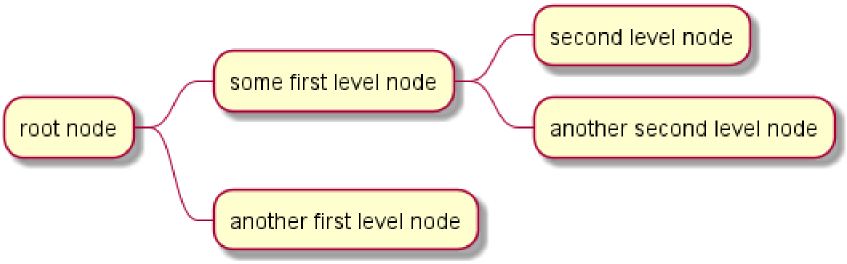
```
@startmindmap
* Debian
** Ubuntu
*** Linux Mint
*** Kubuntu
*** Lubuntu
*** KDE Neon
** LMDE
** SolydXK
** SteamOS
** Raspbian with a very long name
*** <s>Raspmbc</s> => OSMC
*** <s>Raspyfi</s> => Volumio
@endmindmap
```



16.2　マークダウンの文法

マークダウンと互換性がある文法を使えます。

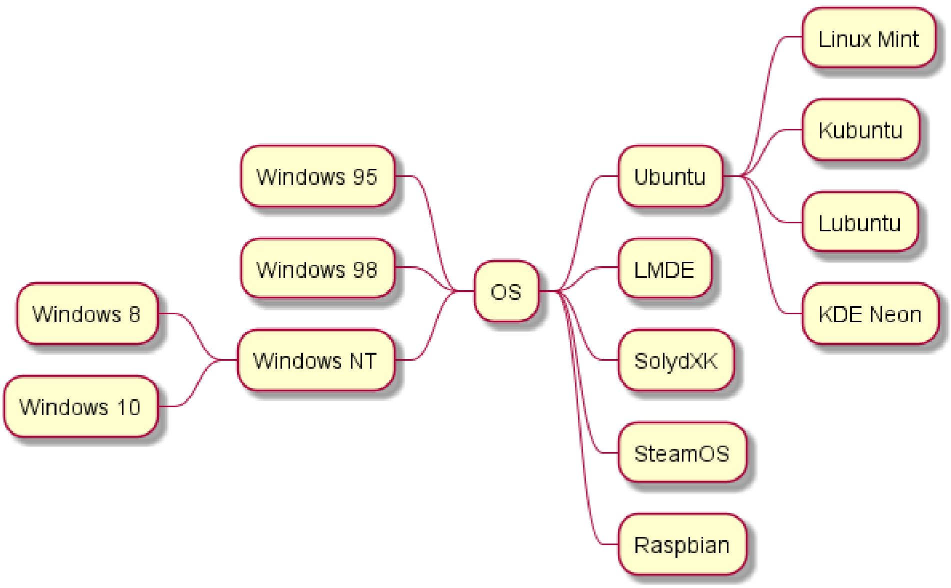
```
@startmindmap
* root node
* some first level node
* second level node
* another second level node
* another first level node
@endmindmap
```



16.3 算術記号による表記

枝を左右どちらの側に伸ばすかを、以下のように算術記号で指定できます。

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

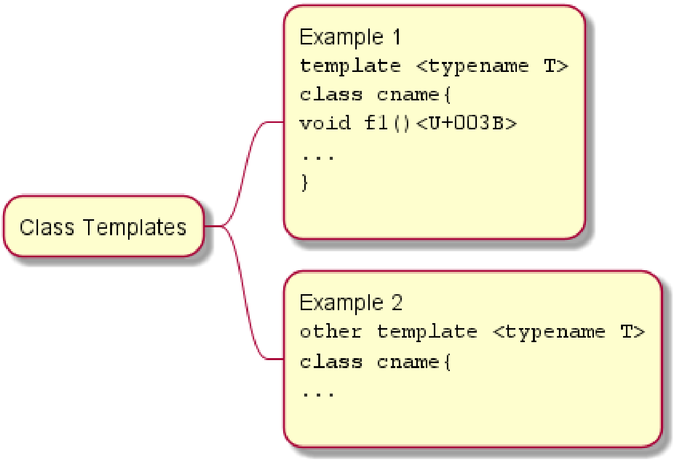


16.4 複数行

: と ; を使って、複数行の箱を作ることができます。

```
@startmindmap
* Class Templates
```

```
**:Example 1
<code>
template <typename T>
class cname{
void f1()<U+003B>
...
}
</code>
;
**:Example 2
<code>
other template <typename T>
class cname{
...
</code>
;
@endmindmap
```



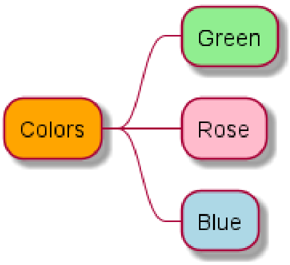
16.5 色

ノードの色を変えることができます。

16.5.1 インラインの色指定

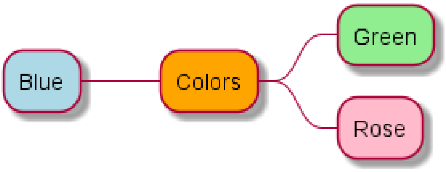
- OrgMode の文法

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



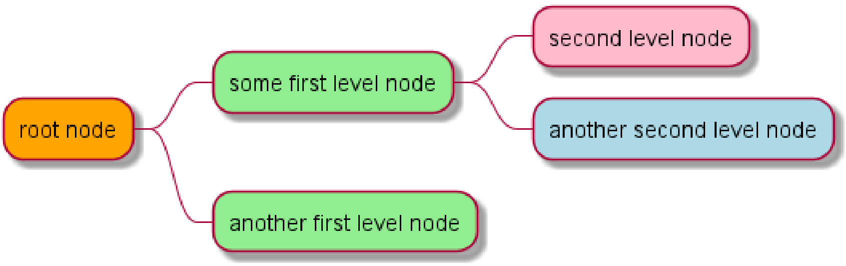
- 算術記号による表記

```
@startmindmap
+[#Orange] Colors
++[#lightgreen] Green
++[#FFBBCC] Rose
--[#lightblue] Blue
@endmindmap
```



- Markdown の文法

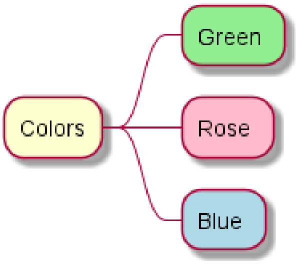
```
@startmindmap
*[#Orange] root node
 *[#lightgreen] some first level node
  *[#FFBBCC] second level node
  *[#lightblue] another second level node
 *[#lightgreen] another first level node
@endmindmap
```



16.5.2 スタイルによる色指定

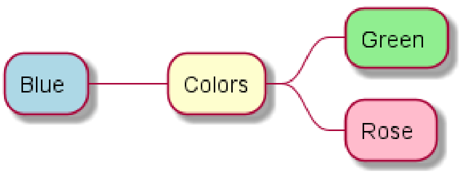
- OrgMode の文法

```
@startmindmap
<style>
mindmapDiagram {
  .green {
    BackgroundColor lightgreen
  }
  .rose {
    BackgroundColor #FFBBCC
  }
  .your_style_name {
    BackgroundColor lightblue
  }
}
</style>
* Colors
** Green <<green>>
** Rose <<rose>>
** Blue <<your_style_name>>
@endmindmap
```



- 算術記号による表記

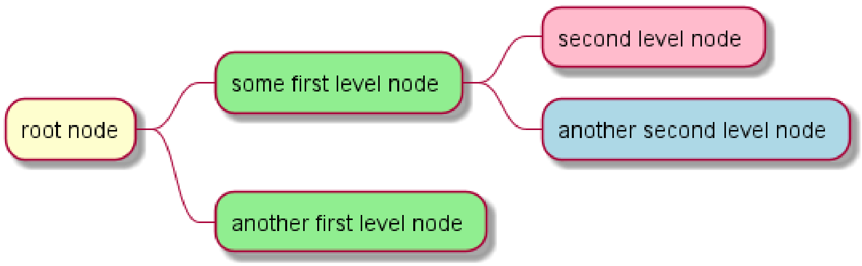
```
@startmindmap
<style>
mindmapDiagram {
  .green {
    BackgroundColor lightgreen
  }
  .rose {
    BackgroundColor #FFBBCC
  }
  .your_style_name {
    BackgroundColor lightblue
  }
}
</style>
+ Colors
++ Green <<green>>
++ Rose <<rose>>
-- Blue <<your_style_name>>
@endmindmap
```



- Markdown の文法

```
@startmindmap
<style>
mindmapDiagram {
  .green {
    BackgroundColor lightgreen
  }
  .rose {
    BackgroundColor #FFBBCC
  }
  .your_style_name {
    BackgroundColor lightblue
  }
}
</style>
* root node
 * some first level node <<green>>
  * second level node <<rose>>
  * another second level node <<your_style_name>>
```

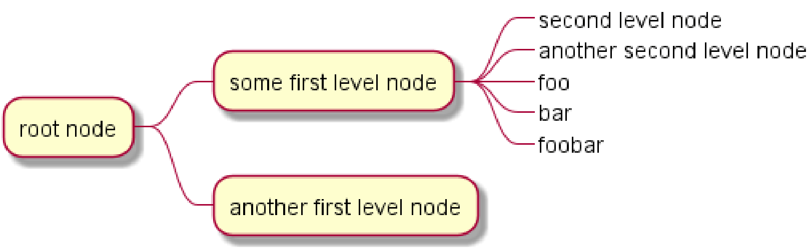
```
 * another first level node <<green>>
@endmindmap
```



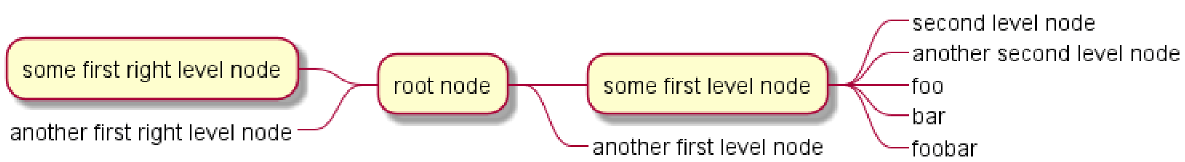
16.6 箱を消す

アンダースコア _ を使うことで、箱を消すことができます。

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



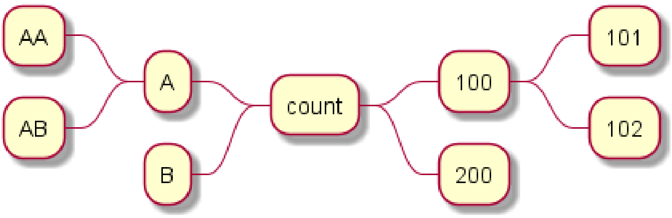

```
@startmindmap
+ root node
++ some first level node
+++_ second level node
+++_ another second level node
+++_ foo
+++_ bar
+++_ foobar
++_ another first level node
-- some first right level node
--_ another first right level node
@endmindmap
```



16.7 図の方向の変更

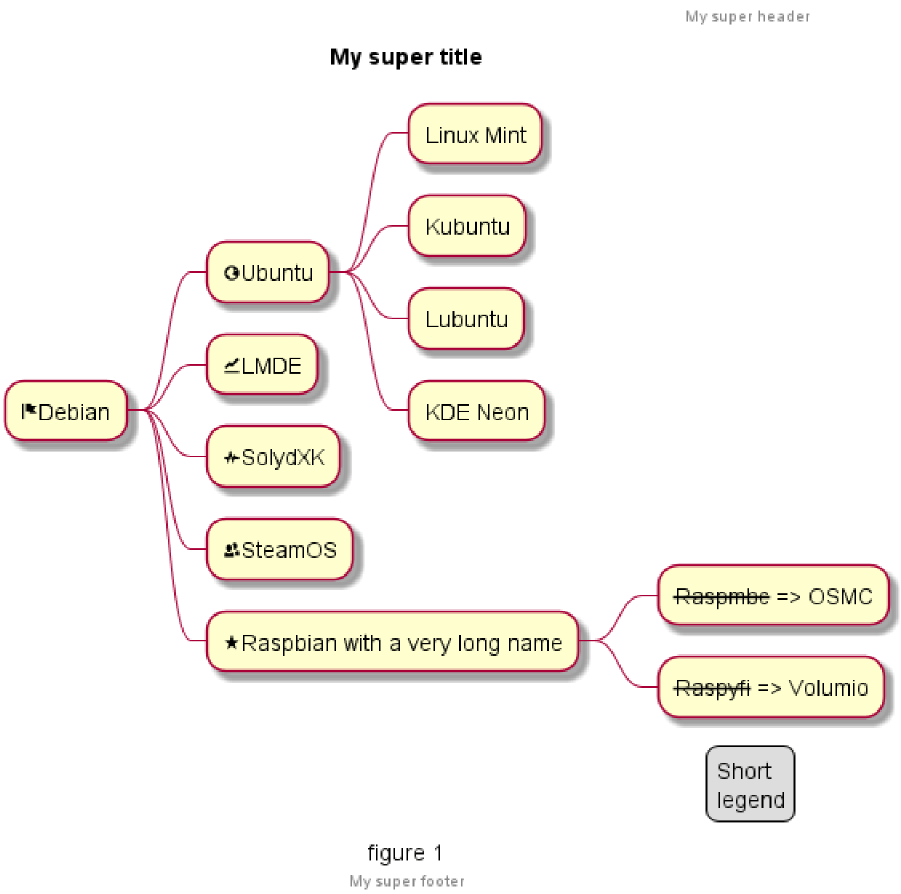
図の両側を使うことができます。

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

left side

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



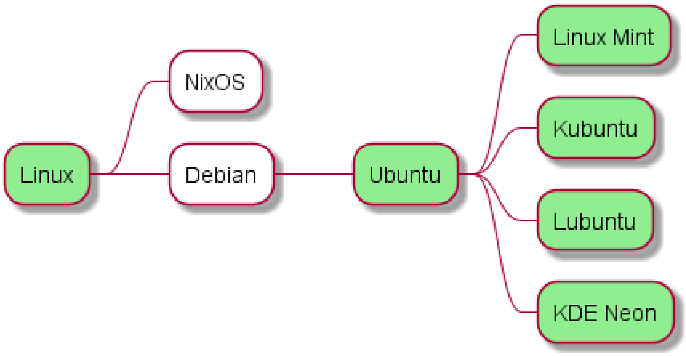
16.8 完全な例

```
@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
```



16.9 スタイル変更

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

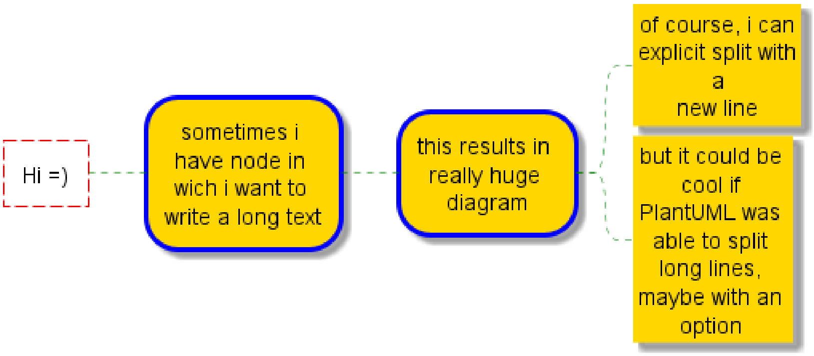


16.10 単語の折り返し

MaximumWidth で、最大幅を設定すると、自動的に単語を折り返すことができます。使用する単位はピクセルです。

```
@startmindmap


<style>
node {
    Padding 12
    Margin 3
    HorizontalAlignment center
    LineColor blue
    LineThickness 3.0
    BackgroundColor gold
    RoundCorner 40
    MaximumWidth 100
}

rootNode {
    LineStyle 8.0;3.0
    LineColor red
    BackgroundColor white
    LineThickness 1.0
    RoundCorner 0
    Shadowing 0.0
}

leafNode {
    LineColor gold
    RoundCorner 0
    Padding 3
}

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

* Hi =)
** sometimes i have node in wich i want to write a long text
*** this results in really huge diagram
```

```
**** of course, i can explicit split with a\nnew line
**** but it could be cool if PlantUML was able to split long lines, maybe with an option

@endmindmap
```



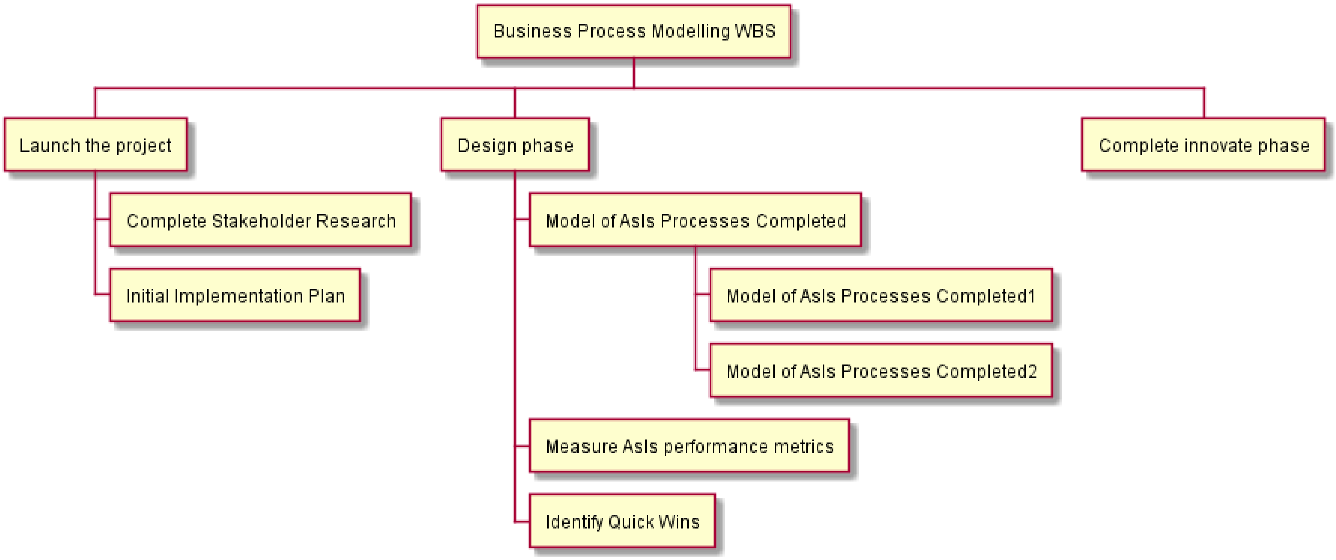
17 Work Breakdown Structure

WBS 図はまだベータ版です。文法は予告なく変更するかもしれません。

17.1 OrgMode の文法

OrgMode と互換性のある文法です。

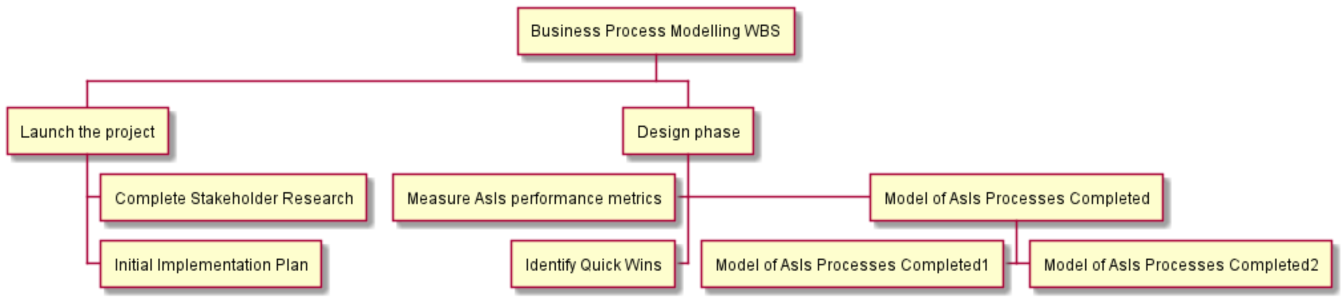
```
@startwbs
* Business Process Modelling WBS
** Launch the project
*** Complete Stakeholder Research
*** Initial Implementation Plan
** Design phase
*** Model of AsIs Processes Completed
**** Model of AsIs Processes Completed1
**** Model of AsIs Processes Completed2
*** Measure AsIs performance metrics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



17.2 方向の変更

< と > を使うことで、方向を変更できます。

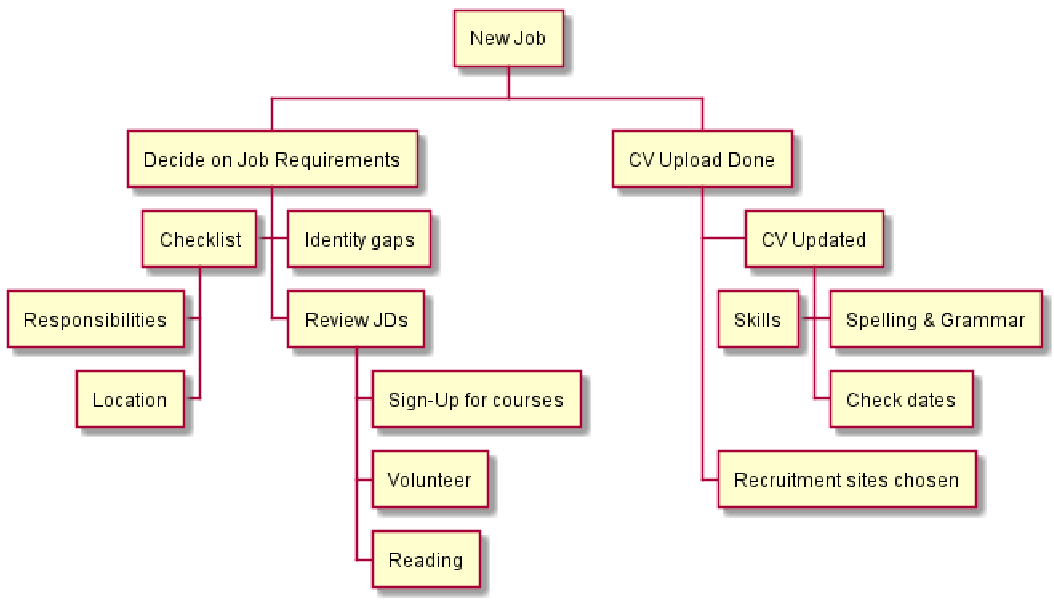
```
@startwbs
* Business Process Modelling WBS
** Launch the project
*** Complete Stakeholder Research
*** Initial Implementation Plan
** Design phase
*** Model of AsIs Processes Completed
****< Model of AsIs Processes Completed1
****> Model of AsIs Processes Completed2
***< Measure AsIs performance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



17.3 算術記号による表記

左右どちらの側に配置するかを、以下のように算術記号で指定できます。

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

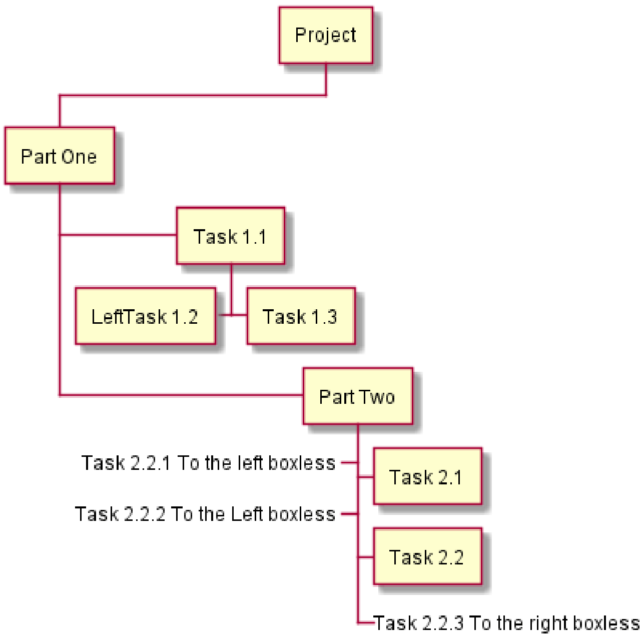


17.4 箱を消す

アンダースコア _ を使って箱を非表示にすることができます。

```
@startwbs
```

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

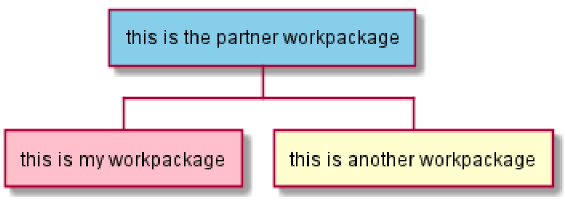


17.5 色(インライン指定とスタイルの色)

ノードの色を変更できます:

- インラインの色指定

```
@startwbs
*[#SkyBlue] this is the partner workpackage
**[#pink] this is my workpackage
** this is another workpackage
@endwbs
```

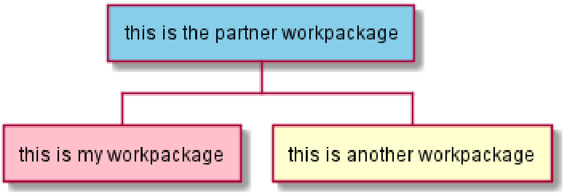


[Ref. QA-12374, only from v1.2020.20]

- スタイルの色指定

```
@startwbs
<style>
wbsDiagram {
```

```
  .pink {
      BackgroundColor pink
  }
  .your_style_name {
      BackgroundColor SkyBlue
  }
}
</style>
* this is the partner workpackage <<your_style_name>>
** this is my workpackage <<pink>>
** this is another workpackage
@endwbs
```

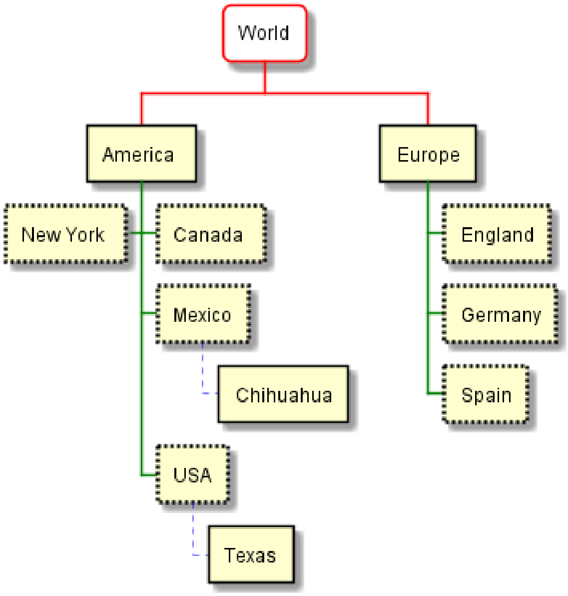


17.6 スタイルを適用する

ダイアグラムのスタイルを変更することができます。

```
@startwbs
<style>
wbsDiagram {
  // all lines (meaning connector and borders, there are no other lines in WBS) are black by default
  Linecolor black
  arrow {
    // note that connector are actually "arrow" even if they don't look like as arrow
    // This is to be consistent with other UML diagrams. Not 100% sure that it's a good idea
    // So now connector are green
    LineColor green
  }
  :depth(0) {
      // will target root node
      BackgroundColor White
      RoundCorner 10
      LineColor red
      // Because we are targetting depth(0) for everything, border and connector for level 0 will be
  }
  arrow {
    :depth(2) {
      // Targetting only connector between Mexico-Chihuahua and USA-Texas
      LineColor blue
      LineStyle 4
      LineThickness .5
    }
  }
  node {
    :depth(2) {
      LineStyle 2
      LineThickness 2.5
    }
  }

}
```

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

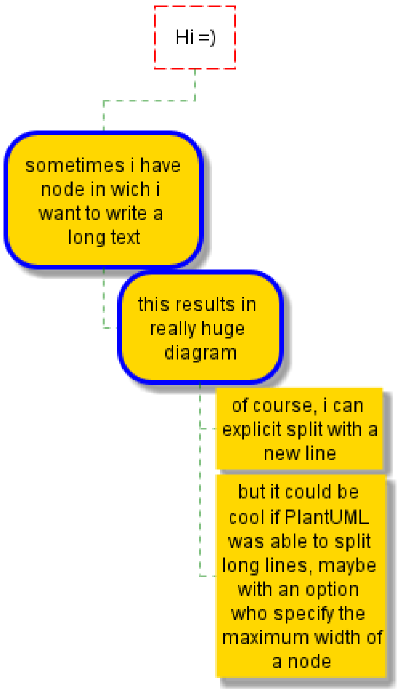


17.7 単語の折り返し

MaximumWidth で、最大幅を設定すると、自動的に単語を折り返すことができます。使用する単位はピクセルです。

```
@startwbs


<style>
node {
    Padding 12
    Margin 3
    HorizontalAlignment center
    LineColor blue
    LineThickness 3.0
    BackgroundColor gold
    RoundCorner 40
    MaximumWidth 100
}

rootNode {
    LineStyle 8.0;3.0
    LineColor red
    BackgroundColor white
```

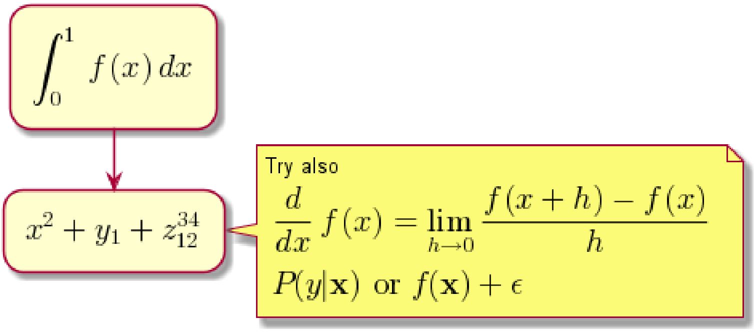
```
    LineThickness 1.0
    RoundCorner 0
    Shadowing 0.0
}

leafNode {
    LineColor gold
    RoundCorner 0
    Padding 3
}

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

* Hi =)
** sometimes i have node in wich i want to write a long text
*** this results in really huge diagram
**** of course, i can explicit split with a\nnew line
**** but it could be cool if PlantUML was able to split long lines, maybe with an option who specify

@endwbs
```



18　数式

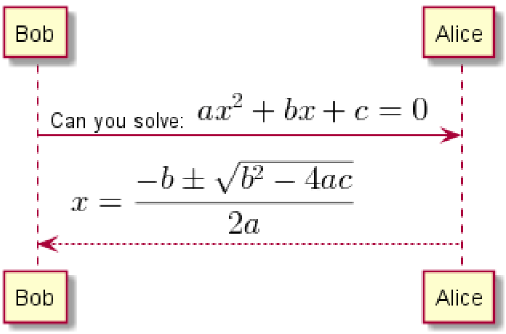
PlantUML では、AsciiMath や JLaTeXMath の構文が使用できます。

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



または

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



18.1　単体で使用する場合

AsciiMath で記述した式を単体で使用したい場合は、`@startmath` と `@endmath` を使用します。

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

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

また、JLaTeXMath の場合は、`@startlatex` と `@endlatex` を使用します。

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

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

18.2　どのように処理しているのか

これらの式を表示するのに、PlantUML では 2 つのオープンソースプロジェクトを使用します。

- AsciiMath：AsciiMath を LaTeX へ変換します。
- JLatexMath LaTeX で書かれた式を表示します。JLaTeXMath は LaTeX のコードを表示するのに最適な Java のライブラリです。

ASCIIMathTeXImg.js は PlantUML の標準ディストリビューションで使用する上で十分に小さいです。(訳者注：そのため、別途インストールする必要はありません。)

PlantUML は、JavaScript エンジンを読み込んで JavaScript コードを実行するために、Java Scripting API (特に、`new ScriptEngineManager().getEngineByName("JavaScript");`) に依存しています。Java 8 には Nashorn という JavaScript エンジンが含まれていますが、これは Java 11 で非推奨になりました。

Java 11 で AsciiMath を使用する場合、次のような警告が表示されます:

```
Warning: Nashorn engine is planned to be removed from a future JDK release
```

Nashorn は Java 15 で削除されました。次の依存関係を設定すれば、GraalVM JavaScript エンジンを代わりに使用することができます：

```
<dependency>
  <groupId>org.graalvm.js</groupId>
  <artifactId>js</artifactId>
  <version>20.2.0</version>
</dependency>
<dependency>
  <groupId>org.graalvm.js</groupId>
  <artifactId>js-scriptengine</artifactId>
  <version>20.2.0</version>
</dependency>
```

また、Java 11 で GraalVM JavaScript エンジンを使用することもでき、その場合、警告メッセージは表示されません。

JLatexMath は比較的大きいです。ダウンロードページからダウンロードし、4 つの jar ファイル (*batik-all-1.7.jar*, *jlatexmath-minimal-1.0.3.jar*, *jlm_cyrillic.jar* and *jlm_greek.jar*) を PlantUML.jar と同じディレクトリに置く必要があります。

19 ER 図

インフォメーションエンジニアリングの表記法をベースにしています。

すでに存在している Class Diagram の拡張になります。

拡張内容:

- インフォメーションエンジニアリング用の関係線の追加
- entity を、クラス図の class と読み替え
- 必須属性を表すものとして、* の表示修飾子を追加

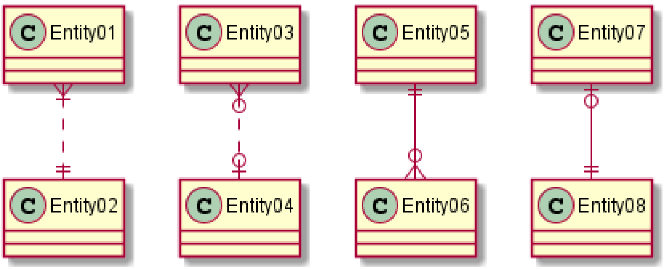
また、クラス図と同じ文法です。クラス図の機能は全て使うことができます。

19.1 インフォメーションエンジニアリングの関係線

Type	記号
0 か 1	|o--
1 のみ	||--
0 以上	}o--
1 以上	}|--

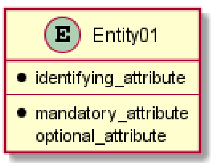
例:

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



19.2 エンティティ

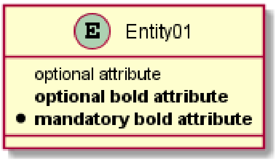
```
@startuml
entity Entity01 {
  * identifying_attribute
  --
  * mandatory_attribute
  optional_attribute
}
@enduml
```



もう一度いいますが、普通のクラス図の文法です。(class の代わりに entity を使うのはさておいて)。クラス図でできることは、ER 図でもできます。

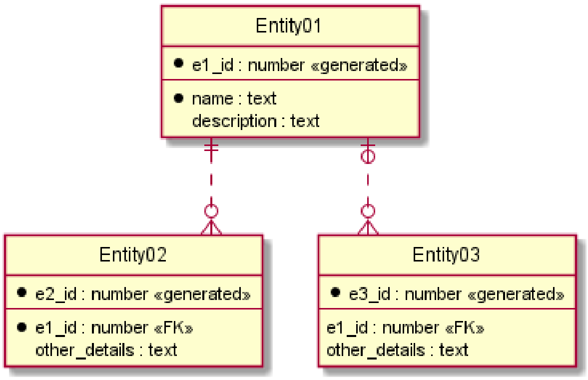
* 表示修飾子は必須属性を表します。空白を 1 文字後ろに入れることで、強調と解釈されることを防ぐと良いでしょう:

```
@startuml
entity Entity01 {
   optional attribute
   **optional bold attribute**
   * **mandatory bold attribute**
}
@enduml
```



19.3 完全な例

```
@startuml

' hide the spot
hide circle

' avoid problems with angled crows feet
skinparam linetype ortho

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

entity "Entity02" as e02 {
  *e2_id : number <<generated>>
  --
  *e1_id : number <<FK>>
  other_details : text
}

entity "Entity03" as e03 {
  *e3_id : number <<generated>>
  --
  e1_id : number <<FK>>
  other_details : text
}

e01 ||..o{ e02
e01 |o..o{ e03

@enduml
```



現状では、エンティティに角度付きで関係線を引こうとすると、見た目がよく有りません。(訳者注:45 度でつながってしまいます)

`linetype ortho` skinparam を使うことで回避できます。

20 共通コマンド

20.1 コメント

シングルクォート' 以降はすべてコメントです。

/' で始まり'/で終わる複数行のコメントを入れることもできます。

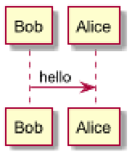
20.2 拡大

scale コマンドを使って、生成する画像を拡大できます。

小数または分数で拡大率を指定できます。ピクセル単位で width（幅）または height（高さ）を指定することもできます。幅と高さの両方を指定することもできます。この場合は、指定したサイズの内側に収まるように調整されます。

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

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

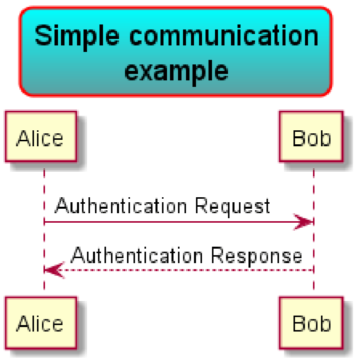


20.3 タイトル

title キーワードを使用してタイトルを入れることができます。タイトルではを使用して改行することができます。

skinparam 設定を使用してタイトルに枠線を付けることができます。

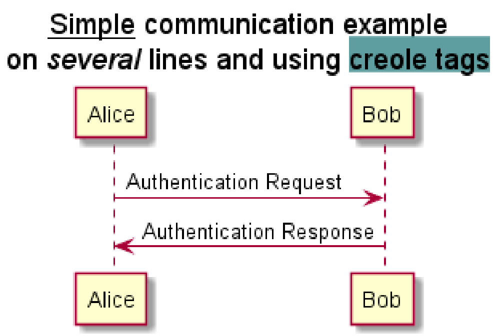
```
@startuml
skinparam titleBorderRoundCorner 15
skinparam titleBorderThickness 2
skinparam titleBorderColor red
skinparam titleBackgroundColor Aqua-CadetBlue

title Simple communication\nexample

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

@enduml
```



タイトル中で creole 書式を使用することもできます。

また、title と end title キーワードを使用することで、複数行にわたってタイトルを記述することもできます。

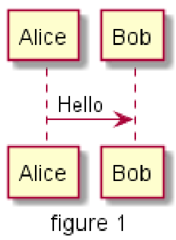
```
@startuml

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
```



20.4 キャプション

caption キーワードを使用して図の下部にキャプションを入れることができます。

```
@startuml

caption figure 1
Alice -> Bob: Hello

@enduml
```



20.5 フッタとヘッダ

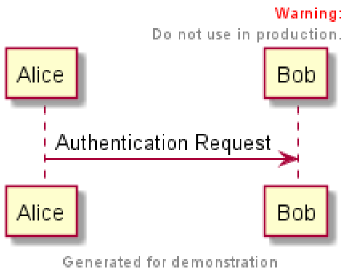
footer、header のコマンドを使って、生成された図にフッタとヘッダを追加することができます。

center、left、right を使ってフッタ、ヘッダの表示位置を指定することもできます。

タイトルと同様に、複数行にわたってフッタまたはヘッダを定義することができます。

また、フッタとヘッダでは HTML タグを使用することもできます。

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

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

center footer Generated for demonstration

@enduml
```

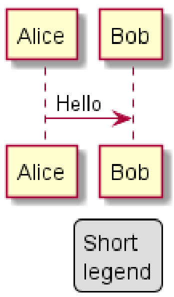


20.6 図の凡例

legend と end legend を使って凡例を追加できます。

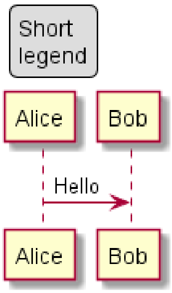
left、right、top、bottom、center を使って、凡例の位置を指定することもできます。

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




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

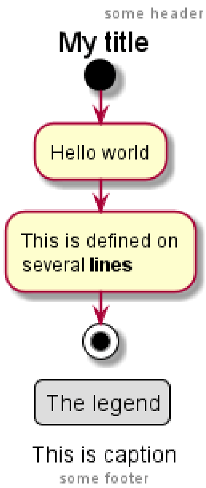
```
@enduml
```



20.7 付録：すべての図の例

20.7.1 アクティビティ図

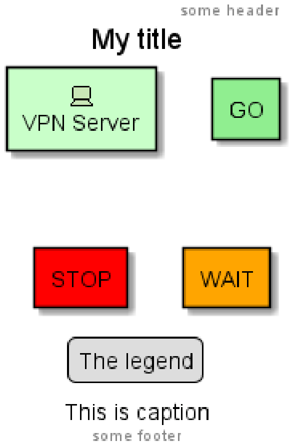
```
@startuml
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

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

@enduml
```



20.7.2 アーキテクチャ図

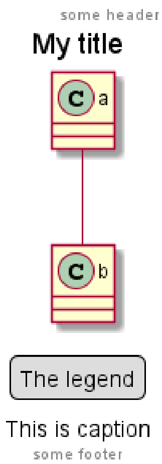
```
@startuml
header some header
```

```
footer some footer

title My title

caption This is caption

legend
The legend
end legend

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

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

@enduml
```



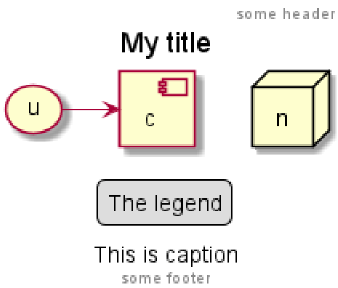
20.7.3 クラス図

```
@startuml
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

a -- b

@enduml
```



20.7.4 コンポーネント図、配置図、ユースケース図

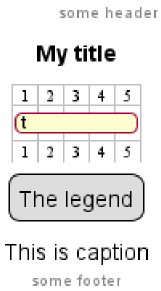
```
@startuml
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

node n
(u) -> [c]

@enduml
```



20.7.5 ガントチャート

```
@startuml
header some header

footer some footer

title My title

caption This is caption

legend
The legend
```

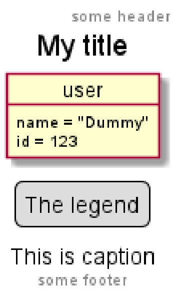
```
end legend

[t] lasts 5 days

@enduml
```



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

20.7.6 オブジェクト図

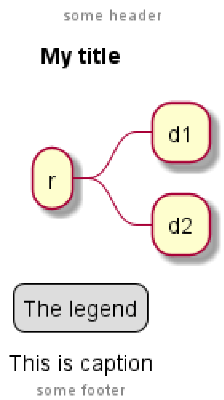
```
@startuml
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

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

@enduml
```



20.7.7 マインドマップ

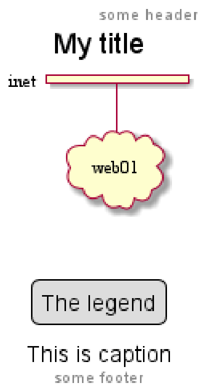
```
@startmindmap
header some header

footer some footer

title My title
```

```
caption This is caption

legend
The legend
end legend

* r
** d1
** d2

@endmindmap
```



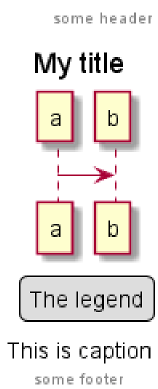
20.7.8 ネットワーク図（nwdiag）

```
@startuml
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

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

@enduml
```



20.7.9 シーケンス図

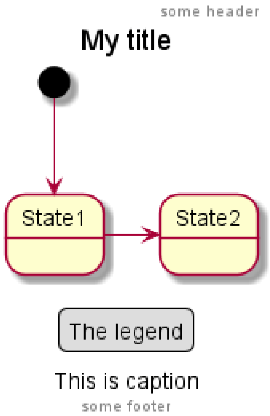
```
@startuml
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

a->b
@enduml
```



20.7.10 ステート図

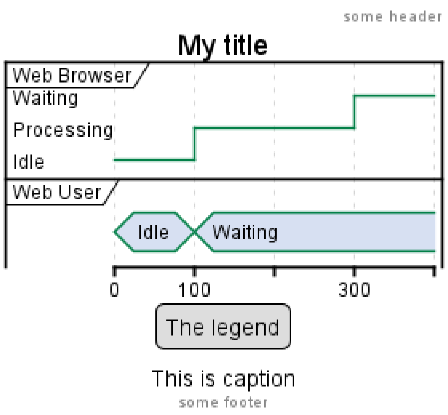
```
@startuml
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend
```

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

@enduml
```



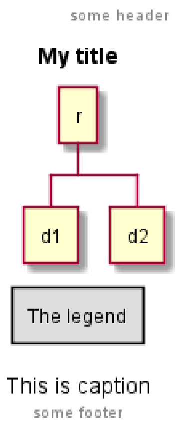
20.7.11 タイミング図

```
@startuml
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

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

@0
WU is Idle
WB is Idle

@100
WU is Waiting
WB is Processing

@300
WB is Waiting

@enduml
```



20.7.12 Work Breakdown Structure (WBS)

```
@startwbs
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

* r
** d1
** d2

@endwbs
```

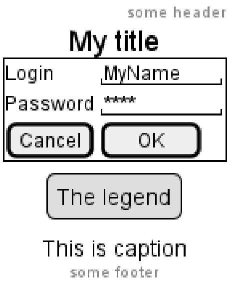


TODO: DONE *[Corrected on V1.2020.17]*

20.7.13 Wireframe (SALT)

```
@startsalt
header some header

footer some footer
```

```
title My title

caption This is caption

legend
The legend
end legend

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



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

20.8 付録：すべての図でスタイルを指定した例

TODO: DONE

FYI:

- すべてが正常に使えるのはシーケンス図のみです。
- `title`、`caption`、`legend` は、**salt diagram** 以外のすべての図で正常に使えます。

TODO: FIXME

- 現状 *(test on 1.2020.18-19)*、`header`、`footer` はシーケンス図を除くすべての図で正常に動作しません。

To be fix; Thanks

TODO: FIXME

ここでは、すべての図を使って `title`、`header`、`footer`、`caption`、`legend` を、デバッグ形式でテストしています：

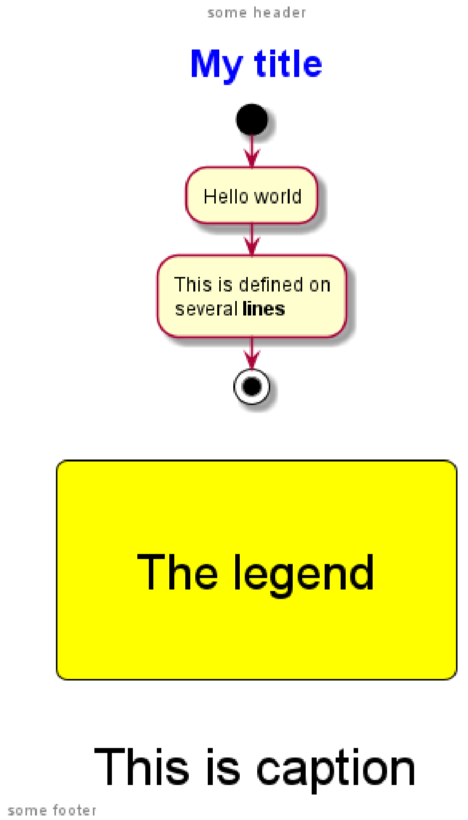
```
<style>
title {
  HorizontalAlignment right
  FontSize 24
  FontColor blue
}

header {
  HorizontalAlignment center
  FontSize 26
  FontColor purple
}

footer {
  HorizontalAlignment left
```

```
  FontSize 28
  FontColor red
}

legend {
  FontSize 30
  BackGroundColor yellow
  Margin 30
  Padding 50
}

caption {
  FontSize 32
}
</style>
```

20.8.1 アクティビティ図

```
@startuml
<style>
title {
  HorizontalAlignment right
  FontSize 24
  FontColor blue
}

header {
  HorizontalAlignment center
  FontSize 26
  FontColor purple
}

footer {
  HorizontalAlignment left
  FontSize 28
  FontColor red
}

legend {
  FontSize 30
  BackGroundColor yellow
  Margin 30
  Padding 50
}

caption {
  FontSize 32
}
</style>
header some header

footer some footer

title My title

caption This is caption

legend
The legend
```

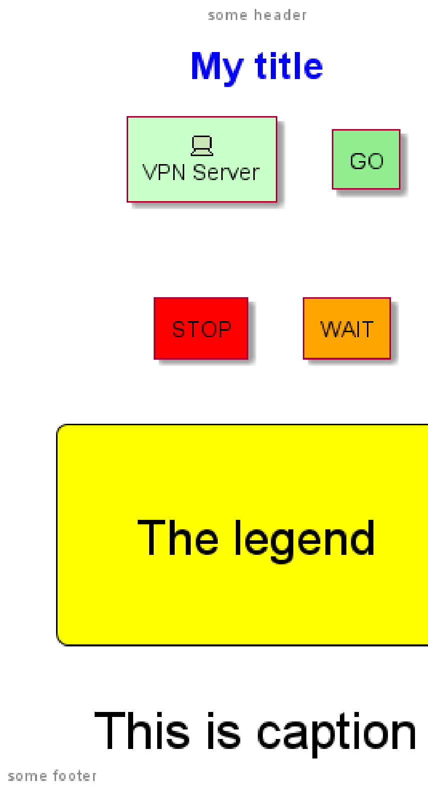
```
end legend

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

@enduml
```



20.8.2 アーキテクチャ図

```
@startuml
<style>
title {
  HorizontalAlignment right
  FontSize 24
  FontColor blue
}

header {
  HorizontalAlignment center
  FontSize 26
  FontColor purple
}

footer {
  HorizontalAlignment left
  FontSize 28
  FontColor red
}
```

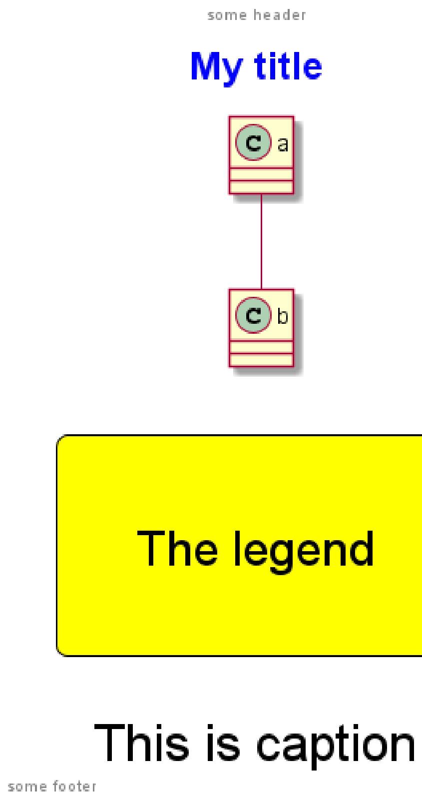
```
legend {
  FontSize 30
  BackGroundColor yellow
  Margin 30
  Padding 50
}

caption {
  FontSize 32
}
</style>
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

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

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

@enduml
```



20.8.3　クラス図

```
@startuml
```

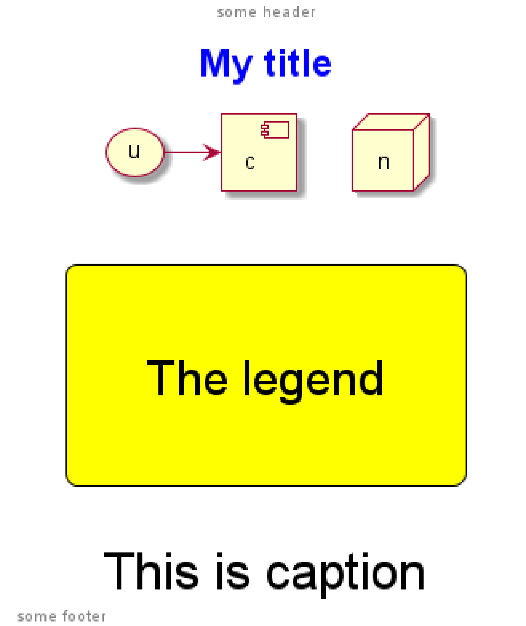
```
<style>
title {
  HorizontalAlignment right
  FontSize 24
  FontColor blue
}

header {
  HorizontalAlignment center
  FontSize 26
  FontColor purple
}

footer {
  HorizontalAlignment left
  FontSize 28
  FontColor red
}

legend {
  FontSize 30
  BackGroundColor yellow
  Margin 30
  Padding 50
}

caption {
  FontSize 32
}
</style>
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

a -- b

@enduml
```



20.8.4 コンポーネント図、配置図、ユースケース図

```
@startuml
<style>
title {
  HorizontalAlignment right
  FontSize 24
  FontColor blue
}

header {
  HorizontalAlignment center
  FontSize 26
  FontColor purple
}

footer {
  HorizontalAlignment left
  FontSize 28
  FontColor red
}

legend {
  FontSize 30
  BackGroundColor yellow
  Margin 30
  Padding 50
}

caption {
  FontSize 32
}
</style>
```

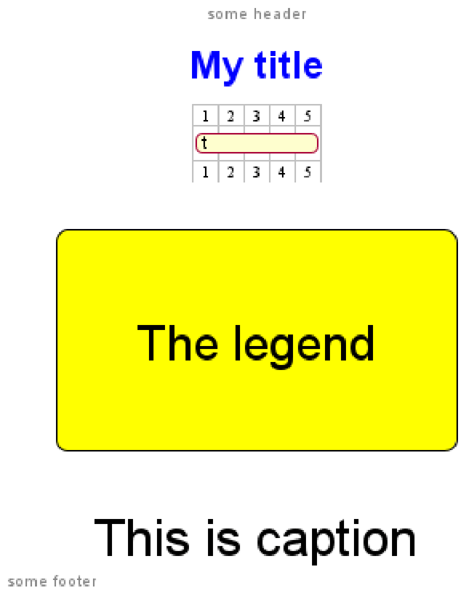
```
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

node n
(u) -> [c]

@enduml
```



20.8.5 ガントチャート

```
@startuml
<style>
title {
  HorizontalAlignment right
  FontSize 24
  FontColor blue
}

header {
  HorizontalAlignment center
  FontSize 26
  FontColor purple
}

footer {
  HorizontalAlignment left
  FontSize 28
  FontColor red
}
```

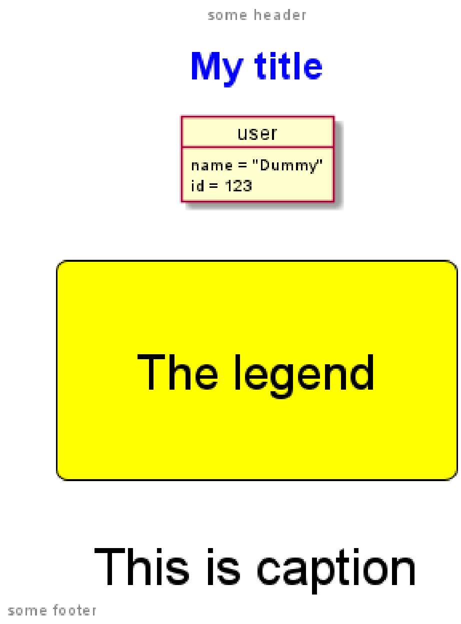
```
legend {
  FontSize 30
  BackGroundColor yellow
  Margin 30
  Padding 50
}

caption {
  FontSize 32
}
</style>
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend


[t] lasts 5 days

@enduml
```



20.8.6 オブジェクト図

```
@startuml
<style>
title {
  HorizontalAlignment right
  FontSize 24
  FontColor blue
}

header {
```

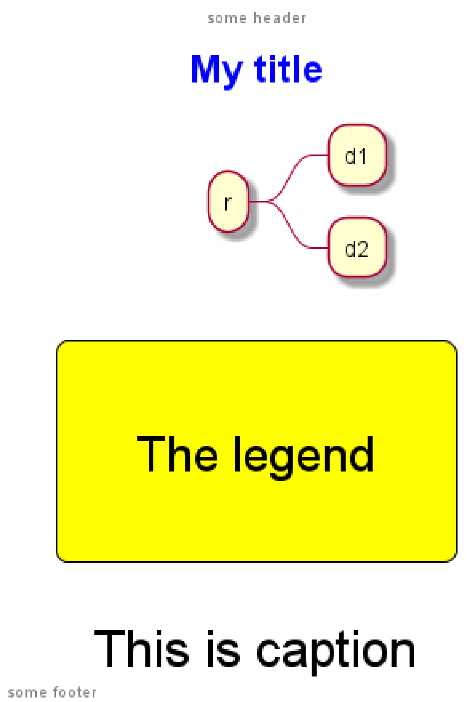
```
  HorizontalAlignment center
  FontSize 26
  FontColor purple
}

footer {
  HorizontalAlignment left
  FontSize 28
  FontColor red
}

legend {
  FontSize 30
  BackGroundColor yellow
  Margin 30
  Padding 50
}

caption {
  FontSize 32
}
</style>
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

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

@enduml
```



20.8.7 マインドマップ

```
@startmindmap
<style>
title {
  HorizontalAlignment right
  FontSize 24
  FontColor blue
}

header {
  HorizontalAlignment center
  FontSize 26
  FontColor purple
}

footer {
  HorizontalAlignment left
  FontSize 28
  FontColor red
}

legend {
  FontSize 30
  BackGroundColor yellow
  Margin 30
  Padding 50
}

caption {
  FontSize 32
}
</style>
header some header

footer some footer

title My title
```

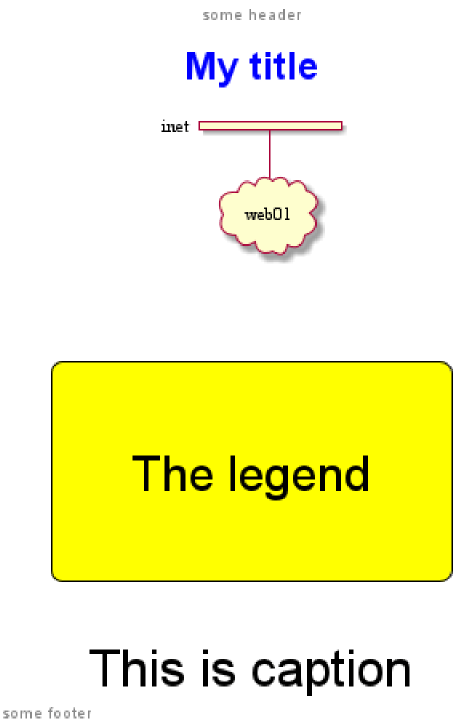
```
caption This is caption

legend
The legend
end legend

* r
** d1
** d2

@endmindmap
```



20.8.8 ネットワーク図（nwdiag）

```
@startuml
<style>
title {
  HorizontalAlignment right
  FontSize 24
  FontColor blue
}

header {
  HorizontalAlignment center
  FontSize 26
  FontColor purple
}

footer {
  HorizontalAlignment left
  FontSize 28
  FontColor red
}
```

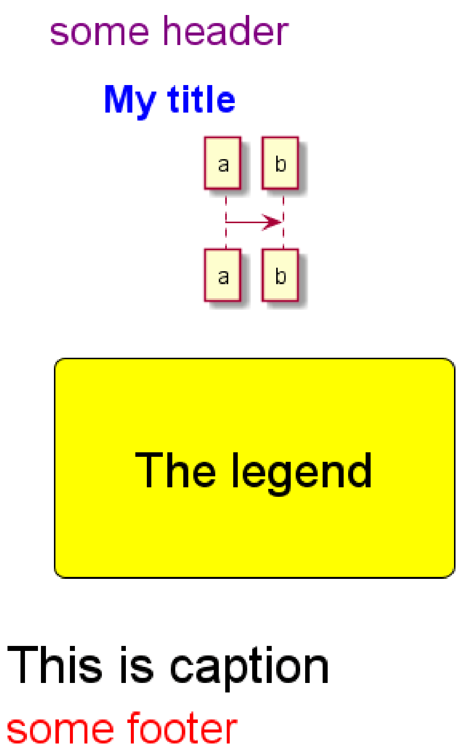
```
legend {
  FontSize 30
  BackGroundColor yellow
  Margin 30
  Padding 50
}

caption {
  FontSize 32
}
</style>
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

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

@enduml
```



20.8.9 シーケンス図

```
@startuml
<style>
title {
```

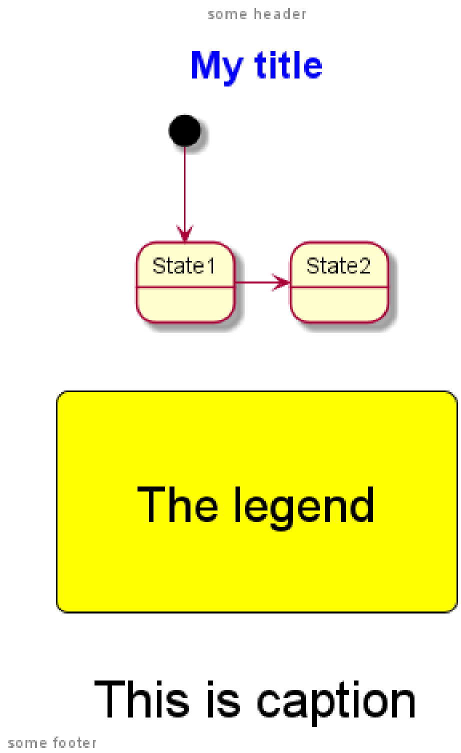
```
  HorizontalAlignment right
  FontSize 24
  FontColor blue
}

header {
  HorizontalAlignment center
  FontSize 26
  FontColor purple
}

footer {
  HorizontalAlignment left
  FontSize 28
  FontColor red
}

legend {
  FontSize 30
  BackGroundColor yellow
  Margin 30
  Padding 50
}

caption {
  FontSize 32
}
</style>
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

a->b
@enduml
```



20.8.10 ステート図

```
@startuml
<style>
title {
  HorizontalAlignment right
  FontSize 24
  FontColor blue
}

header {
  HorizontalAlignment center
  FontSize 26
  FontColor purple
}

footer {
  HorizontalAlignment left
  FontSize 28
  FontColor red
}

legend {
  FontSize 30
  BackGroundColor yellow
  Margin 30
  Padding 50
}

caption {
  FontSize 32
}
</style>
```

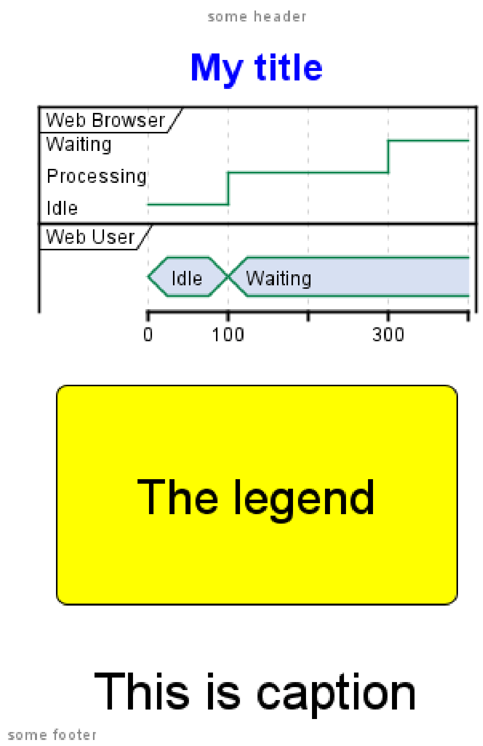
```
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

[*] --> State1
State1 -> State2

@enduml
```



20.8.11 タイミング図

```
@startuml
<style>
title {
  HorizontalAlignment right
  FontSize 24
  FontColor blue
}

header {
  HorizontalAlignment center
  FontSize 26
  FontColor purple
}

footer {
  HorizontalAlignment left
```

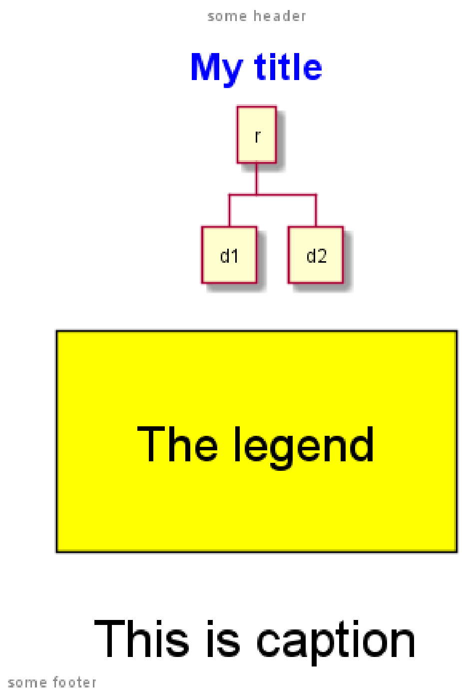
```
  FontSize 28
  FontColor red
}

legend {
  FontSize 30
  BackGroundColor yellow
  Margin 30
  Padding 50
}

caption {
  FontSize 32
}
</style>
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

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

@0
WU is Idle
WB is Idle

@100
WU is Waiting
WB is Processing

@300
WB is Waiting

@enduml
```



20.8.12 Work Breakdown Structure (WBS)

```
@startwbs
<style>
title {
  HorizontalAlignment right
  FontSize 24
  FontColor blue
}

header {
  HorizontalAlignment center
  FontSize 26
  FontColor purple
}

footer {
  HorizontalAlignment left
  FontSize 28
  FontColor red
}

legend {
  FontSize 30
  BackGroundColor yellow
  Margin 30
  Padding 50
}

caption {
  FontSize 32
}
</style>
```

```
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

* r
** d1
** d2

@endwbs
```

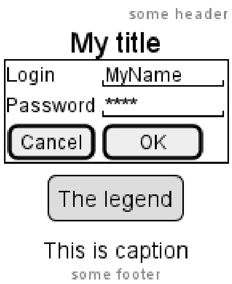


20.8.13 Wireframe (SALT)

TODO: FIXME Fix all (**`title`**, **`caption`**, **`legend`**, **`header`**, **`footer`**) for salt. **TODO:** FIXME

```
@startsalt
<style>
title {
  HorizontalAlignment right
  FontSize 24
  FontColor blue
}

header {
  HorizontalAlignment center
  FontSize 26
  FontColor purple
}
```

```
footer {
  HorizontalAlignment left
  FontSize 28
  FontColor red
}

legend {
  FontSize 30
  BackGroundColor yellow
  Margin 30
  Padding 50
}

caption {
  FontSize 32
}
</style>
@startsalt
header some header

footer some footer

title My title

caption This is caption

legend
The legend
end legend

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



21 Creole

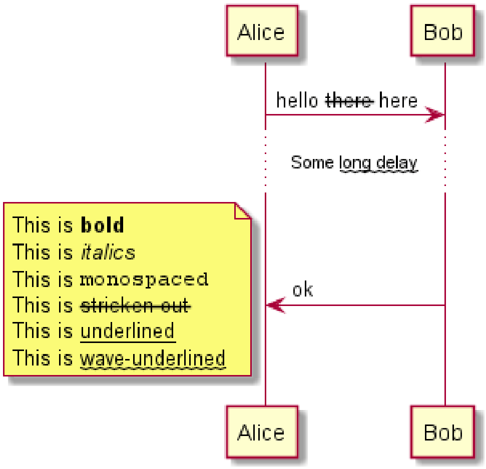
Creole は、さまざまな wiki で使われる、軽量の共通マークアップ言語です。PlantUML には、軽量の Creole エンジンが内蔵されており、テキストのスタイルを統一的な方法で定義することができます。

この構文はすべてのダイアグラムでサポートされています。

後方互換性のため、HTML 構文も残されていることにご注意ください。

21.1 テキストの強調

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



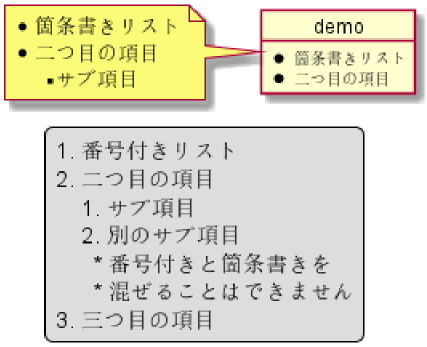
21.2 リスト

ノードのテキストや注釈の中で、番号付きリストと箇条書きリストを使用できます。

TODO: FIXME リストとそのサブリストの中で番号付きリストと箇条書きリストを混ぜて使うとうまくいきません。

```
@startuml
object demo {
  * 箇条書きリスト
  * 二つ目の項目
}
note left
  * 箇条書きリスト
  * 二つ目の項目
  ** サブ項目
end note

legend
  # 番号付きリスト
```

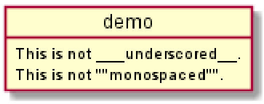
```
  # 二つ目の項目
  ## サブ項目
  ## 別のサブ項目
        * 番号付きと箇条書きを
        * 混ぜることはできません
  # 三つ目の項目
end legend
@enduml
```



21.3 エスケープ文字

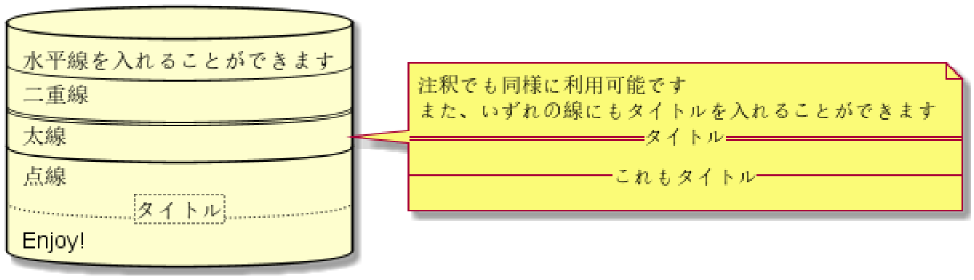
チルダ ~ を使用して、creole の特殊文字をエスケープすることができます。

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



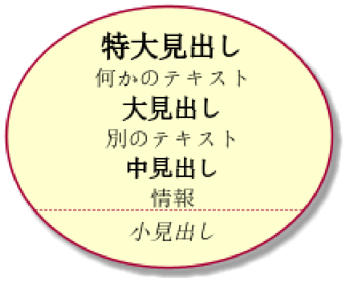
21.4 水平線

```
@startuml
database DB1 as "
水平線を入れることができます
----
二重線
====
太線
____
点線
..タイトル..
Enjoy!
"
note right
  注釈でも同様に利用可能です
  また、いずれの線にもタイトルを入れることができます
  ==タイトル==
  --これもタイトル--
end note

@enduml
```



21.5　見出し

```
@startuml
usecase UC1 as "
= 特大見出し
何かのテキスト
== 大見出し
別のテキスト
=== 中見出し
情報
....
==== 小見出し"
@enduml
```



21.6　レガシー HTML

Creole と同時に、次の HTML タグも利用可能です：

- `<b>` で太字
- `<u>` または `<u:#AAAAAA>` または `<u:[[color|colorName]]>` で下線
- `<i>` でイタリック
- `<s>` または `<s:#AAAAAA>` または `<s:[[color|colorName]]>` で打消し線
- `<w>` または `<w:#AAAAAA>` または `<w:[[color|colorName]]>` で波下線
- `<color:#AAAAAA>` または `<color:[[color|colorName]]>` で文字色
- `<back:#AAAAAA>` または `<back:[[color|colorName]]>` で背景色
- `<size:nn>` でフォントサイズの変更
- `<img:file>`：ファイルシステム中にアクセス可能なファイルを指定
- `<img:http://plantuml.com/logo3.png>`：インターネット上で利用可能な URL を指定

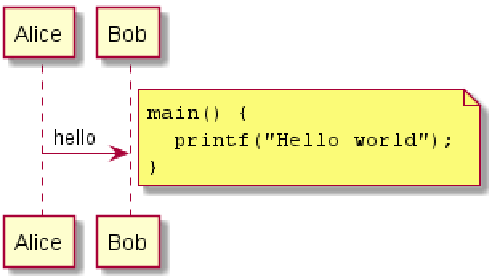
```
@startuml
:* <color:red>文字色</color>の変更
* <back:cadetblue>背景色</back>の変更
* <size:18>フォントサイズ</size>の変更
* <u>legacy</u> <b>HTML <i>tag</i></b>
* <u:red>HTML</u><s:green>タグ中での</s><w:#0000FF>色の使用</w>
```

```
----
* 画像：<img:http://plantuml.com/logo3.png>
;
@enduml
```

21.7 コード

<code> を使用して、ダイアグラム中にプログラミングのコードを記述できます（シンタックスハイライトは未対応です）。

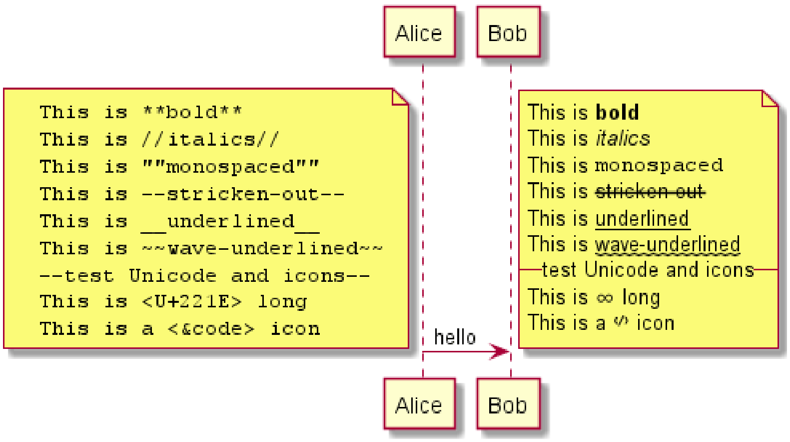
```
@startuml
Alice -> Bob : hello
note right
<code>
main() {
  printf("Hello world");
}
</code>
end note
@enduml
```



これは、PlantUML のコードとその出力結果を表示する場合に、特に便利です：

```
@startuml
Alice -> Bob : hello
note left
<code>
  This is **bold**
  This is //italics//
  This is ""monospaced""
  This is --stricken-out--
  This is __underlined__
  This is ~~wave-underlined~~
  --test Unicode and icons--
```

```
  This is <U+221E> long
  This is a <&code> icon
</code>
end note
note right
  This is **bold**
  This is //italics//
  This is ""monospaced""
  This is --stricken-out--
  This is __underlined__
  This is ~~wave-underlined~~
  --test Unicode and icons--
  This is <U+221E> long
  This is a <&code> icon
end note
@enduml
```

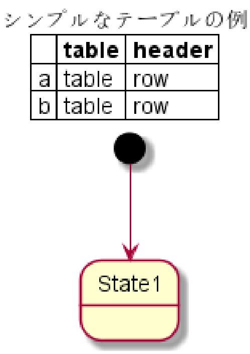


21.8 テーブル

21.8.1 テーブルの作成

| で区切ることで、テーブルを作成できます。

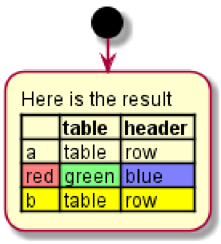
```
@startuml
skinparam titleFontSize 14
title
  シンプルなテーブルの例
  |= |= table |= header |
  | a | table | row |
  | b | table | row |
end title
[*] --> State1
@enduml
```



21.8.2 行とセルの色

行とセルに背景色を設定できます。

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



21.8.3 枠線とテキストの色

枠線とテキストの色を設定することもできます。

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

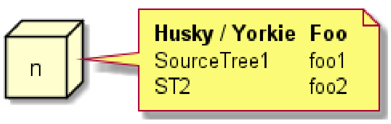
Step	Date	Name	Status	Link
1.1	TBD	plantuml news	Unknown	plantuml news

[Ref. QA-7184]

21.8.4 枠線無し（背景と同色の枠線）

枠線の色を背景色と同じ色に設定することができます。

```
@startuml
node n
note right of n
  <#FBFB77,#FBFB77>|= Husky / Yorkie |= Foo |
  | SourceTree1 | foo1 |
  | ST2 | foo2 |
end note
@enduml
```

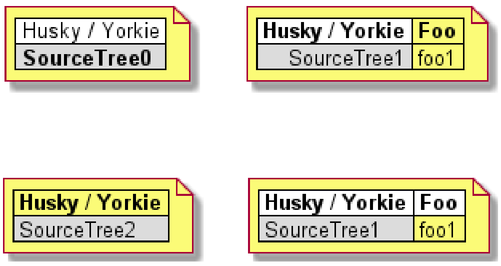


[Ref. QA-12448]

21.8.5 ヘッダを太字にするかどうか

セルの最初の文字を = にすると、太字にすることができます（通常はヘッダを表すために使用します）。

```
@startuml
note as deepCSS0
  |<#white> Husky / Yorkie |
  |=<#gainsboro> SourceTree0 |
endnote

note as deepCSS1
  |= <#white> Husky / Yorkie |= Foo |
  |<#gainsboro><r> SourceTree1 | foo1 |
endnote

note as deepCSS2
  |= Husky / Yorkie |
  |<#gainsboro> SourceTree2 |
endnote

note as deepCSS3
  <#white>|= Husky / Yorkie |= Foo |
  |<#gainsboro> SourceTree1 | foo1 |
endnote
@enduml
```



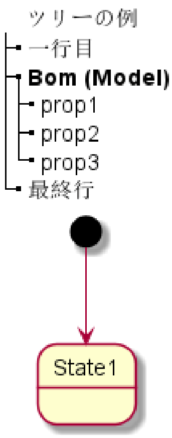
[Ref. QA-10923]

21.9 ツリー

|_ の文字列を使ってツリーを作ることができます。

title のような共通のコマンドに対して：

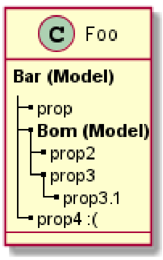
```
@startuml
skinparam titleFontSize 14
title
  ツリーの例
  |_ 一行目
  |_ **Bom (Model)**
    |_ prop1
    |_ prop2
    |_ prop3
  |_ 最終行
end title
[*] --> State1
@enduml
```



クラス図に対して。

(セパレータを使って、空の 2 つ目の区画を作る必要があることに注意してください。そうしないと、(Model) に含まれる括弧によって、テキストが別の区画に移動されてしまいます):

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

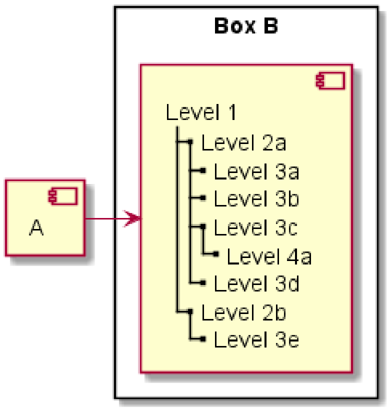


[Ref. QA-3448]

コンポーネント図、配置図に対して：

```
@startuml
[A] as A
rectangle "Box B" {
    component B [
        Level 1
        |_ Level 2a
          |_ Level 3a
          |_ Level 3b
          |_ Level 3c
            |_ Level 4a
          |_ Level 3d
        |_ Level 2b
          |_ Level 3e
    ]
}
```

```
A -> B
@enduml
```

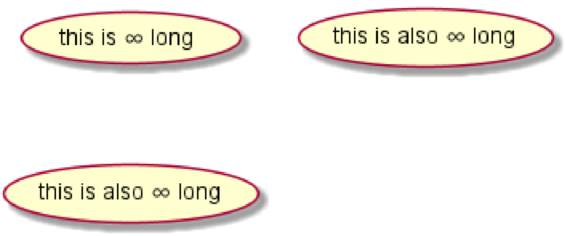


[Ref. QA-11365]

21.10 特殊文字

&#XXXX または、<U+XXXX> の構文、または直接記述することで、任意の Unicode 文字を使うことができます。

```
@startuml
usecase direct as "this is ∞ long"
usecase ampHash as "this is also ∞ long"
usecase angleBrackets as "this is also <U+221E> long"
@enduml
```

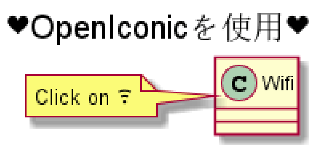


21.11 OpenIconic

OpenIconic はオープンソースの素晴らしいアイコンセットです。これらのアイコンは creole パーサーに組み込まれているので、簡単に使用することができます。

次の構文を使用します：<&ICON_NAME>.

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



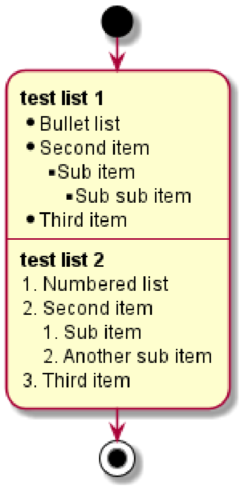
利用可能なアイコンの一覧は、OpenIconic Website にあります。もしくは、次の特別なダイアグラムを使用します：

```
@startuml
listopeniconic
@enduml
```

21.12 Appendix: Examples of "Creole List" on all diagrams

21.12.1 Activity

```
@startuml
start
:**test list 1**
* Bullet list
* Second item
** Sub item
*** Sub sub item
* Third item
----
**test list 2**
# Numbered list
# Second item
## Sub item
## Another sub item
# Third item;
stop
@enduml
```



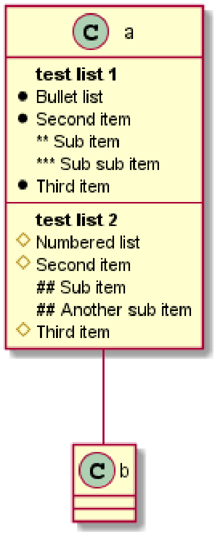
21.12.2 Class

TODO: FIXME

- *Sub item*
- *Sub sub item*

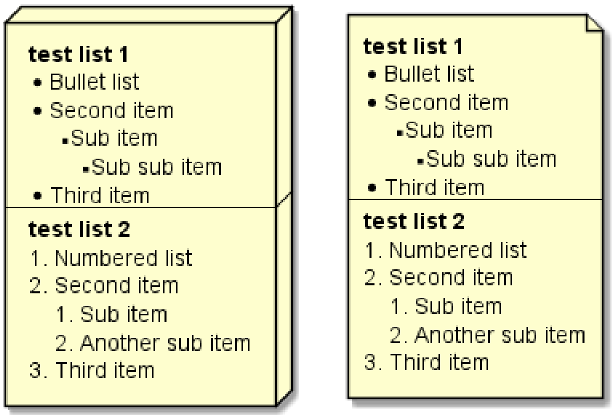
TODO: FIXME

```
@startuml

class a {
**test list 1**
* Bullet list
* Second item
** Sub item
*** Sub sub item
* Third item
----
**test list 2**
# Numbered list
# Second item
## Sub item
## Another sub item
# Third item
}

a -- b

@enduml
```



21.12.3 Component, Deployment, Use-Case

```
@startuml
node n [
**test list 1**
* Bullet list
* Second item
** Sub item
*** Sub sub item
* Third item
----
**test list 2**
# Numbered list
# Second item
## Sub item
## Another sub item
# Third item
]

file f as "
**test list 1**
* Bullet list
* Second item
** Sub item
*** Sub sub item
* Third item
----
**test list 2**
# Numbered list
# Second item
## Sub item
## Another sub item
# Third item
"
@enduml
```



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

21.12.4 Gantt project planning

N/A

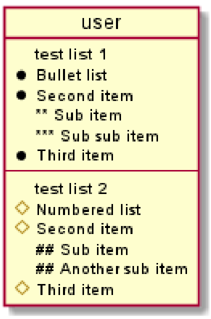
21.12.5 Object

TODO: FIXME

- *Sub item*
- *Sub sub item*

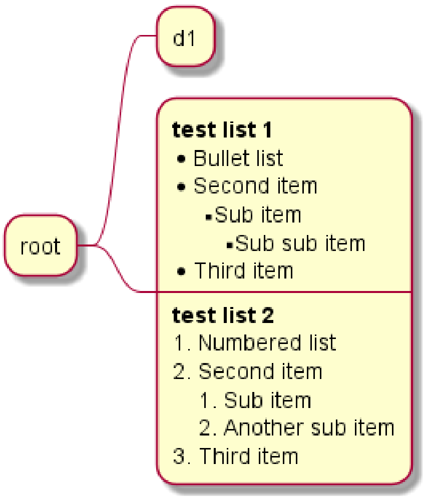
TODO: FIXME

```
@startuml
object user {
**test list 1**
* Bullet list
* Second item
** Sub item
*** Sub sub item
* Third item
----
**test list 2**
# Numbered list
# Second item
## Sub item
## Another sub item
# Third item
}

@enduml
```



21.12.6 MindMap

```
@startmindmap

* root
** d1
**:**test list 1**
* Bullet list
* Second item
** Sub item
*** Sub sub item
* Third item
----
**test list 2**
# Numbered list
# Second item
## Sub item
## Another sub item
# Third item;


@endmindmap
```



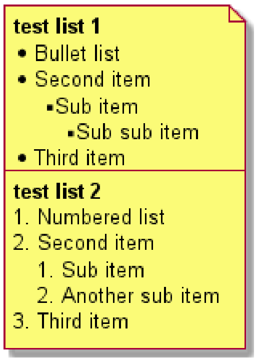
21.12.7 Network (nwdiag)

N/A

21.12.8 Note

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

```
## Another sub item
# Third item
end note
@enduml
```



21.12.9 Sequence

N/A *(or on note or common commands)*

21.12.10 State

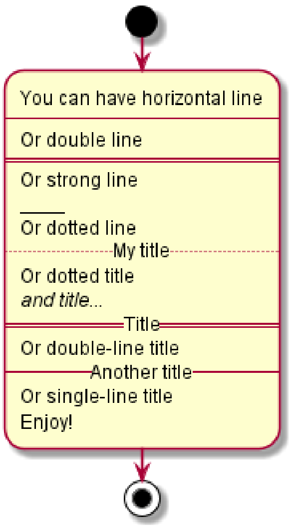
N/A *(or on note or common commands)*

21.13 Appendix: Examples of ”Creole horizontal lines” on all diagrams

21.13.1 Activity

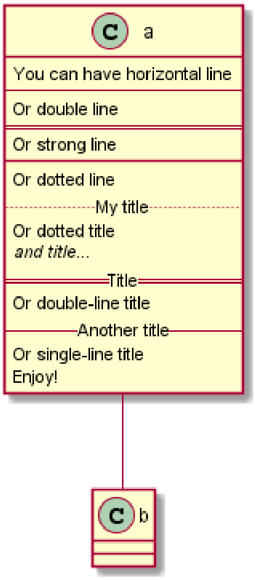
TODO: FIXME strong line ____ **TODO:** FIXME

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



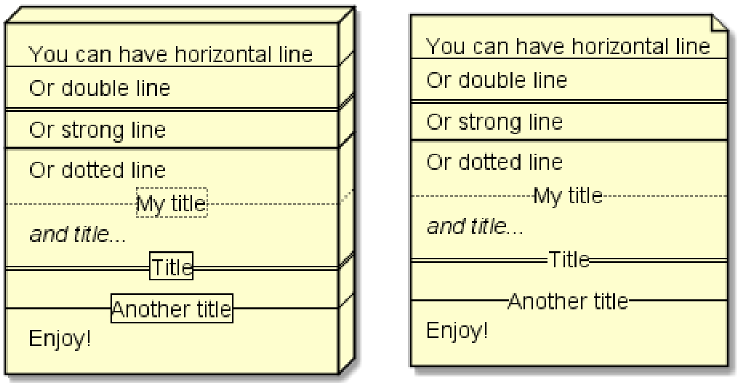
21.13.2 Class

```
@startuml

class a {
You can have horizontal line
----
Or double line
====
Or strong line
____
Or dotted line
..My title..
Or dotted title
//and title... //
==Title==
Or double-line title
--Another title--
Or single-line title
Enjoy!
}

a -- b

@enduml
```



21.13.3 Component, Deployment, Use-Case

```
@startuml
node n [
You can have horizontal line
----
Or double line
====
Or strong line
____
Or dotted line
..My title..
//and title... //
==Title==
--Another title--
Enjoy!
]

file f as "
You can have horizontal line
----
Or double line
====
Or strong line
____
Or dotted line
..My title..
//and title... //
==Title==
--Another title--
Enjoy!
"
@enduml
```

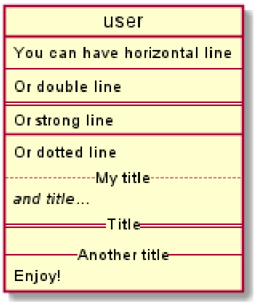


21.13.4 Gantt project planning

N/A

21.13.5 Object

```
@startuml
object user {
You can have horizontal line
----
Or double line
====
Or strong line
____
Or dotted line
..My title..
//and title... //
==Title==
--Another title--
Enjoy!
}

@enduml
```



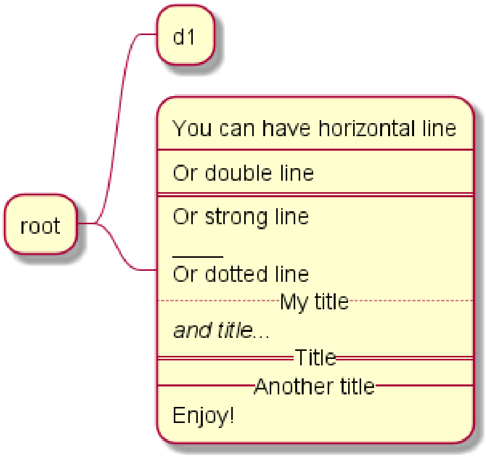
TODO: DONE *[Corrected on V1.2020.18]*

21.13.6 MindMap

TODO: FIXME strong line ____ **TODO:** FIXME

```
@startmindmap

* root
** d1
**:You can have horizontal line
----
```

```
Or double line
====
Or strong line
____
Or dotted line
..My title..
//and title... //
==Title==
--Another title--
Enjoy!;

@endmindmap
```

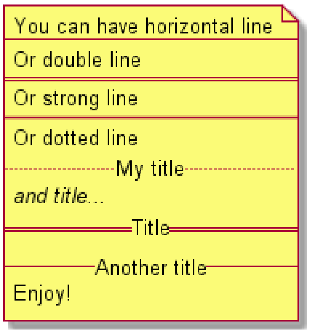


21.13.7 Network (nwdiag)

N/A

21.13.8 Note

```
@startuml
note as n
You can have horizontal line
----
Or double line
====
Or strong line
____
Or dotted line
..My title..
//and title... //
==Title==
--Another title--
Enjoy!
end note
@enduml
```



21.13.9 Sequence

N/A *(or on note or common commands)*

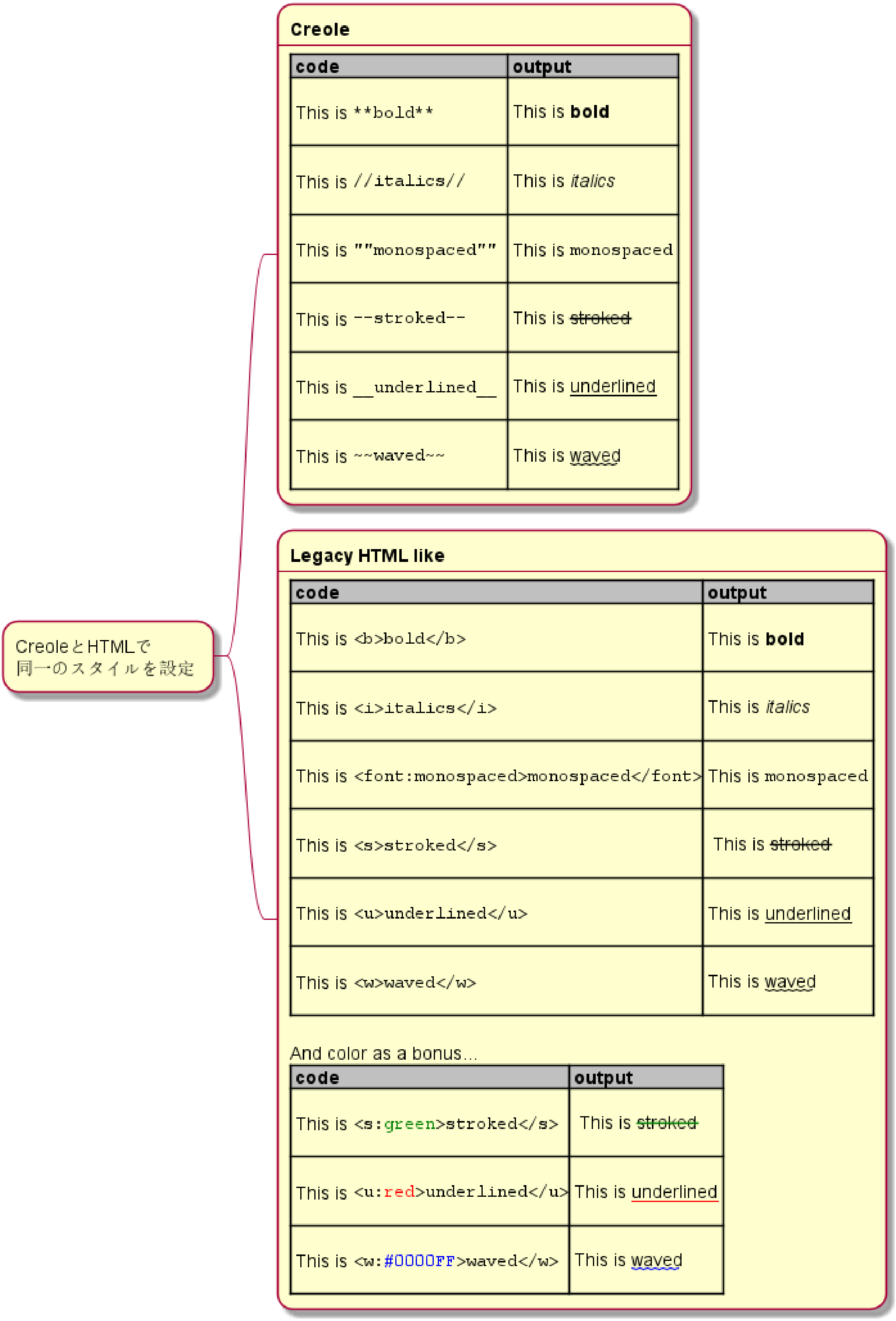
21.13.10 State

N/A *(or on note or common commands)*

21.14 スタイル対応表 (Creole と HTML)

スタイル	Creole	Legacy HTML like
bold	This is `**bold**`	This is `<b>bold</b>`
italics	This is `//italics//`	This is `<i>italics</i>`
`monospaced`	This is `""monospaced""`	This is `<font:monospaced>monospaced</font>`
~~stroked~~	This is `--stroked--`	This is `<s>stroked</s>`
underlined	This is `__underlined__`	This is `<u>underlined</u>`
waved	This is `~~~`	This is `<w>waved</w>`

```
@startmindmap
* CreoleとHTMLで\n同一のスタイルを設定
**:**Creole**
----
<#silver>|= code|= output|
| \n This is ""~**bold**""\n | \n This is **bold** |
| \n This is ""~//italics//""\n | \n This is //italics// |
| \n This is ""~""monospaced~"" ""\n | \n This is ""monospaced"" |
| \n This is ""~--stroked--""\n | \n This is --stroked-- |
| \n This is ""~__underlined__""\n |  \n This is __underlined__ |
| \n This is ""<U+007E><U+007E>waved<U+007E><U+007E>""\n | \n This is ~~waved~~ |;
**:<b>Legacy HTML like
----
<#silver>|= code|= output|
| \n This is ""~<b>bold</b>""\n | \n This is <b>bold</b> |
| \n This is ""~<i>italics</i>""\n | \n This is <i>italics</i> |
| \n This is ""~<font:monospaced>monospaced</font>""\n | \n This is <font:monospaced>monospaced</fon
| \n This is ""~<s>stroked</s>""\n | \n  This is <s>stroked</s> |
| \n This is ""~<u>underlined</u>""\n | \n This is <u>underlined</u> |
| \n This is ""~<w>waved</w>""\n | \n This is <w>waved</w> |

And color as a bonus...
<#silver>|= code|= output|
| \n This is ""~<s:""<color:green>""green""</color>"">stroked</s>""\n | \n  This is <s:green>stroked
| \n This is ""~<u:""<color:red>""red""</color>"">underlined</u>""\n | \n This is <u:red>underlined<
| \n This is ""~<w:""<color:#0000FF>""#0000FF""</color>"">waved</w>""\n | \n This is <w:#0000FF>wave
@endmindmap
```


CreoleとHTMLで
同一のスタイルを設定
Creole
code
output
This is **bold**
This is bold
This is //italics//
This is italics
This is ""monospaced""
This is monospaced
This is --stroked--
This is stroked
This is __underlined__
This is underlined
This is ~~waved~~
This is waved
Legacy HTML like
code
output
This is bold
This is bold
This is <i>italics</i>
This is italics
This is <font:monospaced>monospaced
This is monospaced
This is <s>stroked</s>
This is stroked
This is <u>underlined</u>
This is underlined
This is <w>waved</w>
This is waved
And color as a bonus...
code
output
This is <s:green>stroked</s>
This is stroked
This is <u:red>underlined</u>
This is underlined
This is <w:#0000FF>waved</w>
This is waved

22 Defining and using sprites

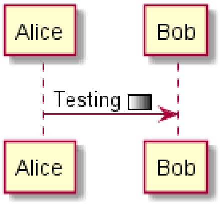
A *Sprite* is a small graphic element that can be used in diagrams.

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

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

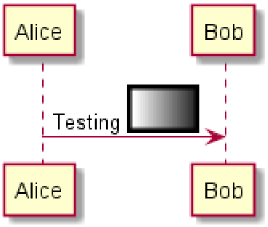
Then you can use the sprite using `<$XXX>` where XXX is the name of the sprite.

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



You can scale the sprite.

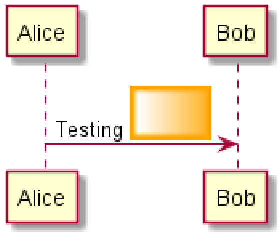
```
@startuml
sprite $foo1 {
  FFFFFFFFFFFFFFF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  F0123456789ABCF
  FFFFFFFFFFFFFFF
}
Alice -> Bob : Testing <$foo1{scale=3}>
@enduml
```



22.1 Changing colors

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

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



22.2 Encoding Sprite

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

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

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

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

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

22.3 Importing Sprite

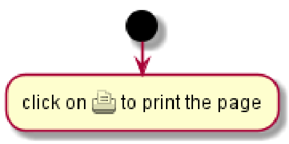
You can also launch the GUI to generate a sprite from an existing image.

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

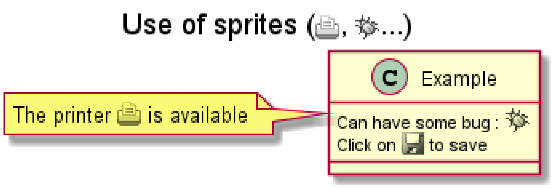
After copying an image into you clipboard, several possible definitions of the corresponding sprite will be displayed : you will just have to pickup the one you want.

22.4 Examples

```
@startuml
sprite $printer [15x15/8z] NOtH3W0W208HxFz_kMAhj7lHWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXwvl
start
:click on <$printer> to print the page;
@enduml
```



```
@startuml
 sprite $bug [15x15/16z] PKzR2i0m2BFMi15p__FEjQEqB1z27aeqCqixa8S4OT7C53cKpsHpaYPDJY_12MHM-BLRyywPhrr
 sprite $printer [15x15/8z] NOtH3W0W208HxFz_kMAhj7lHWpa1XC716sz0Pq4MVPEWfBHIuxP3L6kbTcizR8tAhzaqFvXw
 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
```



22.5 StdLib

The PlantUML StdLib includes a number of ready icons in various IT areas such as architecture, cloud services, logos etc. It including AWS, Azure, Kubernetes, C4, product Logos and many others. To explore these libraries:

- Browse the Github folders of PlantUML StdLib
- Browse the source repos of StdLib collections that interest you. Eg if you are interested in logos you can find that it came from gilbarbara-plantuml-sprites, and quickly find its

sprites-list. (The next section shows how to list selected sprites but unfortunately that's in grayscale whereas this custom listing is in color.)

- Study the in-depth Hitchhiker's Guide to PlantUML, eg sections Standard Library Sprites and PlantUML Stdlib Overview

22.6 Listing Sprites

You can use the `listsprites` command to show available sprites:

- Used on its own, it just shows ArchiMate sprites

- If you include some sprite libraries in your diagram, the command shows all these sprites, as explained in View all the icons with listsprites.

(Example from Hitchhikers Guide to PlantUML)

```
@startuml

!define osaPuml https://raw.githubusercontent.com/Crashedmind/PlantUML-opensecurityarchitecture2-icon
!include osaPuml/Common.puml
!include osaPuml/User/all.puml

listsprites
@enduml
```

Most collections have files called `all` that allow you to see a whole sub-collection at once. Else you need to find the sprites that interest you and include them one by one. Unfortunately, the version of a collection included in StdLib often does not have such `all` files, so as you see above we include the collection from github, not from StdLib.

All sprites are in grayscale, but most collections define specific macros that include appropriate (vendor-specific) colors.

23 Skinparam command

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

Example:

```
skinparam backgroundColor transparent
```

23.1 Usage

You can use this command :

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

23.2 Nested

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

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

is strictly equivalent to:

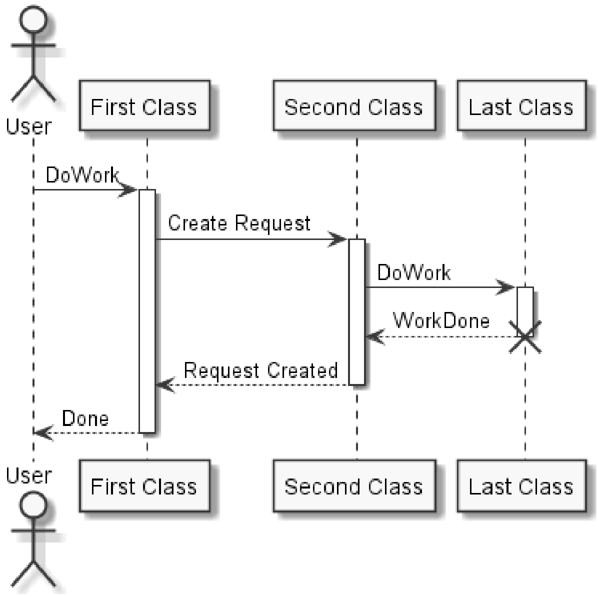
```
skinparam xxxx {
    Param1 value1
    Param2 value2
    Param3 value3
    Param4 value4
}
```

23.3 Black and White

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

```
@startuml

skinparam monochrome true

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

User -> A: DoWork
activate A

A -> B: Create Request
activate B

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

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

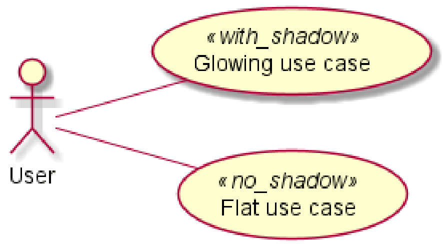
```
A --> User: Done
deactivate A

@enduml
```



23.4 Shadowing

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

```
@startuml

left to right direction

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

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

@enduml
```

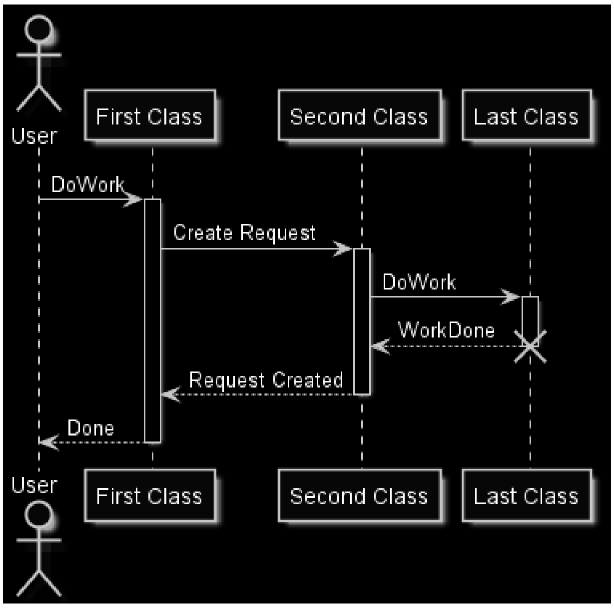


23.5 Reverse colors

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

```
@startuml

skinparam monochrome reverse
```

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

User -> A: DoWork
activate A

A -> B: Create Request
activate B

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

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A

@enduml
```



23.6 Colors

You can use either standard color name or RGB code.

```
@startuml
colors
@enduml
```

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

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

23.7 Font color, name and size

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

Example:

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

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

Example:

```
skinparam defaultFontName Aapex
```

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

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

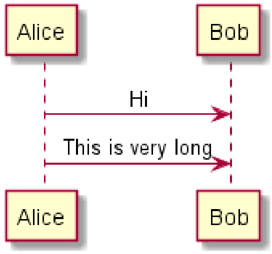
```
java -jar plantuml.jar -language
```

23.8 Text Alignment

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

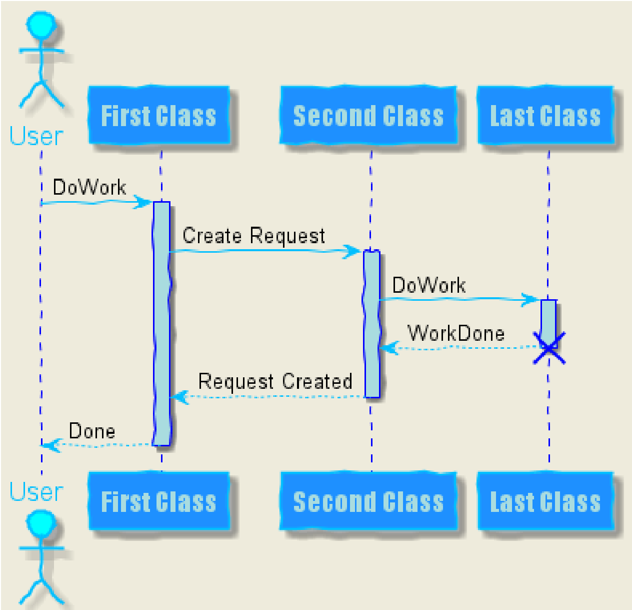
Param name	Default value	Comment
sequenceMessageAlign	left	Used for messages in sequence diagrams
sequenceReferenceAlign	center	Used for `ref over` in sequence diagrams

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

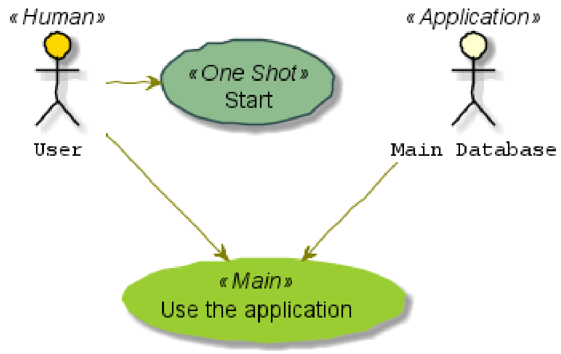


23.9 Examples

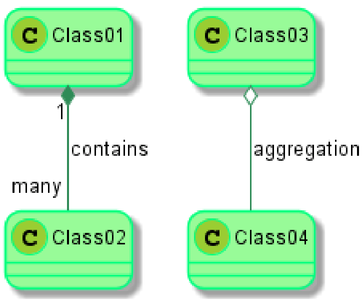
```
@startuml
skinparam backgroundColor #EEEBDC
skinparam handwritten true

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

ParticipantBorderColor DeepSkyBlue
ParticipantBackgroundColor DodgerBlue
ParticipantFontName Impact
ParticipantFontSize 17
ParticipantFontColor #A9DCDF

ActorBackgroundColor aqua
ActorFontColor DeepSkyBlue
ActorFontSize 17
ActorFontName 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
```

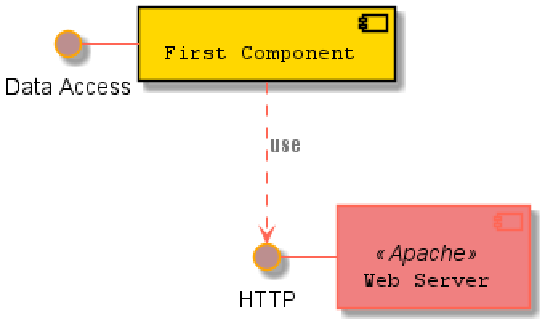



```
@startuml
skinparam roundcorner 20
skinparam class {
BackgroundColor PaleGreen
ArrowColor SeaGreen
BorderColor SpringGreen
}
skinparam stereotypeCBackgroundColor YellowGreen

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

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

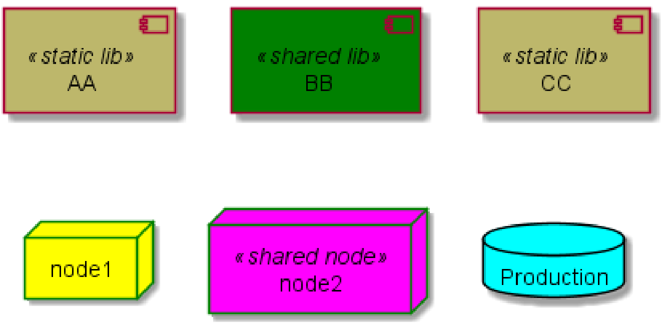



```
@startuml
skinparam interface {
  backgroundColor RosyBrown
  borderColor orange
}

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

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

DA - [First Component]
```

```
[First Component] ..> () HTTP : use
HTTP - [Web Server]
@enduml
```




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

node node1
node node2 <<shared node>>
database Production

skinparam component {
    backgroundColor<<static lib>> DarkKhaki
    backgroundColor<<shared lib>> Green
}

skinparam node {
borderColor Green
backgroundColor Yellow
backgroundColor<<shared node>> Magenta
}
skinparam databaseBackgroundColor Aqua
@enduml
```



23.10 List of all skinparam parameters

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

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

Or you can generate a "diagram" with a list of all the skinparam parameters using `help skinparams`.

That will give you the following result, from this page *(code of this command)*:

- CommandHelpSkinparam.java

```
@startuml
```

```
help skinparams
@enduml
```

Help on skinparam

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

The possible skinparam are :

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

You can also view each skinparam parameters with its results displayed at the page `All Skin Parameters` of `Ashley's PlantUML Doc`:

- https://plantuml-documentation.readthedocs.io/en/latest/formatting/all-skin-params.html.

24 前処理

PlantUML にはいくつかの前処理機能があり、それはすべてのダイアグラムに対して利用可能です。

これらの機能は、特殊文字 `#` がエクスクラメーションマーク`!` に置き換えられていることを除くと、C 言語のプリプロセッサに非常によく似ています。

24.1 以降に関する注意事項

現行のプリプロセッサは、レガシーなプリプロセッサーからアップデートしたものです。

レガシーな機能の内いくつかは、現行のプリプロセッサでもサポートされていますが、今後はそれらを使わないようにしてください (将来的に削除される可能性があります)。

- `!define` と`!definelong` は使用しないでください。代わりに`!function`、!procedure?code?? もしくは変数定義を使用します。
  - `!define` は return`!function` に置き換えてください。
  - `!definelong` は`!procedure` に置き換えてください。
- `!include` で複数のインクルードが可能になったので、`!include_many` を使う必要はありません。
- `!include` は URL を受け取れるようになったので`!includeurl` を使う必要はありません。
- `%date%` などのいくつかの機能は、組み込みの関数 (`%date()` など) に置き換えられました。
- レガシーな`!definelong` マクロを引数無しで呼ぶ場合、必ず括弧を使用する必要があります.`my_own_definelong()` ではなく `my_own_definelong` のように括弧を省略してしまうと、新しいプリプロセッサでは認識されません。

Please contact us if you have any issues.

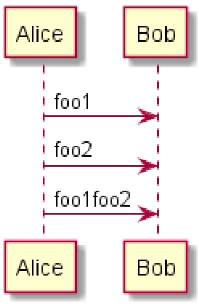
24.2 変数定義

これは必須ではありませんが、変数名を `$` で始めることを強く推奨します。データの種類は 2 つあります。

- 整数
- 文字列 - シングルクォートもしくはダブルクォートで囲んでください

関数の外側に作られた変数はグローバルであり、関数内を含むどこからでもアクセスすることができます。このことを明示するために、変数定義に `global` というキーワードを付けることもできます。

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

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



24.3 Boolean expression

24.3.1 Boolean represention [0 is false]

There is not real boolean type, but PlantUML use this integer convention:

- Integer `0` means **false**
- and any non-null number (as `1`) or any string (as `"1"`, or even `"0"`) means **true**.

[Ref. QA-9702]

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

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

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

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

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

For convenience, you can use those boolean builtin functions:

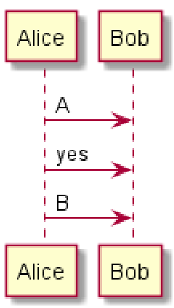
- `%false()`
- `%true()`
- `%not(<exp>)`

[See also Builtin functions]

24.4 Conditions [!if, !else, !elseif, !endif]

- You can use expression in condition.
- *else* and *elseif* are also implemented

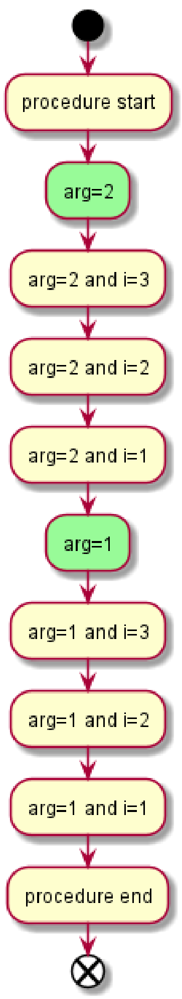
```
@startuml
!$a = 10
!$ijk = "foo"
Alice -> Bob : A
!if ($ijk == "foo") && ($a+10>=4)
Alice -> Bob : yes
!else
Alice -> Bob : This should not appear
!endif
Alice -> Bob : B
@enduml
```



24.5 While loop [!while, !endwhile]

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

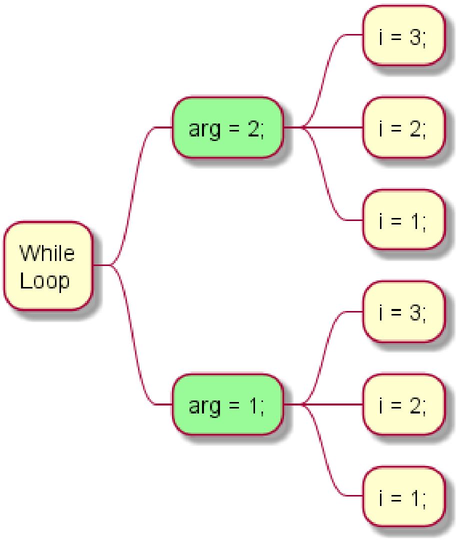
```
@startuml
!procedure $foo($arg)
  :procedure start;
  !while $arg!=0
    !$i=3
    #palegreen:arg=$arg;
    !while $i!=0
      :arg=$arg and i=$i;
      !$i = $i - 1
    !endwhile
    !$arg = $arg - 1
  !endwhile
  :procedure end;
!endprocedure

start
$foo(2)
end
@enduml
```



[Adapted from QA-10838]

```
@startmindmap
!procedure $foo($arg)
```

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

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



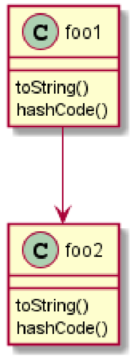
24.6 Procedure [!procedure, !endprocedure]

- Procedure names *should* start with a $
- Argument names *should* start with a $
- Procedures can call other procedures

Example:

```
@startuml
!procedure $msg($source, $destination)
  $source --> $destination
!endprocedure

!procedure $init_class($name)
  class $name {
    $addCommonMethod()
  }
!endprocedure


!procedure $addCommonMethod()
  toString()
  hashCode()
```

```
!endprocedure


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



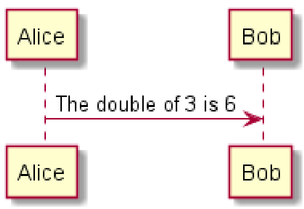
Variables defined in procedures are **local**. It means that the variable is destroyed when the procedure ends.

24.7 Return function [!function, !endfunction]

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

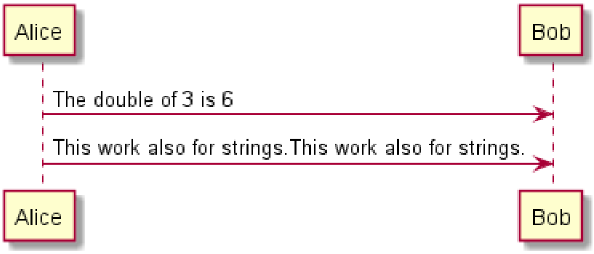
- directly in variable definition or in diagram text
- from other return functions
- from procedures
- Function name *should* start with a $
- Argument names *should* start with a $

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

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



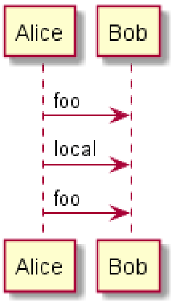
It is possible to shorten simple function definition in one line:

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

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



As in procedure (void function), 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.

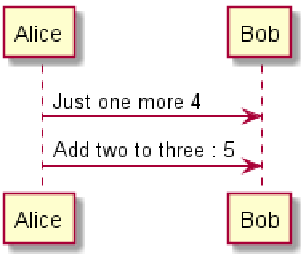
```
@startuml
!function $dummy()
!local $ijk = "local"
!return "Alice -> Bob : " + $ijk
!endfunction

!global $ijk = "foo"

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



24.8 Default argument value

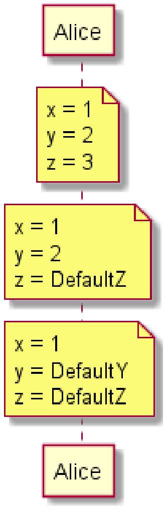
In both procedure and return functions, you can define default values for arguments.

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

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



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

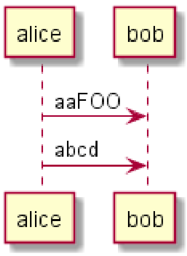
```
@startuml
!procedure defaulttest($x, $y="DefaultY", $z="DefaultZ")
note over Alice
  x = $x
  y = $y
  z = $z
end note
!endprocedure

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



24.9 Unquoted procedure or function [!unquoted]

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

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

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

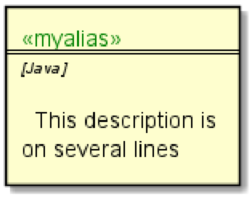


24.10 Keywords arguments

Like in Python, you can use keywords arguments :

```
@startuml

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

```
<color:$colour><<$alias>></color>
==$label==
//<size:$size>[$technology]</size>//

  $description"
!endprocedure

$element(myalias, "This description is %newline()on several lines", $size=10, $technology="Java")
@enduml
```

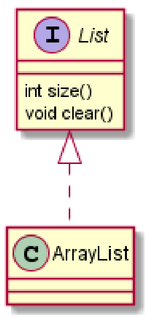


24.11 Including files or URL [!include, !include_many, !include_once]

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

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

```
@startuml

interface List
List : int size()
List : void clear()
List <|.. ArrayList
@enduml
```



File List.iuml

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

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

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

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

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

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

24.12 Including Subpart [!startsub, !endsub, !includesub]

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

file1.puml:

```
@startuml

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

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

```
@startuml

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

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

file2.puml

```
@startuml

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

This file would be rendered exactly as if:

```
@startuml

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

24.13 Builtin functions [%]

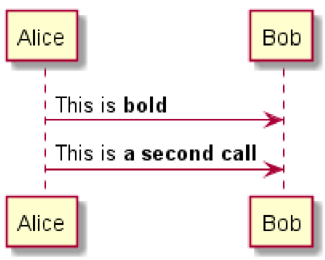
Some functions are defined by default. Their name starts by `%`

Name	Description	Example
`%date`	Retrieve current date. You can provide an optional format for the date	`%date("yyyy.MM.d`
`%dirpath`	Retrieve current dirpath	`%dirpath()`
`%false`	Return always `false`	`%false()`
`%file_exists`	Check if a file exists on the local filesystem	`%file_exists("c:`
`%filename`	Retrieve current filename	`%filename()`
`%function_exists`	Check if a function exists	`%function_exists`
`%get_variable_value`	Retrieve some variable value	`%get_variable_va`
`%getenv`	Retrieve environment variable value	`%getenv("OS")`
`%intval`	Convert a String to Int	`%intval("42")`
`%lower`	Return a lowercase string	`%lower("Hello")`
`%newline`	Return a newline	`%newline()`
`%not`	Return the logical negation of an expression	`%not(2+2==4)`
`%set_variable_value`	Set a global variable	`%set_variable_va`
`%string`	Convert an expression to String	`%string(1 + 2)`
`%strlen`	Calculate the length of a String	`%strlen("foo")`
`%strpos`	Search a substring in a string	`%strpos("abcdef"`
`%substr`	Extract a substring. Takes 2 or 3 arguments	`%substr("abcdef"`
`%true`	Return always `true`	`%true()`
`%upper`	Return an uppercase string	`%upper("Hello")`
`%variable_exists`	Check if a variable exists	`%variable_exists`
`%version`	Return PlantUML current version	`%version()`

24.14 Logging [!log]

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

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

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

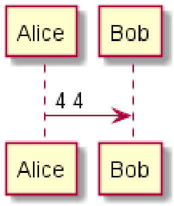


24.15 Memory dump [!memory_dump]

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

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

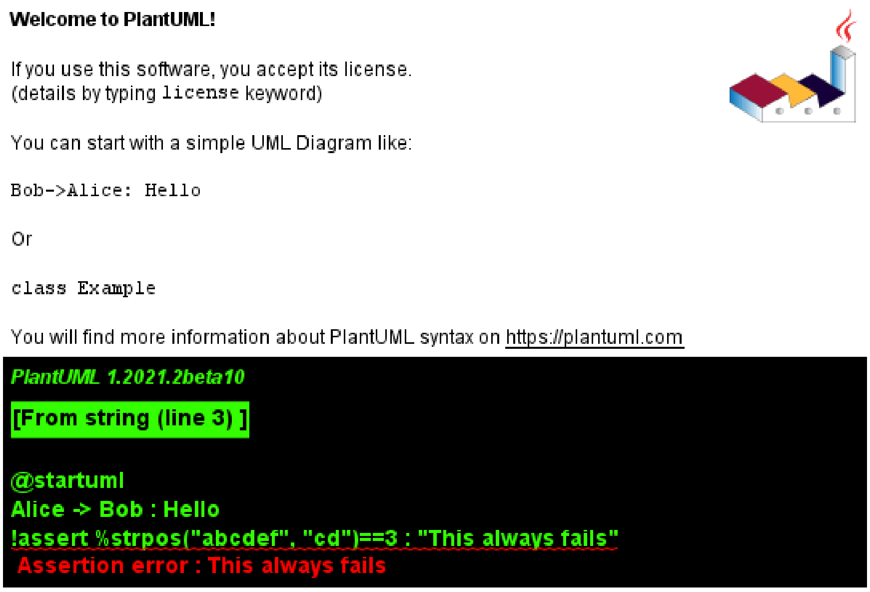
```
!dump_memory
!return $val+1
!endfunction

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



24.16 Assertion [!assert]

You can put assertions in your diagram.

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



24.17 Building custom library [!import, !include]

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

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

Example:

```
@startuml

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

!include myFolder/myFile.iuml
' Assuming that myFolder/myFile.iuml is located somewhere
```

```
' either inside "customLibrary.zip" or on the local filesystem

...
```

24.18 Search path

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

For example:

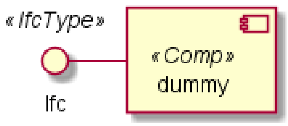
```
java -Dplantuml.include.path="c:/mydir" -jar plantuml.jar atest1.txt
```

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

24.19 Argument concatenation [##]

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

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

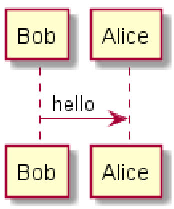


24.20 Dynamic invocation [%invoke_procedure(), %call_user_func()]

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

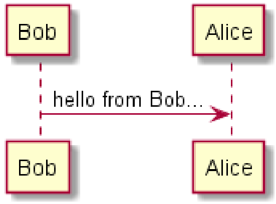
For example, you can have:

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

!$wrapper = "$go"

%invoke_procedure($wrapper)
@enduml
```



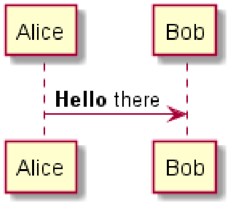

```
@startuml
!procedure $go($txt)
  Bob -> Alice : $txt
!endprocedure
```

```
%invoke_procedure("$go", "hello from Bob...")
@enduml
```



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

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

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



24.21 Evaluation of addition depending of data types [+]

Evaluation of `$a + $b` depending of type of `$a` or `$b`

```
@startuml
title
<#LightBlue>|= |=  $a |=  $b |=  <U+0025>string($a + $b)|
<#LightGray>| type | str | str | str (concatenation) |
| example |= "a" |= "b" |= %string("a" + "b") |
<#LightGray>| type | str | int | str (concatenation) |
| ex.|= "a" |=  2  |= %string("a" + 2)    |
<#LightGray>| type | str | int | str (concatenation) |
| ex.|=  1  |= "b" |= %string(1 + "b")    |
<#LightGray>| type | bool | str | str (concatenation) |
| ex.|= <U+0025>true() |= "b" |= %string(%true() + "b") |
<#LightGray>| type | str | bool | str (concatenation) |
| ex.|= "a" |= <U+0025>false() |= %string("a" + %false()) |
<#LightGray>| type |  int  |  int | int (addition of int) |
| ex.|=  1  |=  2  |= %string(1 + 2)      |
<#LightGray>| type |  bool  |  int | int (addition) |
| ex.|= <U+0025>true() |= 2 |= %string(%true() + 2) |
<#LightGray>| type |  int  |  bool | int (addition) |
| ex.|=  1  |= <U+0025>false() |= %string(1 + %false()) |
<#LightGray>| type |  int  |  int | int (addition) |
| ex.|=  1  |=  <U+0025>intval("2")  |= %string(1 + %intval("2")) |
end title
@enduml
```

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

24.22 Preprocessing JSON

You can extend the functionality of the current Preprocessing with JSON Preprocessing features:

- JSON Variable definition
- Access to JSON data
- Loop over JSON array

(See more details on Preprocessing-JSON page)

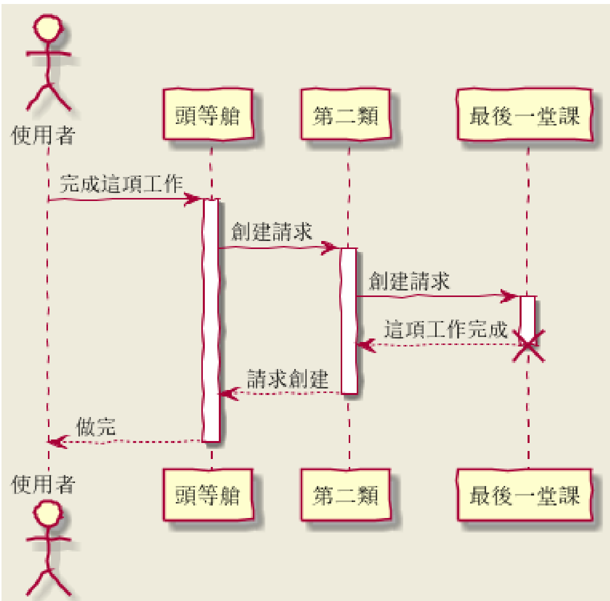
25 Unicode

PlantUML 言語ではアクターやユースケースなどを定義するのに「文字」を使用します。

しかし「文字」は A~Z のアルファベットに限定されません。あらゆる言語のあらゆる文字を使用することができます。

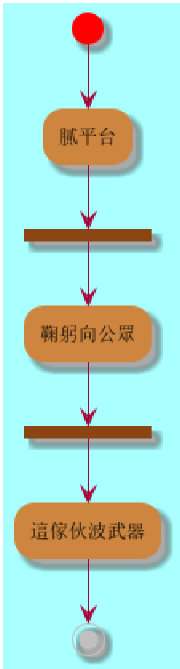
25.1 例

```
@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
```

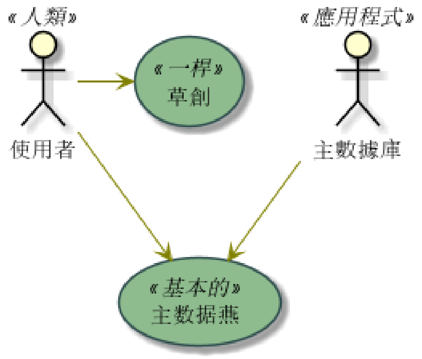



```
@startuml

(*) --> "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
```



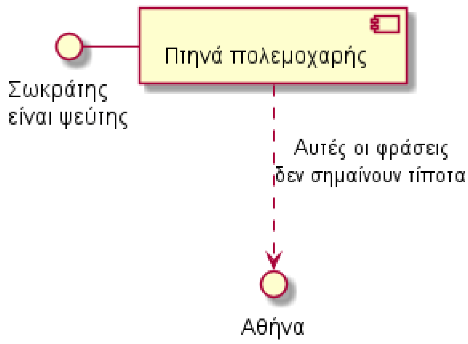
@startuml

() ”Σ ” as Σ

Σ - [Π]

[Π] ..> () A : A

@enduml



25.2 文字コード

UML を含むテキストを読み込むために使用するデフォルトの文字コード（charset）は、システムに依存しています。

通常、それで問題は起きないはずですが、他の文字コードを使用したい場合もあるでしょう。例えば、次のようなコマンドを使用します：

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

Ant タスクを使用する場合：

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

インストールした Java 環境によりますが、次の文字コードが使用可能なはずです：ISO-8859-1、UTF-8、UTF-16BE、UTF-16LE、UTF-16。

26 標準ライブラリ

このページでは、PlantUML の標準ライブラリ (stdlib) について説明します。標準ライブラリは PlantUML の公式リリースに含まれています。含まれるファイルは、" 標準 C ライブラリ" の規約に従っています。

ライブラリの内容は、サードパーティのコントリビュータによるものです。貢献して下さった方々に感謝します。

26.1 標準ライブラリの一覧

特別なダイアグラムを使用して、標準ライブラリのフォルダ一覧を得ることができます:

```
@startuml
stdlib
@enduml
```

コマンドラインから `java -jar plantuml.jar -stdlib` を実行することでも同じ一覧を得ることができます。

`java -jar plantuml.jar -extractstdlib` を実行すると、標準ライブラリのすべてのソースを取り出すことができます。すべてのファイルは `stdlib` フォルダに出力されます。

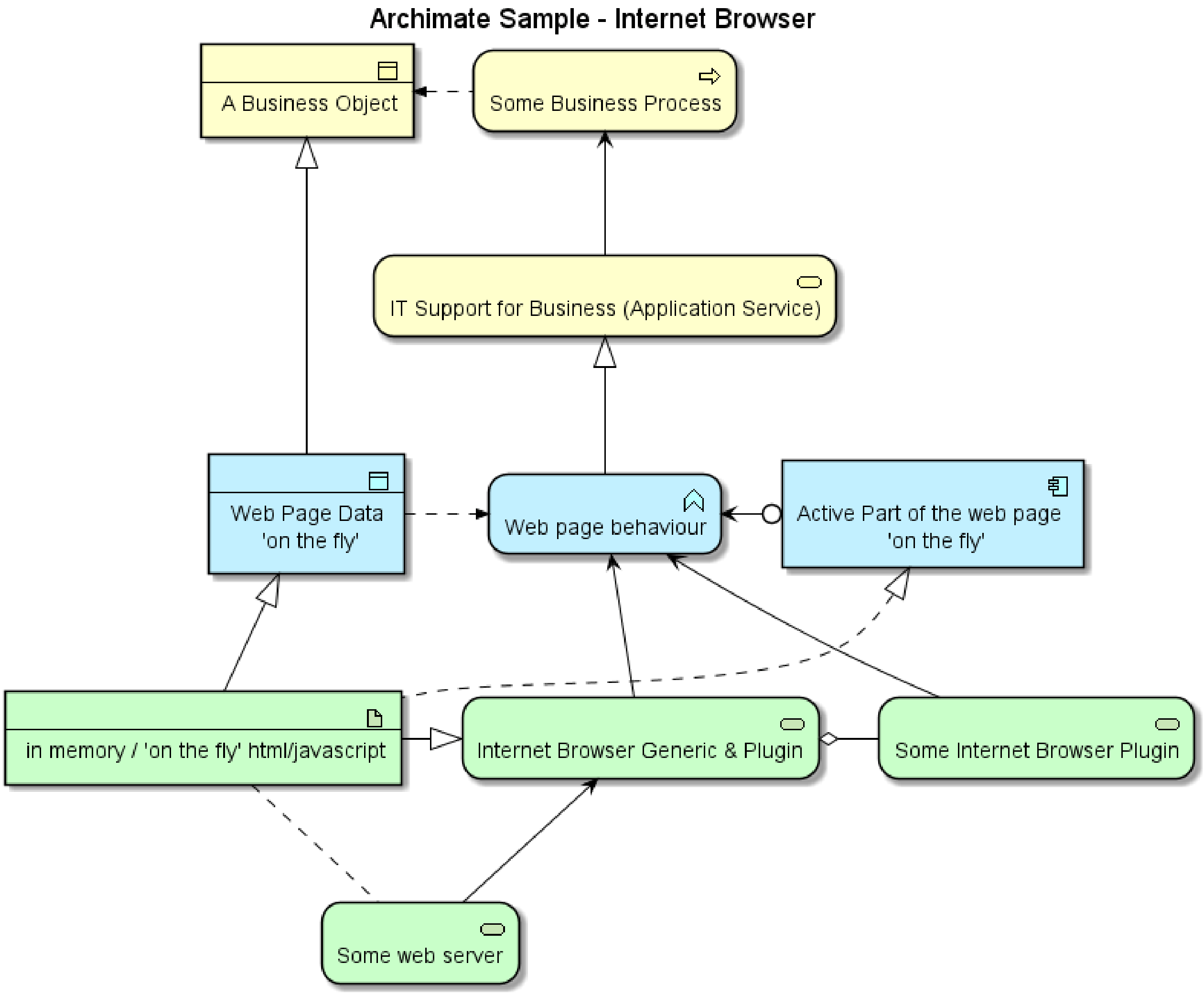
公式の PlantUML リリースのビルドに使用されるソースは、https://github.com/plantuml/plantuml-stdlib にホストされています。ライブラリのアップデートや関連するライブラリを追加したい場合は、プルリクエストを作成することができます。

26.2 ArchiMate (archimate)

- https://github.com/plantuml/plantuml-stdlib/tree/master/archimate
- https://github.com/ebbypeter/Archimate-PlantUML

このリポジトリには ArchiMate の PlantUML マクロ、および、ArchiMate の図を簡単に一貫性をもって作成するためのその他のインクルードが含まれています。

```
@startuml
!include <archimate/Archimate>

title Archimate Sample - Internet Browser

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

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

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

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



26.3 AWS ライブラリ (aws)

- https://github.com/plantuml/plantuml-stdlib/tree/master/aws
- https://github.com/milo-minderbinder/AWS-PlantUML

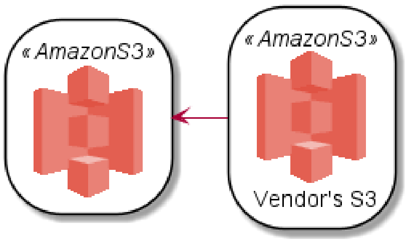
AWS ライブラリには Amazon AWS のアイコンが含まれています。それぞれ 2 つの異なるサイズ (normal と large) のアイコンがあります。

スプライトの含まれるファイルをインポートして使います。例: `!include <aws/Storage/AmazonS3/AmazonS3>`. インポートされると、通常のスプライトと同様、`<$sprite_name>` のように使用することができます。

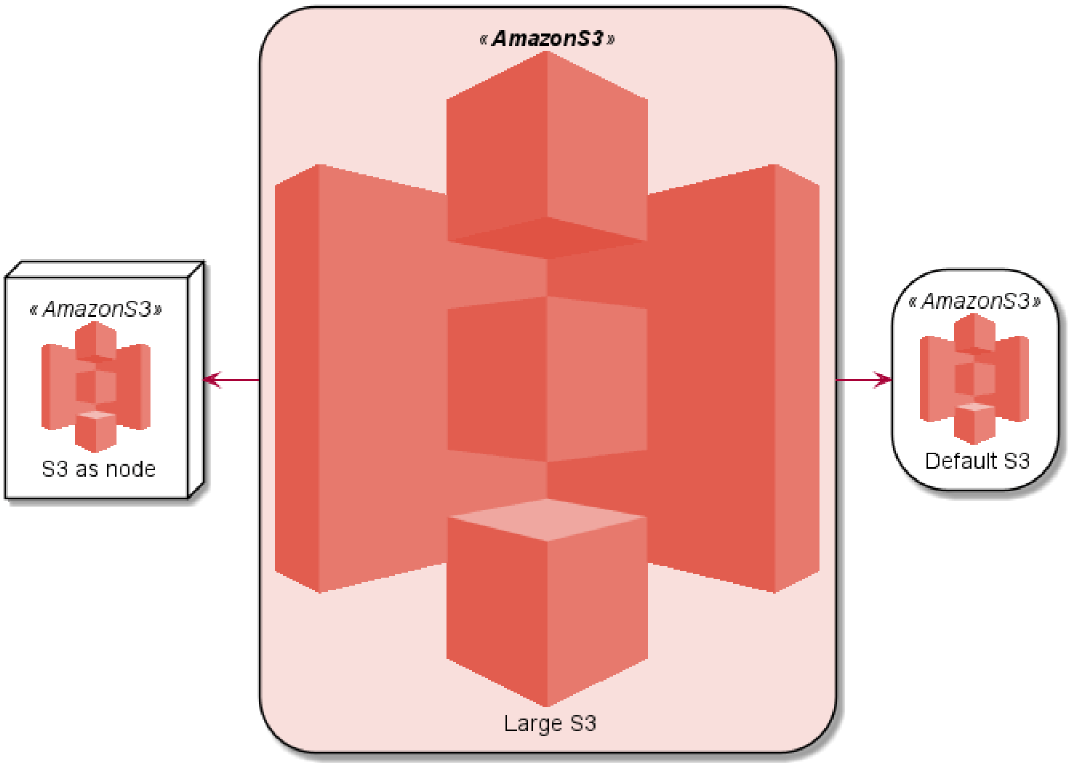
`!include <aws/common>` のようにして `common.puml` をインクルードすると、そこに定義されたヘルパーマクロを使用することができます。`common.puml` をインポートすると、`NAME_OF_SPRITE(parameters...)` マクロを使用することができます。

使用例:

```
@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
```




```
@startuml
!include <aws/common>
!include <aws/Storage/AmazonS3/AmazonS3>
!include <aws/Storage/AmazonS3/AmazonS3_LARGE>

skinparam nodeBackgroundColor White
skinparam storage<<**AmazonS3**>> {
    backgroundColor #F9DFDC
}
AMAZONS3(s3_internal,"Default S3")
AMAZONS3(s3_internal2,"S3 as node",node)
AMAZONS3_LARGE(s3_partner,"Large S3")

s3_internal2 <-r- s3_partner
s3_internal <-l- s3_partner
@enduml
```



26.4 Amazon Labs AWS ライブラリ (awslib)

- https://github.com/plantuml/plantuml-stdlib/tree/master/awslib
- https://github.com/awslabs/aws-icons-for-plantuml

Amazon Labs AWS ライブラリは、PlantUML のスプライト、マクロ、その他の Amazon Web Services(AWS) 向けサービスとリソースを提供します。

AWS コンポーネントを含む PlantUML ダイアグラムを作成するために使用します。すべての要素は

公式の AWS Architecture Icons から生成されていて、PlantUML や C4 model と組み合わせることで、設計、デプロイ、トポロジーをコードとして伝えるための素晴らしい手段となります。

```
@startuml
'Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
'SPDX-License-Identifier: MIT (For details, see https://github.com/awslabs/aws-icons-for-plantuml/bl

!include <awslib/AWSCommon>

' Uncomment the following line to create simplified view
' !include <awslib/AWSSimplified>

!include <awslib/General/Users>
!include <awslib/Mobile/APIGateway>
!include <awslib/SecurityIdentityAndCompliance/Cognito>
!include <awslib/Compute/Lambda>
!include <awslib/Database/DynamoDB>

left to right direction

Users(sources, "Events", "millions of users")
APIGateway(votingAPI, "Voting API", "user votes")
Cognito(userAuth, "User Authentication", "jwt to submit votes")
Lambda(generateToken, "User Credentials", "return jwt")
Lambda(recordVote, "Record Vote", "enter or update vote per user")
DynamoDB(voteDb, "Vote Database", "one entry per user")

sources --> userAuth
sources --> votingAPI
userAuth <--> generateToken
votingAPI --> recordVote
recordVote --> voteDb
@enduml
```

26.5 Azure ライブラリ (azure)

- https://github.com/plantuml/plantuml-stdlib/tree/master/azure
- https://github.com/RicardoNiepel/Azure-PlantUML/

Azure ライブラリには Microsoft Azure のアイコンが含まれます。

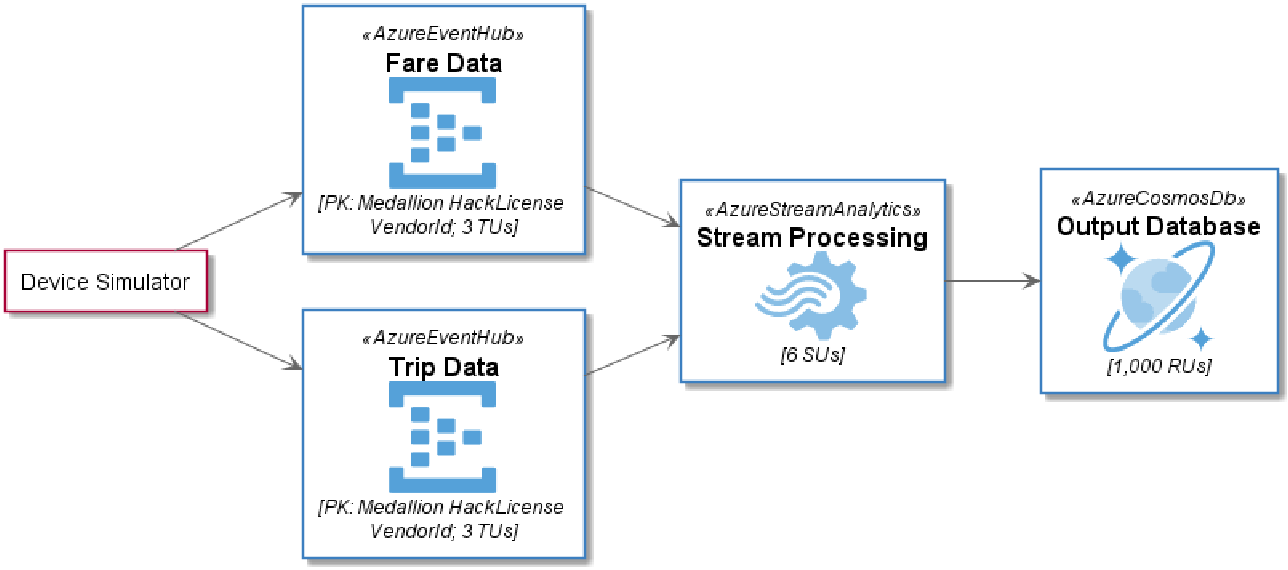
スプライトの含まれるファイルをインポートして使います。例: `!include <azure/Analytics/AzureEventHub.puml>` インポートされると、通常のスプライトと同様、`<$sprite_name>` のように使用することができます。

`!include <aws/common>` のようにして `AzureCommon.puml` をインクルードすると、そこに定義されたヘルパーマクロを使用することができます。`AzureCommon.puml` をインポートすると、`NAME_OF_SPRITE(parameters...)` マクロを使用することができます。

使用例:

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

left to right direction

agent "Device Simulator" as devices #fff

AzureEventHub(fareDataEventHub, "Fare Data", "PK: Medallion HackLicense VendorId; 3 TUs")
```

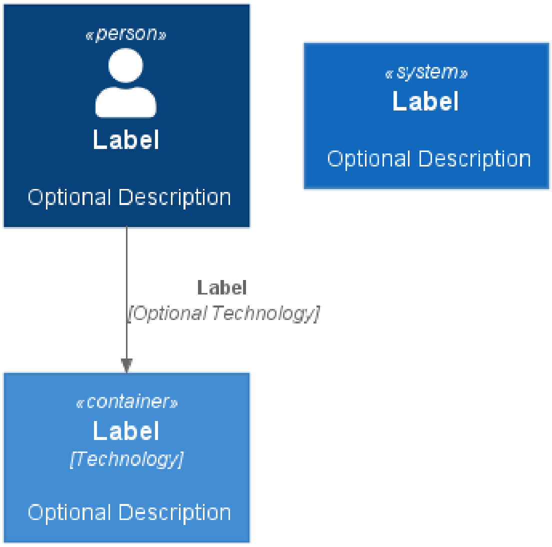
```
AzureEventHub(tripDataEventHub, "Trip Data", "PK: Medallion HackLicense VendorId; 3 TUs")
AzureStreamAnalytics(streamAnalytics, "Stream Processing", "6 SUs")
AzureCosmosDb(outputCosmosDb, "Output Database", "1,000 RUs")

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



26.6 C4 ライブラリ (C4)

- https://github.com/plantuml/plantuml-stdlib/tree/master/C4
- https://github.com/plantuml-stdlib/C4-PlantUML

```
@startuml
!include <C4/C4_Container>

Person(personAlias, "Label", "Optional Description")
Container(containerAlias, "Label", "Technology", "Optional Description")
System(systemAlias, "Label", "Optional Description")

Rel(personAlias, containerAlias, "Label", "Optional Technology")
@enduml
```

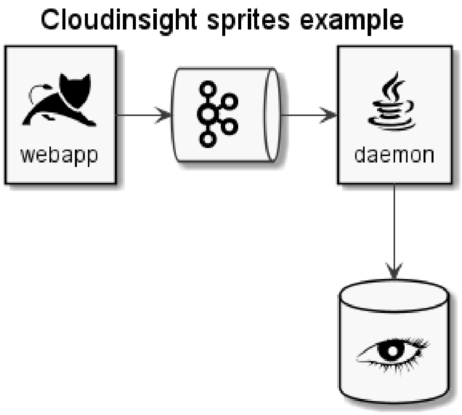


26.7 Cloud Insight (cloudinsight)

- https://github.com/plantuml/plantuml-stdlib/tree/master/cloudinsight
- https://github.com/rabelenda/cicon-plantuml-sprites

このリポジトリには、Cloudinsight のアイコンから生成した PlantUML スプライトがあります。PlantUML の図の中で簡単に使用することができ、一般的なテクノロジーの美しいビジュアル表現を行うことができます。

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

title Cloudinsight sprites example

skinparam monochrome true

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

webapp -> kafka
kafka -> daemon
daemon --> cassandra
@enduml
```

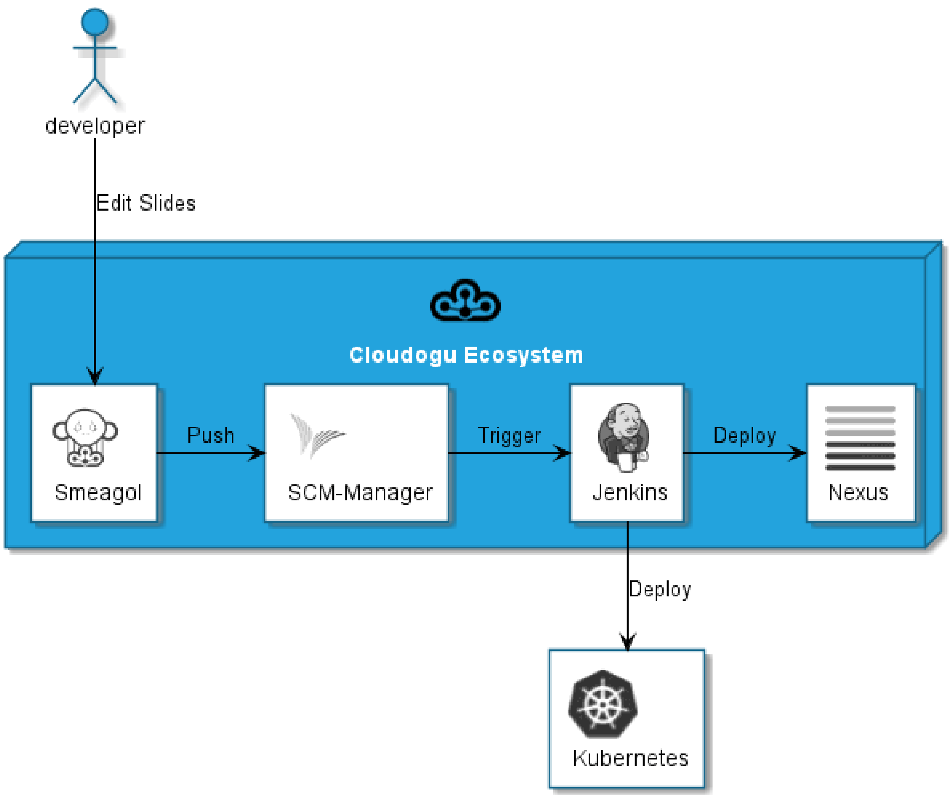


26.8 Cloudogu (cloudogu)

- https://github.com/plantuml/plantuml-stdlib/tree/master/cloudogu
- https://github.com/cloudogu/plantuml-cloudogu-sprites
- https://cloudogu.com

Cloudogu ライブラリは、PlantUML のスプライト、マクロ、その他の Cloudogu 向けサービスとリソースを提供します。

```
@startuml
!include <cloudogu/common.puml>
!include <cloudogu/dogus/jenkins.puml>
!include <cloudogu/dogus/cloudogu.puml>
!include <cloudogu/dogus/scm.puml>
!include <cloudogu/dogus/smeagol.puml>
!include <cloudogu/dogus/nexus.puml>
!include <cloudogu/tools/k8s.puml>

node "Cloudogu Ecosystem" <<$cloudogu>> {
DOGU_JENKINS(jenkins, Jenkins) #ffffff
DOGU_SCM(scm, SCM-Manager) #ffffff
DOGU_SMEAGOL(smeagol, Smeagol) #ffffff
DOGU_NEXUS(nexus,Nexus) #ffffff
}

TOOL_K8S(k8s, Kubernetes) #ffffff

actor developer

developer --> smeagol : "Edit Slides"
smeagol -> scm : Push
scm -> jenkins : Trigger
jenkins -> nexus : Deploy
jenkins --> k8s : Deploy
@enduml
```



すべての cloudogu スプライト

利用可能なすべての cloudogu スプライトは plantuml-cloudogu-sprites を確認してください。

26.9 Elastic ライブラリ (elastic)

- https://github.com/plantuml/plantuml-stdlib/tree/master/elastic
- https://github.com/Crashedmind/PlantUML-Elastic-icons

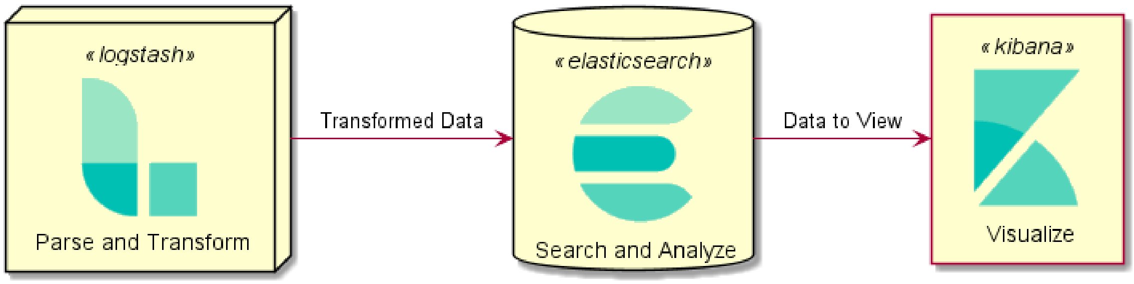
Elastic ライブラリには、Elastic のアイコンが含まれています。これは、AWS および Azure ライブラリと類似のものです (同じツールを使用して作られました)。

スプライトの含まれるファイルをインポートして使います。例: `!include elastic/elastic_search/elastic_search.puml` インポートされると、通常のスプライトと同様、`<$sprite_name>` のように使用することができます。

`!include <elastic/common>` のようにして `common.puml` をインクルードすると、そこに定義されたヘルパーマクロを使用することができます。`common.puml` をインポートすると、`NAME_OF_SPRITE(parameters...)` マクロを使用することができます。

使用例:

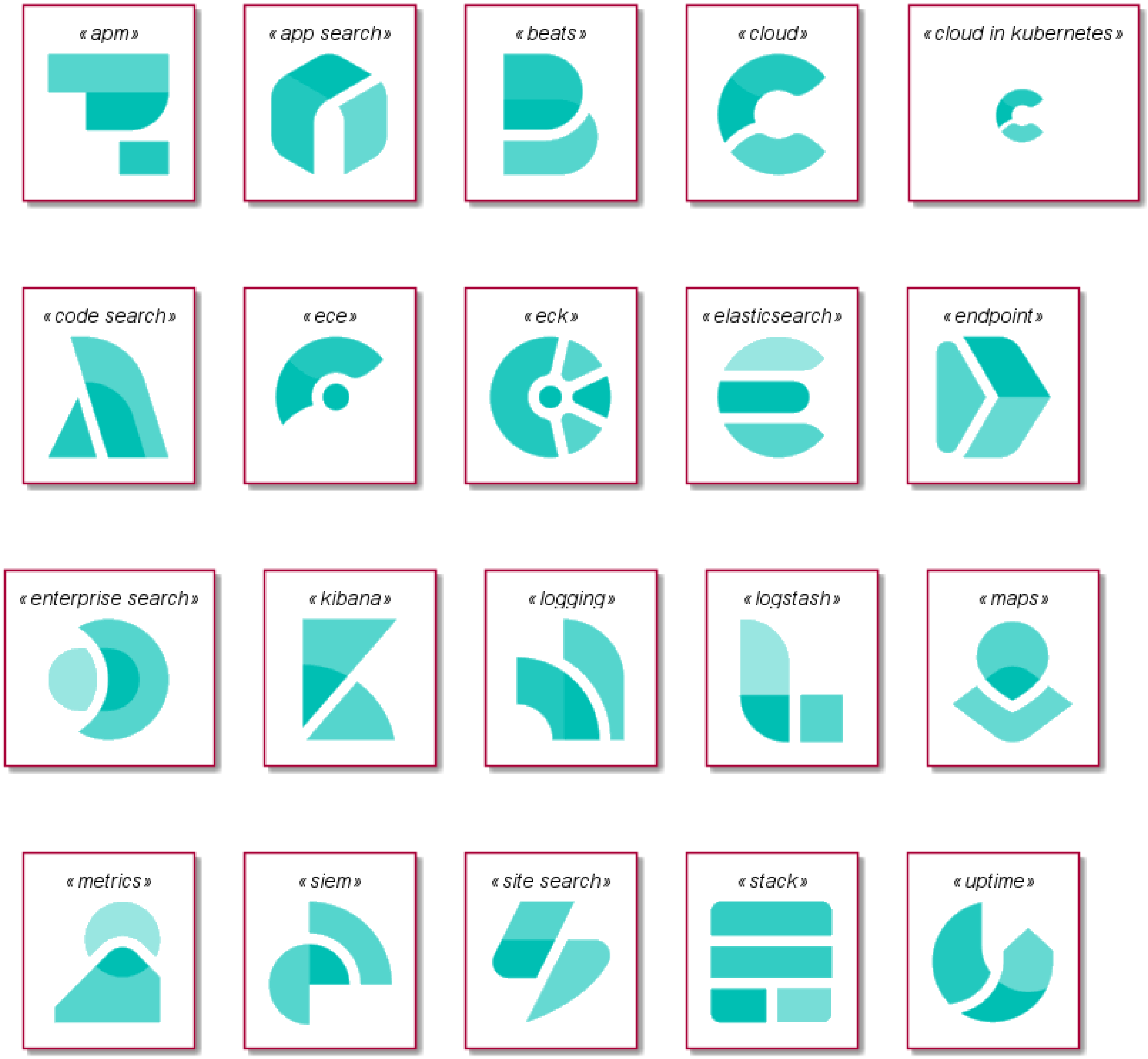
```
@startuml
!include <elastic/common>
!include <elastic/elasticsearch/elasticsearch>
!include <elastic/logstash/logstash>
!include <elastic/kibana/kibana>

ELASTICSEARCH(ElasticSearch, "Search and Analyze",database)
LOGSTASH(Logstash, "Parse and Transform",node)
KIBANA(Kibana, "Visualize",agent)

Logstash -right-> ElasticSearch: Transformed Data
ElasticSearch -right-> Kibana: Data to View
@enduml
```



すべての Elastic スプライト

```
@startuml
'Adapted from https://github.com/Crashedmind/PlantUML-Elastic-icons/blob/master/All.puml

'Elastic stuff here
'=================================

!include <elastic/common.puml>
!include <elastic/apm/apm.puml>
!include <elastic/app_search/app_search.puml>
!include <elastic/beats/beats.puml>
!include <elastic/cloud/cloud.puml>
!include <elastic/cloud_in_kubernetes/cloud_in_kubernetes.puml>
!include <elastic/code_search/code_search.puml>
!include <elastic/ece/ece.puml>
!include <elastic/eck/eck.puml>
' Beware of the difference between Crashedmind and plantuml-stdlib version: with '_' usage!
!include <elastic/elasticsearch/elasticsearch.puml>
!include <elastic/endpoint/endpoint.puml>
!include <elastic/enterprise_search/enterprise_search.puml>
!include <elastic/kibana/kibana.puml>
!include <elastic/logging/logging.puml>
!include <elastic/logstash/logstash.puml>
!include <elastic/maps/maps.puml>
!include <elastic/metrics/metrics.puml>
!include <elastic/siem/siem.puml>
!include <elastic/site_search/site_search.puml>
!include <elastic/stack/stack.puml>
!include <elastic/uptime/uptime.puml>

skinparam agentBackgroundColor White

APM(apm)
APP_SEARCH(app_search)
BEATS(beats)
CLOUD(cloud)
CLOUD_IN_KUBERNETES(cloud_in_kubernetes)
CODE_SEARCH(code_search)
ECE(ece)
ECK(eck)
ELASTICSEARCH(elastic_search)
ENDPOINT(endpoint)
ENTERPRISE_SEARCH(enterprise_search)
KIBANA(kibana)
LOGGING(logging)
LOGSTASH(logstash)
MAPS(maps)
METRICS(metrics)
```

```
SIEM(siem)
SITE_SEARCH(site_search)
STACK(stack)
UPTIME(uptime)
@enduml
```



26.10 Google Material Icons (material)

- https://github.com/plantuml/plantuml-stdlib/tree/master/material
- https://github.com/Templarian/MaterialDesign

このライブラリには、Google やその他の作成者によって作られた、フリーのマテリアルスタイルのアイコンが含まれています。

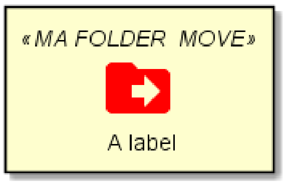
スプライトの含まれるファイルをインポートして使います。例: `!include <material/ma_folder_move>` インポートされると、通常のスプライトと同様、`<$ma_sprite_name>` のように使用することができます。このライブラリでは、他のライブラリの同じ名前のスプライトとの衝突を避けるために、スプライト名にプレフィックス `ma_` を付ける必要があることに注意してください。

`!include <material/common>` のようにして `common.puml` をインクルードすると、そこに定義された

ヘルパーマクロを使用することができます。common.puml をインポートすると、MA_NAME_OF_SPRITE(parameters...) マクロを使用することができます。こちらもプレフィックス MA_ を付ける必要があることに注意してください。

使用例:

```
@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
```

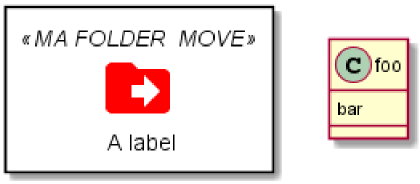


注意：

例えば次のように、クラスとともに rectangle を加えようとした場合のように、スプライトマクロをほかの要素と一緒に使う場合にシンタックスエラーが発生することがあります。そのような場合には、マクロの後ろに {と} を加えて空の rectangle を作るようにしてください。

使用例:

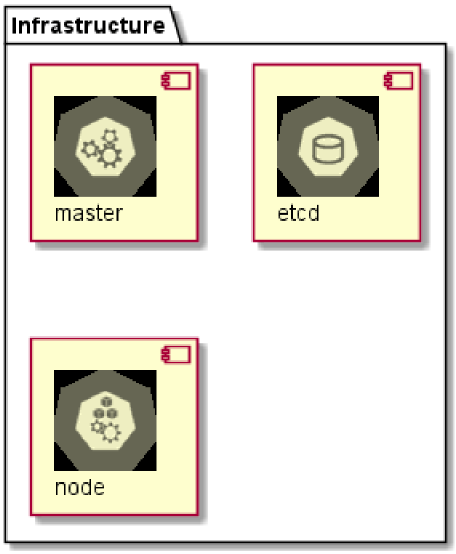
```
@startuml
!include <material/common>
' To import the sprite file you DON'T need to place a prefix!
!include <material/folder_move>

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

class foo {
    bar
}
@enduml
```



26.11 Kubernetes (kubernetes)

- https://github.com/plantuml/plantuml-stdlib/tree/master/kubernetes
- https://github.com/michiel/plantuml-kubernetes-sprites

```
@startuml
!include <kubernetes/k8s-sprites-unlabeled-25pct>
package "Infrastructure" {
  component "<$master>\nmaster" as master
  component "<$etcd>\netcd" as etcd
  component "<$node>\nnode" as node
}
@enduml
```

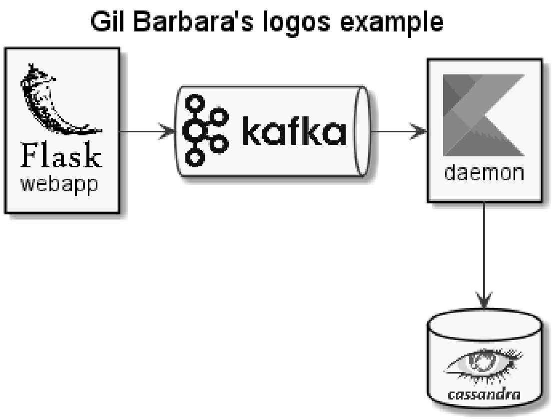


26.12 Logos (logos)

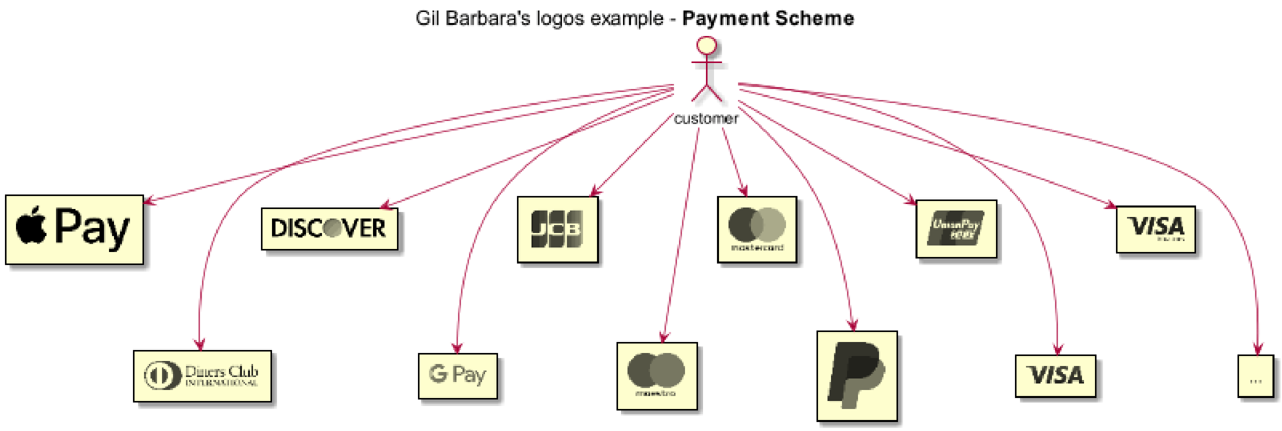
- https://github.com/plantuml/plantuml-stdlib/tree/master/logos
- https://github.com/plantuml-stdlib/gilbarbara-plantuml-sprites

このリポジトリには Gil Barbara's logos から生成された PlantUML のスプライトが含まれています。素晴らしい視覚資料を、PlantUML の図の中で簡単に使用することができます。

```
@startuml
!include <logos/flask.puml>
!include <logos/kafka.puml>
!include <logos/kotlin.puml>
!include <logos/cassandra.puml>

title Gil Barbara's logos example

skinparam monochrome true

rectangle "<$flask>\nwebapp" as webapp
queue "<$kafka>" as kafka
rectangle "<$kotlin>\ndaemon" as daemon
database "<$cassandra>" as cassandra

webapp -> kafka
kafka -> daemon
daemon --> cassandra
@enduml
```



```
@startuml
scale 0.7
!include <logos/apple-pay.puml>
!include <logos/dinersclub.puml>
!include <logos/discover.puml>
!include <logos/google-pay.puml>
!include <logos/jcb.puml>
!include <logos/maestro.puml>
!include <logos/mastercard.puml>
!include <logos/paypal.puml>
!include <logos/unionpay.puml>
!include <logos/visaelectron.puml>
!include <logos/visa.puml>
' ...

title Gil Barbara's logos example - **Payment Scheme**

actor customer
rectangle "<$apple-pay>"    as ap
rectangle "<$dinersclub>"   as dc
rectangle "<$discover>"     as d
rectangle "<$google-pay>"   as gp
rectangle "<$jcb>"          as j
rectangle "<$maestro>"      as ma
rectangle "<$mastercard>"   as m
rectangle "<$paypal>"       as p
rectangle "<$unionpay>"     as up
rectangle "<$visa>"         as v
rectangle "<$visaelectron>" as ve
rectangle "..." as etc

customer --> ap
customer ---> dc
customer --> d
customer ---> gp
customer --> j
customer ---> ma
customer --> m
customer ---> p
customer --> up
customer ---> v
customer --> ve
customer ---> etc
@enduml
```



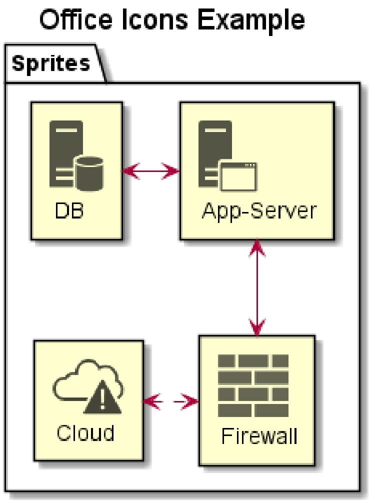
26.13 Office (office)

- https://github.com/plantuml/plantuml-stdlib/tree/master/office
- https://github.com/Roemer/plantuml-office

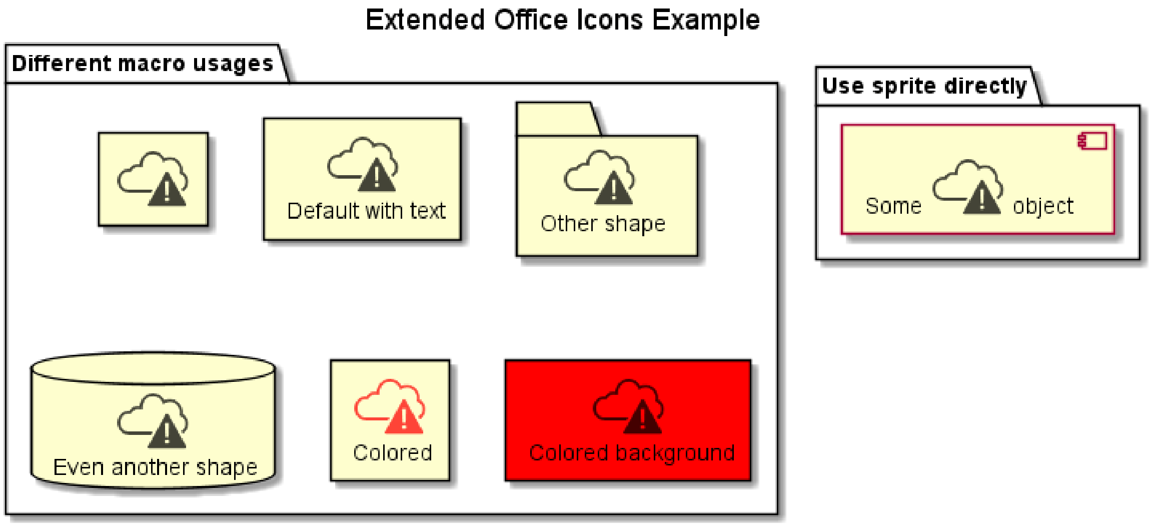
スプライト (*.puml) と色付きの PNG アイコンが利用可能です。名前に色名が含まれていたとしてもスプライトはすべてモノクロです (ファイルが自動生成されているためです)。マクロ (以下の例を参照) を使ってスプライトに色を付けるか、フルカラーの PNG を使用することができます。スプライト、PNG、マクロの使い方は以下の例を参照してください。

使用例:

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

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

title Office Icons Example

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




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

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

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

title Extended Office Icons Example

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

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



26.14 Open Security Architecture (OSA) [osa]

- https://github.com/plantuml/plantuml-stdlib/tree/master/osa
- https://github.com/Crashedmind/PlantUML-opensecurityarchitecture-icons
- https://www.opensecurityarchitecture.org

```
@startuml
'Adapted from https://github.com/Crashedmind/PlantUML-opensecurityarchitecture-icons/blob/master/all
scale .5
!include <osa/arrow/green/left/left.puml>
!include <osa/arrow/yellow/right/right.puml>
!include <osa/awareness/awareness.puml>
!include <osa/contract/contract.puml>
!include <osa/database/database.puml>
!include <osa/desktop/desktop.puml>
!include <osa/desktop/imac/imac.puml>
!include <osa/device_music/device_music.puml>
!include <osa/device_scanner/device_scanner.puml>
!include <osa/device_usb/device_usb.puml>
!include <osa/device_wireless_router/device_wireless_router.puml>
!include <osa/disposal/disposal.puml>
!include <osa/drive_optical/drive_optical.puml>
!include <osa/firewall/firewall.puml>
!include <osa/hub/hub.puml>
!include <osa/ics/drive/drive.puml>
!include <osa/ics/plc/plc.puml>
!include <osa/ics/thermometer/thermometer.puml>
!include <osa/id/card/card.puml>
!include <osa/laptop/laptop.puml>
!include <osa/lifecycle/lifecycle.puml>
!include <osa/lightning/lightning.puml>
!include <osa/media_flash/media_flash.puml>
!include <osa/media_optical/media_optical.puml>
!include <osa/media_tape/media_tape.puml>
!include <osa/mobile/pda/pda.puml>
!include <osa/padlock/padlock.puml>
!include <osa/printer/printer.puml>
!include <osa/site_branch/site_branch.puml>
!include <osa/site_factory/site_factory.puml>
!include <osa/vpn/vpn.puml>
```

```
!include <osa/wireless/network/network.puml>

rectangle "OSA" {
rectangle "Left: <$left>"
rectangle "Right: <$right>"
rectangle "Awareness: <$awareness>"
rectangle "Contract: <$contract>"
rectangle "Database: <$database>"
rectangle "Desktop: <$desktop>"
rectangle "Imac: <$imac>"
rectangle "Device_music: <$device_music>"
rectangle "Device_scanner: <$device_scanner>"
rectangle "Device_usb: <$device_usb>"
rectangle "Device_wireless_router: <$device_wireless_router>"
rectangle "Disposal: <$disposal>"
rectangle "Drive_optical: <$drive_optical>"
rectangle "Firewall: <$firewall>"
rectangle "Hub: <$hub>"
rectangle "Drive: <$drive>"
rectangle "Plc: <$plc>"
rectangle "Thermometer: <$thermometer>"
rectangle "Card: <$card>"
rectangle "Laptop: <$laptop>"
rectangle "Lifecycle: <$lifecycle>"
rectangle "Lightning: <$lightning>"
rectangle "Media_flash: <$media_flash>"
rectangle "Media_optical: <$media_optical>"
rectangle "Media_tape: <$media_tape>"
rectangle "Pda: <$pda>"
rectangle "Padlock: <$padlock>"
rectangle "Printer: <$printer>"
rectangle "Site_branch: <$site_branch>"
rectangle "Site_factory: <$site_factory>"
rectangle "Vpn: <$vpn>"
rectangle "Network: <$network>"
}
@enduml
```

26.15 Tupadr3 ライブラリ (tupadr3)

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

このライブラリには Devicons や Font Awesome を含む、いくつかのアイコンライブラリが含まれています。

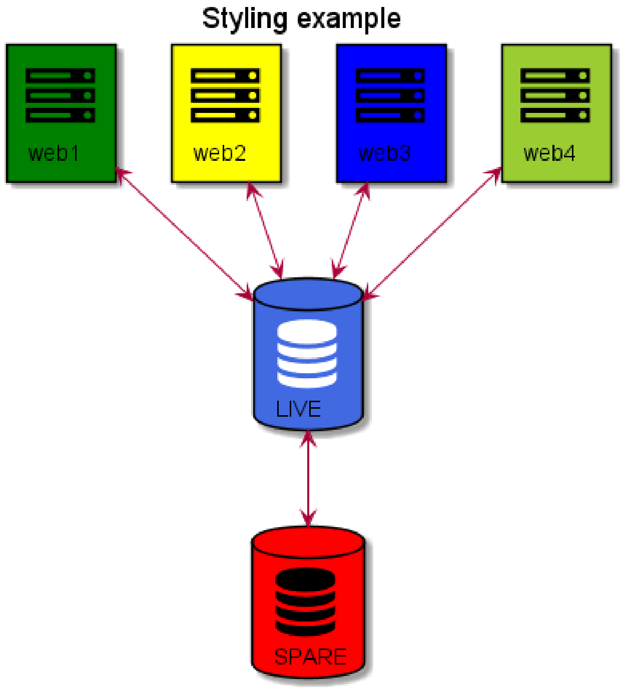
スプライトの含まれるファイルをインポートして使います。例: `!include <font-awesome/align_center>` インポートされると、通常のスプライトと同様、`<$sprite_name>` のように使用することができます。

`!include <font-awesome/common>` のようにして `common.puml` をインクルードすると、そこに定義されたヘルパーマクロを使用することができます。`common.puml` をインポートすると、`NAME_OF_SPRITE(parameters...)` マクロを使用することができます。

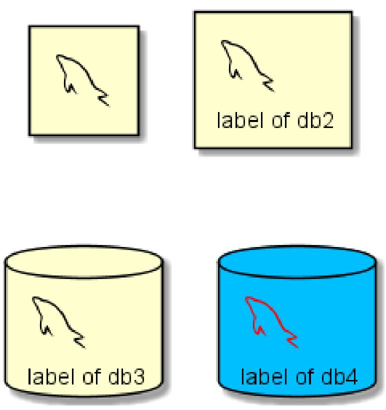
使用例:

```
@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
```



Contents