PlantUML is an Open Source project that allows to quickly write:

- Sequence diagram,
- Usecase diagram,
- Class diagram,
- Activity diagram,
- Component diagram,
- State diagram,
- Object diagram.

Diagrams are defined using a simple and intuitive language.
1 Sequence Diagram

1.1 Basic examples

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

To have a dotted arrow, you use -->

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

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

1.2 Declaring participant

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

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

- actor
- boundary
- control
- entity
- database

@startuml
actor Foo1
boundary Foo2
control Foo3
entity Foo4
database Foo5
collections Foo6
Foo1 -> Foo2: To boundary
Foo1 -> Foo3: To control
Foo1 -> Foo4: To entity
Foo1 -> Foo5: To database
Foo1 -> Foo6: To collections
@enduml
You can rename a participant using the `as` keyword.
You can also change the background color of actor or participant.

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

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

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

```plantuml
@startuml
participant Last order 30
participant Middle order 20
participant First order 10
@enduml
```
```
1.3 Use non-letters in participants

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

```plantuml
@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
```

![Diagram showing sequence diagram with participants and messages]

1.4 Message to Self

A participant can send a message to itself.

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

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

![Diagram showing message to self]

1.5 Change arrow style

You can change arrow style by several ways:

- add a final `x` to denote a lost message
- use `\` or `/` instead of `<` or `>` to have only the bottom or top part of the arrow
- repeat the arrow head (for example, `>>` or `//`) head to have a thin drawing
- use `--` instead of `-` to have a dotted arrow
- add a final "o" at arrow head
- use bidirectional arrow
1.6 Change arrow color

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

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

1.7 Message sequence numbering

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

```plantuml
@startuml
autonumber
Bob -> Alice : Authentication Request
Bob <-> Alice : Authentication Response
@enduml
```
You can specify a start number with `autonumber 'start'`, and also an increment with `autonumber 'start' 'increment'`.

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

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

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

You can specify a format for your number by using between double-quote. The formatting is done with the Java class `DecimalFormat` (’0’ means digit, ’#’ means digit and zero if absent).

You can use some html tag in the format.

```plantuml
@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 </b>"
Bob -> Alice : Yet another authentication Request
Bob <- Alice : Yet another authentication Response
@enduml
```
You can also use `autonumber stop` and `autonumber resume 'increment' 'format'` to respectively pause and resume automatic numbering.

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

autonumber stop
Bob -> Alice : dummy

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

autonumber stop
Bob -> Alice : dummy

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

1.8 Splitting diagrams

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

You can put a title for the new page just after the `newpage` keyword.

This is very handy with Word to print long diagram on several pages.
1.9 Grouping message

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

- alt/else
- opt
- loop
- par
- break
- critical
- group, followed by a text to be displayed
It is possible to add a text that will be displayed into the header (except for group). The `end` keyword is used to close the group. Note that it is possible to nest groups.

```plantuml
@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
```

1.10 Notes on messages

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

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

```plantuml
@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
@enduml
```
1.11 Some other notes

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

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

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

```plantuml
@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```
1.12 Changing notes shape

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

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

1.13 Creole and HTML

It is also possible to use creole formatting:

```plantuml
@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__
@enduml
```
1.14 Divider

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

@startuml

== Initialization ==

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

== Repetition ==

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

@enduml
1.15 Reference

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

```
@startuml
participanet Alice
actor Bob

ref over Alice, Bob : init
Alice --> Bob : hello
ref over Bob
This can be on
several lines
end ref
@enduml
```

1.16 Delay

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

```
@startuml
Alice --> Bob: Authentication Request
...
Bob --> Alice: Authentication Response
```
1.17 Space

You can use ||| to indicate some spacing in the diagram.
It is also possible to specify a number of pixel to be used.

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

1.18 Lifeline Activation and Destruction

The activate and deactivate are used to denote participant activation.
Once a participant is activated, its lifeline appears.
The activate and deactivate apply on the previous message.
The destroy denote the end of the lifeline of a participant.
1.18 Lifeline Activation and Destruction

@startuml
participant User

User -> A: DoWork
activate A

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

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

B -->> A: RequestCreated
deactivate B

A -> User: Done
deactivate A
@enduml

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

@startuml
participant User

User -> A: DoWork
activate A #FFBBBB

A -> A: Internal call
activate A #DarkSalmon

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

B -->> A: RequestCreated
deactivate B
deactivate A

A -> User: Done
deactivate A
@enduml
1.19 Participant creation

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

```
@startuml
Bob -> Alice : hello
create Other
Alice -> Other : new
create control String
Alice -> String
note right : You can also put notes!
Alice --> Bob : ok
@enduml
```

1.20 Incoming and outgoing messages

You can use incoming or outgoing arrows if you want to focus on a part of the diagram.

Use square brackets to denote the left "[" or the right "]" side of the diagram.

```
@startuml
[-> A: DoWork
activate A
A -> A: Internal call
activate A
A ->] : << createRequest >>
@enduml
```
You can also have the following syntax:

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

[<- Bob
[x<- Bob

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

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

1.21 Stereotypes and Spots

It is possible to add stereotypes to participants using << and >>.

In the stereotype, you can add a spotted character in a colored circle using the syntax \((X, \text{color})\).
1.22 More information on titles

You can use creole formatting in the title.

```plantuml
@startuml
title __Simple__ **communication** example
Alice -> Bob: Authentication Request
Bob -> Alice: Authentication Response
@enduml```

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

```plantuml
@startuml
skinparam guillemet false
participant "Famous Bob" as Bob << Generated >>
participant Alice << (C,#ADD1B2) Testable >>
Bob->Alice: First message
@enduml```

```plantuml
@startuml
participant Bob << (C,#ADD1B2) >>
participant Alice << (C,#ADD1B2) >>
Bob->Alice: First message
@enduml```
You can add newline using \n in the title description.

```plantuml
@startuml
title __Simple__ communication example
on several lines
Alice -> Bob: Authentication Request
Bob -> Alice: Authentication Response
@enduml
```

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

```plantuml
@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
```
1.23 Participants encompass

It is possible to draw a box around some participants, using box and end box commands. You can add an optional title or a optional background color, after the box keyword.

```plantuml
@startuml
box "Internal Service" #LightBlue
participant Bob
participant Alice
end box
participant Other
Bob -> Alice : hello
Alice -> Other : hello
@enduml
```

1.24 Removing Footer

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

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

1.25 Skinparam

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

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

You can also change other rendering parameter, as seen in the following examples:
@startuml
skinparam sequenceArrowThickness 2
skinparam roundcorner 20
skinparam maxmessagesize 60
skinparam sequenceParticipant underline

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

User -> A: DoWork
activate A

A -> B: Create Request
activate B

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

B --> A: Request Created
deactivate B

A --> User: Done
deactivate A
@enduml

@startuml
skinparam backgroundColor #EEEBDC
skinparam handwritten true

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

ParticipantBorderColor DeepSkyBlue
ParticipantBackgroundColor DodgerBlue
ParticipantFontName Impact
ParticipantFontSize 17
@enduml
1.26 Changing padding

It is possible to tune some padding settings.

```plantuml
@startuml
skinparam ParticipantPadding 20
skinparam BoxPadding 10
box "Foo1"
participant Alice1
participant Alice2
end box
box "Foo2"
participant Bob1
participant Bob2
@enduml
```
end box
Alice1 -> Bob1 : hello
Alice1 -> Out : out
@enduml
2 Use Case Diagram

2.1 Usecases

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

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

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

2.2 Actors

Actor are enclosed using between two points.

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

We will see later that the actor definitions are optional.

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

2.3 Usecases description

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

You can also use the following separators: `\-.\-\-` `\-\-\-\-`. And you can put titles within the separators.
2.4 Basic example

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

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

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

```
@startuml
User -> (Start)
User --> (Use the application) : A small label
:Main Admin: ---> (Use the application) : This is\nyet another\nlabel
@enduml
```

2.5 Extension

If one actor/use case extends another one, you can use the symbol <|-- (which stands for ).
2.6 Using notes

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

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

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

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

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

note right of Admin : This is an example.

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

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

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

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

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

2.8 Changing arrows direction

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

```plantuml
@startuml
:user: --> (Use case 1) :user: -> (Use case 2)
@enduml
```
2.8 Changing arrows direction

You can also change directions by reversing the link:

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

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

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

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

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

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

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

2.10 Left to right direction

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

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

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

```plantuml
@startuml
left to right direction
user1 --> (Usecase 1)
user2 --> (Usecase 2)
@enduml
```
2.11 Skinparam

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

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

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

```plantuml
@startuml
skinparam handwritten true

skinparam usecase {
  BackgroundColor DarkSeaGreen
  BorderColor DarkSlateGray

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

  ArrowColor Olive
  ActorBorderColor black
  ActorFontName Courier

  ActorBackgroundColor<< Human >> Gold
}

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

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

MySql --> (Use)
@enduml
```
2.12 Complete example

```plantuml
@startuml
left to right direction
skinparam packageStyle rectangle
actor customer
actor clerk
rectangle checkout {
    customer -- (checkout)
    (checkout) .> (payment) : include
    (help) .> (checkout) : extends
    (checkout) -- clerk
}
@enduml
```
### 3 Class Diagram

#### 3.1 Relations between classes

Relations between classes are defined using the following symbols:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="./extension.png" alt="Extension" /></td>
<td>Extension</td>
</tr>
<tr>
<td><img src="./composition.png" alt="Composition" /></td>
<td>Composition</td>
</tr>
<tr>
<td><img src="./aggregation.png" alt="Aggregation" /></td>
<td>Aggregation</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

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

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

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

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

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

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

```plantuml
@startuml
Object <|-- ArrayList
Object : equals()
ArrayList : Object[] elementData
ArrayList : size()
@enduml
```

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

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

```plantuml
@startuml
class Dummy {
    String data
    void methods()
}
class Flight {
    flightNumber : Integer
    departureTime : Date
}
@enduml
```

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

```plantuml
@startuml
class Dummy {
    {field} A field (despite parentheses)
    {method} Some method
}
@enduml
```
3.4 Defining visibility

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

- private
- protected
- package private
+ public

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

You can turn off this feature using the `skinparam classAttributeIconSize 0` command:

@startuml
skinparam classAttributeIconSize 0
class Dummy {
- field1
# field2
~ method1()
+ method2()
}
@enduml
3.5 Abstract and Static

You can define static or abstract methods or fields using the \{static\} or \{abstract\} modifier.

These modifiers can be used at the start or at the end of the line. You can also use \{classifier\} instead of \{static\}.

```plantuml
@startuml
class Dummy {
    \{static\} String id
    \{abstract\} void methods()
}
@enduml
```
3.6 Advanced class body

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

You can also use titles within the separators:

```plantuml
@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
```

![Diagram of Foo and User classes](image)

---

3.7 Notes and stereotypes

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

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

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

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

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

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

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

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

![Diagram showing class relationships with notes and stereotypes]
3.8 More on notes

It is also possible to use few html tags like :

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

You can also have a note on several lines.

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

```plantuml
@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></note> <b><color:royalBlue>on several</color> words</s> lines
And this is hosted by <img:sourceforge.jpg>
end note
@enduml
```
3.9 Note on links

It is possible to add a note on a link, just after the link definition, using `note on link`.
You can also use `note left on link`, `note right on link`, `note top on link`, and `note bottom on link` if you want to change the relative position of the note with the label.

```plantuml
@startuml
class Dummy

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

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

@enduml
```
3.10 Abstract class and interface

You can declare a class as abstract using "abstract" or "abstract class" keywords. The class will be printed in italic.

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

```plantuml
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
```

![Class Diagram](image-url)
3.11 Using non-letters

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

- Use the `as` keyword in the class definition
- Put quotes `"` around the class name

```plantuml
@startuml
class "This is my class" as class1
class class2 as "It works this way too"
class2 *-- "foo/dummy" : use
@enduml
```
3.12 Hide attributes, methods...

You can parameterize the display of classes using the `hide/show` command. The basic command is: `hide empty members`. This command will hide attributes or methods if they are empty.

Instead of `empty members`, you can use:

- `empty fields` or `empty attributes` for empty fields,
- `empty methods` for empty methods,
- `fields` or `attributes` which will hide fields, even if they are described,
- `methods` which will hide methods, even if they are described,
- `members` which will hide fields and methods, even if they are described,
- `circle` for the circled character in front of class name,
- ` stereotype` for the stereotype.

You can also provide, just after the `hide` or `show` keyword:

- `class` for all classes,
- `interface` for all interfaces,
- `enum` for all enums,
- `<<foo1>>` for classes which are stereotyped with `foo1`,
- an existing class name.

You can use several `show/hide` commands to define rules and exceptions.

```
@startuml
class Dummy1 {
  +myMethods()
}
class Dummy2 {
  +hiddenMethod()
}
class Dummy3 <<Serializable>> {
  String name
}
hide members
hide <<Serializable>> circle
show Dummy1 methods
show <<Serializable>> fields
@enduml
```
3.13 Hide classes

You can also use the `show/hide` commands to hide classes. This may be useful if you define a large included file, and if you want to hide some classes after file inclusion.

```
@startuml
   class Foo1
   class Foo2
   Foo2 *-- Foo1
   hide Foo2
@enduml
```

3.14 Use generics

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

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

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

3.15 Specific Spot

Usually, a spotted character (C, I, E or A) is used for classes, interface, enum and abstract classes. But you can define your own spot for a class when you define the stereotype, adding a single character and a color, like in this example:

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

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

Note that package definitions can be nested.

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

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

3.17 Packages style

There are different styles available for packages.

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

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

dataPackage foo2 <<Rectangle>> {
class Class2
}

dataPackage foo3 <<Folder>> {
class Class3
}

dataPackage foo4 <<Frame>> {
class Class4
}

dataPackage foo5 <<Cloud>> {
class Class5
}

dataPackage foo6 <<Database>> {
class Class6
}
@enduml
```
You can also define links between packages, like in the following example:

```plantuml
@startuml
skinparam packageStyle rectangle

package foo1.foo2 {
}

package foo1.foo2.foo3 {
class Object
}

foo1.foo2 +-- foo1.foo2.foo3
@enduml
```

### 3.18 Namespaces

In packages, the name of a class is the unique identifier of this class. It means that you cannot have two classes with the very same name in different packages.

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

You can refer to classes from other namespaces by fully qualify them. Classes from the default namespace are qualified with a starting dot.

Note that you don’t have to explicitly create namespace: a fully qualified class is automatically put in the right namespace.

```plantuml
@startuml
class BaseClass

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

namespace net.foo {
 net.dummy.Person <|-- Person
 .BaseClass <|-- Person
@enduml
3.19 Automatic namespace creation

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

```plantuml
@startuml
set namespaceSeparator ::
class X1::X2::foo {
    some info
}
@enduml
```

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

```plantuml
@startuml
set namespaceSeparator none
class X1.X2.foo {
    some info
}
@enduml
```
3.20 Lollipop interface

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

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

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

3.21 Changing arrows direction

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

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

You can also change directions by reversing the link:

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

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

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

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

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

You can define it in another direction:

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

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

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

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

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

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

```plantuml
@startuml
skinparam class {
BackgroundColor PaleGreen
ArrowColor SeaGreen
BorderColor SpringGreen
}
skinparam stereotypeCBackgroundColor YellowGreen
Class01 "1" *-- "many" Class02 : contains
Class03 o-- Class04 : aggregation
@enduml
```
3.24 Skinned Stereotypes

You can define specific color and fonts for stereotyped classes.

```plantuml
@startuml

skinparam class {
BackgroundColor PaleGreen
ArrowColor SeaGreen
BorderColor SpringGreen
BackgroundColor<<Foo>> Wheat
BorderColor<<Foo>> Tomato
}

skinparam stereotypeCBackgroundColor YellowGreen
skinparam stereotypeCBackgroundColor<< Foo >> DimGray

Class01 <<Foo>>
Class03 <<Foo>>
Class01 "1" -- "many" Class02 : contains
Class03 o-- Class04 : aggregation

@enduml
```

3.25 Color gradient

It’s possible to declare individual color for classes or note using the notation. You can use either standard color name or RGB code.

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

- \, 
- /,
- \,
- or -

depending the direction of the gradient.

For example, you could have:
3.26  Help on layout

Sometimes, the default layout is not perfect...
You can use together keyword to group some classes together : the layout engine will try to group them (as if they were in the same package).
You can also use hidden links to force the layout.

```
@startuml
class Bar1
class Bar2
together {
class Together1
class Together2
class Together3
}
Together1 - Together2
Together2 - Together3
Together1 -[hidden]--> Bar1
Bar1 -[hidden]--> Bar2
@enduml
```
3.27 Splitting large files

Sometimes, you will get some very large image files. You can use the "page (hpages)x(vpages)" command to split the generated image into several files:

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

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

```plantuml
@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
```
3.27 Splitting large files
4 Activity Diagram

4.1 Simple Activity

You can use (*) for the starting point and ending point of the activity diagram.

In some occasions, you may want to use (*top) to force the starting point to be at the top of the diagram.

Use --> for arrows.

@startuml

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

@enduml

4.2 Label on arrows

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

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

@startuml

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

@enduml

4.3 Changing arrow direction

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

- -down-→ (default arrow)
4.4 Branches

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

```plantuml
@startuml
(•) -up-> "Initialization"
if "Some Test" then
  -->[true] "Some Activity"
  --> "Another activity"
else
  -->[false] "Something else"
  -->[Ending process] (•)
endif
@enduml
```

Unfortunately, you will have to sometimes repeat the same activity in the diagram text:
4.5 More on Branches

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

It is also possible to nest branches.

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

@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"
@enduml
4.6 Synchronization

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

```plantuml
@startuml
(*)->===B1===
"Parallel Activity 1"
===B1===->"Parallel Activity 2"
===B2===
(*)
@enduml
```
4.7 Long activity description

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

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

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

-A1 -up-> "Another activity\n on several lines"

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

4.8 Notes

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

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

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

```plantuml
@startuml

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

You can define a partition using the `partition` keyword, and optionally declare a background color for your partition (Using a html color code or name). When you declare activities, they are automatically put in the last used partition. You can close the partition definition using a closing bracket `}`.

```plantuml
@startuml

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

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

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

partition Orchestra #CCCEEE {
  WavesArmes --> Introduction
  --> "Play music"
}
@enduml
```
4.10 Skinparam

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

You can use this command:

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

You can define specific color and fonts for stereotyped activities.

```plantuml
@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
```

```bash
PlantUML Language Reference Guide (Version 1.2018.2) 60 of 121
```
4.11 Octagon

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

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

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

4.12 Complete example

```
@startuml
title Servlet Container

(♦) --> "ClickServlet.handleRequest()"
--> "new Page"

if "Page.onSecurityCheck" then
  ->[true] "Page.onInit()"
@enduml
```
if "isForward?" then
  ->[no] "Process controls"

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

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

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

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

if "DoRedirect?" then
  ->[yes] "redirect request"
  --==BEFORE_DESTROY===
else
  if "DoForward?" 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.onInit()

IsForward?

Process controls

IsPost?

Page.onPost()

Page.onGet()

Page.onRender()

Do redirect?

Redirect request

Forward request

Page.onDestroy()
5 Activity Diagram (beta)

Current syntax for activity diagram has several limitations and drawbacks (for example, it’s difficult to maintain).
So a completely new syntax and implementation is proposed as beta version to users (starting with V7947), so that we could define a better format and syntax.
Another advantage of this new implementation is that it’s done without the need of having Graphviz installed (as for sequence diagrams).
The new syntax will replace the old one. However, for compatibility reason, the old syntax will still be recognized, to ensure ascending compatibility.
Users are simply encouraged to migrate to the new syntax.

5.1 Simple Activity

Activities label starts with : and ends with ;.
Text formatting can be done using creole wiki syntax.
They are implicitly linked in their definition order.

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

5.2 Start/Stop

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

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

You can also use the end keyword.
5.3 Conditional

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

```plantuml
@startuml
start
if (Graphviz installed?) then (yes)
:process all\ndiagrams;
else (no)
:process only
__sequence__ and __activity__ diagrams;
endif
stop
@enduml
```

You can use the elseif keyword to have several tests:

```plantuml
@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)
@enduml
```
5.4 Repeat loop

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

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

5.5 While loop

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

``` PlantUML
@startuml
start
read data
generate diagrams
more data?
@enduml
```
while (data available?)
:read data;
:generate diagrams;
endwhile

stop
@enduml

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

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

5.6 Parallel processing

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

@startuml
start
if (multiprocessor?) then (yes)
fork
:Treatment 1;
fork again
:Treatment 2;
end fork
else (monoproc)
:Treatment 1;
:Treatment 2;
endif
@enduml
5.7 Notes

Text formatting can be done using creole wiki syntax. A note can be floating, using `floating` keyword.

```plantuml
@startuml
start
:foo1;
floating note left: This is a note
:foo2;
note right
This note is on several //lines// and can contain <b>HTML</b>
====
* Calling the method "foo()" is prohibited
end note
stop
@enduml
```

5.8 Colors

You can use specify a color for some activities.

```plantuml
@startuml
start
:starting progress;
#HotPink:reading configuration files
These files should edited at this point!;
#AAAAAA:ending of the process;
@enduml
```
5.9 Arrows

Using the -> notation, you can add texts to arrow, and change their color. It’s also possible to have dotted, dashed, bold or hidden arrows.

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

5.10 Connector

You can use parentheses to denote connector.

```plantuml
@startuml
start
Some activity;
(A)
@enduml
```
5.11 Grouping

You can group activity together by defining partition:

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

5.12 Swimlanes

Using pipe |, you can define swimlanes.
It’s also possible to change swimlanes color.

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

5.13 Detach

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

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

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

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

@startuml
:Ready;
:next(o) |
:Receiving:
split
:nak(i)<
:ack(o>)
split again
:ack(i)<
:next(o)
on several line|
 :i := i + 1]
:ack(o>)
split again
:err(i)<
:nak(o)>
split again
:foo/
split again
 :i > 5}
stop
end split
:finish;
@enduml
5.15 Complete example

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

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

:if (isPost?) then (yes)
:Process controls;
:else
:if (do forward?) then (yes)
:Forward request;
:else (no)
:Render page template;
endif
endif

stop
@enduml
5.15 Complete example

6 Component Diagram

6.1 Components

Components must be bracketed.

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

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

6.2 Interfaces

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

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

We will see latter that interface definition is optional.

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

6.3 Basic example

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

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

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

```plantuml
@startuml
interface "Data Access" as DA
DA - [First Component]  
[First Component] ..> HTTP : use

note left of HTTP : Web Service only
note right of [First Component]
A note can also be on several lines
end note
@enduml
```

6.5 Grouping Components

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

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

@startuml

package "Some Group" {
HTTP - [First Component]
[Another Component]
}
	node "Other Groups" {
FTP - [Second Component]
[First Component] --> FTP
}

cloud {
[Example 1]
}

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

[Another Component] --> [Example 1]
[Example 1] --> [Folder 3]
[Folder 3] --> [Frame 4]
@enduml
By default, links between classes have two dashes -- and are vertically oriented. It is possible to use horizontal link by putting a single dash (or dot) like this:

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

You can also change directions by reversing the link:

```
@startuml
Interface1 <-- [Component]
Interface2 <- [Component]
@enduml
```
It is also possible to change arrow direction by adding left, right, up or down keywords inside the arrow:

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

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

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

### 6.7 Use UML2 notation

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

@startuml

interface "Data Access" as DA

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

@enduml
6.8 Long description

It is possible to put description on several lines using square brackets.

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

6.9 Individual colors

You can specify a color after component definition.

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

6.10 Using Sprite in Stereotype

You can use sprites within stereotype components.

```plantuml
sprite $businessProcess [16x16/16] {
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
}
```

6.11 Skinparam

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

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

You can define specific color and fonts for stereotyped components and interfaces.

```
@startuml

skinparam interface {
  backgroundColor RosyBrown
  borderColor orange
}

skinparam component {
  fontSize 13
  backgroundColor<<Apache>> Red
  borderColor<<Apache>> #FF6655
  fontName Courier
  borderColor black
  backgroundColor gold
  arrowFontName Impact
  arrowColor #FF6655
  arrowFontColor #777777
}

() "Data Access" as DA

DA - [First Component]
[First Component] ..> () HTTP : use
HTTP - [Web Server] << Apache >>
@enduml
```
@startuml
[AA] <<static lib>>
[BB] <<shared lib>>
[CC] <<static lib>>
node node1
node node2 <<shared node>>
database Production

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

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

skinparam databaseBackgroundColor Aqua
@enduml
7 State Diagram

7.1 Simple State

You can use [*] for the starting point and ending point of the state diagram.
Use --> for arrows.

```plantuml
@startuml
[*] --> State1
State1 --> [*]
State1 : this is a string
State1 : this is another string
State1 --> State2
State2 --> [*]
@enduml```

7.2 Composite state

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

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

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

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

state NewValuePreview {
State1 --> State2
}
@enduml```
7.3 Long name

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

```plantuml
@startuml
scale 600 width
[*] -> State1
State1 --> State2 : Succeeded
State1 --> [*] : Aborted
State2 --> State3 : Succeeded
State2 --> [*] : Aborted
state State3 {
state "Accumulate Enough Data\nLong State Name" as long1
long1 : Just a test
[*] --> long1
long1 --> long1 : New Data
long1 --> ProcessData : Enough Data
}
State3 --> State3 : Failed
State3 --> [*] : Succeeded / Save Result
State3 --> [*] : Aborted
@enduml
```
7.4 Concurrent state

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

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

state Active {
[*] -> NumLockOff
NumLockOff --> NumLockOn : EvNumLockPressed
NumLockOn --> NumLockOff : EvNumLockPressed

[*] -> CapsLockOff
CapsLockOff --> CapsLockOn : EvCapsLockPressed
CapsLockOn --> CapsLockOff : EvCapsLockPressed

[*] -> ScrollLockOff
ScrollLockOff --> ScrollLockOn : EvCapsLockPressed
ScrollLockOn --> ScrollLockOff : EvCapsLockPressed
}
@enduml
```
7.5 Arrow direction

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

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

@startuml
[*] -up-> First
First -right-> Second
Second ---> Third
Third -left-> Last
@enduml
You can shorten the arrow by using only the first character of the direction (for example, -d- instead of -down-) or the two first characters (-do-).

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

### 7.6 Note

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

You can also define notes on several lines.

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

note left of Active : this is a short

note right of Inactive
A note can also
be defined on
several lines
end note
@enduml
```

You can also have floating notes.

```plantuml
@startuml
state foo
note "This is a floating note" as N1
@enduml
```
7.7 More in notes

You can put notes on composite states.

@startuml
[*] --> NotShooting

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

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

7.8 Skinparam

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

You can use this command:

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

You can define specific color and fonts for stereotyped states.

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

[*] --> NotShooting

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

NotShooting --> [*]
@enduml
8 Object Diagram

8.1 Definition of objects

You define instance of objects using the object keywords.

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

8.2 Relations between objects

Relations between objects are defined using the following symbols:

- Extension `<|--`
- Composition `*--`
- Aggregation `o--`

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

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

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

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

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

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

8.3 Adding fields

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

```
@startuml
object user

user : name = "Dummy"
user : id = 123
@enduml
```
It is also possible to ground between brackets \{\} all fields.

@startuml
object user {
    name = "Dummy"
    id = 123
}
@enduml

8.4 Common features with class diagrams

- Visibility
- Defines notes
- Use packages
- Skin the output
9 Common commands

9.1 Footer and header

You can use the commands header or footer to add a footer or a header on any generated diagram. You can optionally specify if you want a center, left or right footer/header, by adding a keyword. As for title, it is possible to define a header or a footer on several lines. It is also possible to put some HTML into the header or footer.

```plantuml
@startuml
Alice -> Bob: Authentication Request
header
<font color=red>Warning:</font> Do not use in production.
endheader
center footer Generated for demonstration
@enduml
```

9.2 Zoom

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

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

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

Salt is a subproject included in PlantUML that may help you to design graphical interface. You can use either `@startsalt` keyword, or `@startuml` followed by a line with `salt` keyword.

10.1 Basic widgets

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

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

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

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

10.2 Using grid

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

For example:

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

```text
Login  MyName
Password ****
[Cancel] [ OK ]
```
10.3 Using separator

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

# To display all vertical and horizontal lines

! To display all vertical lines

- To display all horizontal lines

+ To display external lines

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

### 10.3 Using separator

You can use several horizontal lines as separator.

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

### 10.4 Tree widget

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

```plantuml
@startsalt
{
{T
+ World
++ America
+++ Canada
+++ USA
++++ New York
++++ Boston
++++ Mexico
++ Europe
+++ Italy
+++ Germany
```

```
10.5 Enclosing brackets

You can define subelements by opening a new opening bracket.

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

```
Name
Modifiers:  ○ public ○ default ○ private ○ protected
          □ abstract □ final □ static
Superclass: java.lang.Object

Browse...
```

10.6 Adding tabs

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

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

```
General | Fullscreen | Behavior | Saving
Open image in: Smart Mode
[X] Smooth images when zoomed
[X] Confirm image deletion
[ ] Show hidden images
Close
```

Tab could also be vertically oriented:
10.7 Using menu

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

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

It is also possible to open a menu:

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

You can use two special notations for tables:

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

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

<table>
<thead>
<tr>
<th>Row header 1</th>
<th>value 1</th>
<th>value 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row header 2</td>
<td>A long cell</td>
<td></td>
</tr>
</tbody>
</table>
11 Creole

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

All diagrams are now supporting this syntax.

Note that ascending compatibility with HTML syntax is preserved.

11.1 Emphasized text

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

11.2 List

```plantuml
@startuml
object demo {
* Bullet list
* Second item
** Sub item
}
legend
# Numbered list
# Second item
## Sub item
## Another sub item
# Third item
end legend
@enduml
```
11.3 Escape character

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

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

11.4 Horizontal lines

```plantuml
@startuml
database DB1 as "You can have horizontal line
    ----
    Or double line
    ----- 
    Or strong line
    ----- 
    Or dotted line 
    . . My title ..
    Enjoy!"

    note right
    This is working also in notes
    You can also add title in all these lines
    ==Title==
    --Another title--
    end note
@enduml
```

11.5 Headings

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

11.6 Legacy HTML

Some HTML tags are also working:

- `<b>` for bold text
- `<u>` or `<u:#AAAAAA>` or `<u:colorName>` for underline
- `<i>` for italic
- `<s>` or `<s:#AAAAAA>` or `<s:colorName>` for strike text
11.7 Table

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

11.8 Tree

You can use _ characters to build a tree.

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

11.9 Special characters

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

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

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

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

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

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

```plantuml
@startuml
  listopeniconic
@enduml
```
11.11 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 $foo1 {
    FFFFFFFF
    FF0123456789ABCF
    FF0123456789ABCF
    FF0123456789ABCF
    FF0123456789ABCF
    FF0123456789ABCF
    FF0123456789ABCF
    FF0123456789ABCF
    FFFFFFFF
}
Alice -> Bob : Testing <$foo1>
@enduml
```

11.12 Encoding Sprite

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

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

where foo.png if 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.

11.13 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.

11.14 Examples

```
@startuml
sprite $printer [15x15/8z] NOth3WOW208HzFz_kMAhj71IWpa1XC716sz0Pq4MVPEWfBH1uxP3L6kbTcizR8tAhzaqFvXwvFfPEEq
start :click on <$printer> to print the page;
@enduml
```
@startuml
sprite $bug [15x15/16z] PKzR2i0m2BFMi15p_._FEjQEgBi27aeqCqixa8S40T7C53cKpsHpaYPDJY_12MHH=BLRyyPhbr1w3qu
sprite $printer [15x15/8z] NOtH3W0W208HxFz_kMAhj71HMpa1XC7i6sz0Pq4MVPEWfBHluzPSL6kbTcizB8tAhzaqFvXvxFfPEq
sprite $disk {
444445666677881
436000000009991
43600000000ACA1
53700000001A7A1
53700000012B8A1
53800000123B8A1
63800001233C9A1
634999AABBC99B1
744566778899AB1
7456AAA999AAB1
8566AFC228AABB1
8567AC8118BBBB1
867BD4433BBB1
39AAAAABBBBBBC1
}
title Use of sprites (<$printer>, <$bug>...)

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

note left : The printer <$printer> is available
@enduml
12 Changing fonts and colors

12.1 Usage

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

```
skinparam backgroundColor yellow
```

You can use this command:

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

12.2 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
}
```
### 12.3 Color

You can use either standard color name or RGB code.

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Default Value</th>
<th>Color</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>backgroundColor</td>
<td>white</td>
<td>#A80036</td>
<td>Background of the page</td>
</tr>
<tr>
<td>activityArrowColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Color of arrows in activity diagrams</td>
</tr>
<tr>
<td>activityBackgroundColor</td>
<td>#FEFECE</td>
<td>#FEFECE</td>
<td>Background of activities</td>
</tr>
<tr>
<td>activityBorderColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Color of activity borders</td>
</tr>
<tr>
<td>activityStartColor</td>
<td>black</td>
<td>black</td>
<td>Starting circle in activity diagrams</td>
</tr>
<tr>
<td>activityEndColor</td>
<td>black</td>
<td>black</td>
<td>Ending circle in activity diagrams</td>
</tr>
<tr>
<td>activityBarColor</td>
<td>black</td>
<td></td>
<td>Synchronization bar in activity diagrams</td>
</tr>
<tr>
<td>usecaseArrowColor</td>
<td>#FEFECE</td>
<td>#A80036</td>
<td>Color of arrows in usecase diagrams</td>
</tr>
<tr>
<td>usecaseActorBorderColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Color of actor borders in usecase diagrams</td>
</tr>
<tr>
<td>usecaseBackgroundColor</td>
<td>#FEFECE</td>
<td>#FEFECE</td>
<td>Background of usecases</td>
</tr>
<tr>
<td>usecaseBorderColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Color of usecase borders</td>
</tr>
<tr>
<td>classArrowColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Color of arrows in class diagrams</td>
</tr>
<tr>
<td>classBackgroundColor</td>
<td>#FEFECE</td>
<td>#FEFECE</td>
<td>Background of classes/interface/enum in class diagrams</td>
</tr>
<tr>
<td>classBorderColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Borders of classes/interface/enum in class diagrams</td>
</tr>
<tr>
<td>packageBackgroundColor</td>
<td>#FEFECE</td>
<td>#FEFECE</td>
<td>Background of packages in class diagrams</td>
</tr>
<tr>
<td>packageBorderColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Borders of packages in class diagrams</td>
</tr>
<tr>
<td>stereotypeCBackgroundColor</td>
<td>#ADD1B2</td>
<td>#ADD1B2</td>
<td>Background of class spots in class diagrams</td>
</tr>
<tr>
<td>stereotypeABackgroundColor</td>
<td>#A9DCDF</td>
<td>#A9DCDF</td>
<td>Background of abstract class spots in class diagrams</td>
</tr>
<tr>
<td>stereotypeIBackgroundColor</td>
<td>#B4A7E5</td>
<td>#B4A7E5</td>
<td>Background of interface spots in class diagrams</td>
</tr>
<tr>
<td>stereotypeEBackgroundColor</td>
<td>#EB937F</td>
<td>#EB937F</td>
<td>Background of enum spots in class diagrams</td>
</tr>
<tr>
<td>componentArrowColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Color of arrows in component diagrams</td>
</tr>
<tr>
<td>componentBackgroundColor</td>
<td>#FEFECE</td>
<td>#FEFECE</td>
<td>Background of components</td>
</tr>
<tr>
<td>componentBorderColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Borders of components</td>
</tr>
<tr>
<td>componentInterfaceBackgroundColor</td>
<td>#FEFECE</td>
<td>#FEFECE</td>
<td>Background of interface in component diagrams</td>
</tr>
<tr>
<td>componentInterfaceBorderColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Border of interface in component diagrams</td>
</tr>
<tr>
<td>noteBackgroundColor</td>
<td>#FBBFB77</td>
<td>#FBBFB77</td>
<td>Background of notes</td>
</tr>
<tr>
<td>noteBorderColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Border of notes</td>
</tr>
<tr>
<td>stateBackgroundColor</td>
<td>#FEFECE</td>
<td>#FEFECE</td>
<td>Background of states in state diagrams</td>
</tr>
<tr>
<td>stateBorderColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Borders of states in state diagrams</td>
</tr>
<tr>
<td>stateArrowColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Colors of arrows in state diagrams</td>
</tr>
<tr>
<td>stateStartColor</td>
<td>black</td>
<td>black</td>
<td>Starting circle in state diagrams</td>
</tr>
<tr>
<td>stateEndColor</td>
<td>black</td>
<td>black</td>
<td>Ending circle in state diagrams</td>
</tr>
<tr>
<td>sequenceArrowColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Color of arrows in sequence diagrams</td>
</tr>
<tr>
<td>sequenceActorBackgroundColor</td>
<td>#FEFECE</td>
<td>#FEFECE</td>
<td>Head’s color of actor in sequence diagrams</td>
</tr>
<tr>
<td>sequenceActorBorderColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Border of actor in sequence diagrams</td>
</tr>
<tr>
<td>sequenceGroupBackgroundColor</td>
<td>#EEEEEE</td>
<td>#EEEEEE</td>
<td>Header color of alt/opt/loop in sequence diagrams</td>
</tr>
<tr>
<td>sequenceLifeLineBackgroundColor</td>
<td>white</td>
<td>white</td>
<td>Background of life line in sequence diagrams</td>
</tr>
<tr>
<td>sequenceLineBorderColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Border of life line in sequence diagrams</td>
</tr>
<tr>
<td>sequenceParticipantBackgroundColor</td>
<td>#FEFECE</td>
<td>#FEFECE</td>
<td>Background of participant in sequence diagrams</td>
</tr>
<tr>
<td>sequenceParticipantBorderColor</td>
<td>#A80036</td>
<td>#A80036</td>
<td>Border of participant in sequence diagrams</td>
</tr>
</tbody>
</table>
12.4 Font color, name and size

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

Example:

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

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

Example:

```plantuml
skinparam defaultFontName Aapex
```

Please note the fontname is highly system dependent, so do not over use it, if you look for portability.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Default Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>activityFontColor</td>
<td>black 14 plain</td>
<td>Used for activity box</td>
</tr>
<tr>
<td>activityFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>activityFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>activityFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>activityArrowFontColor</td>
<td>black 13 plain</td>
<td>Used for text on arrows in activity diagrams</td>
</tr>
<tr>
<td>activityArrowFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>activityArrowFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>activityArrowFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>circledCharacterFontColor</td>
<td>black 17 bold</td>
<td>Used for text in circle for class, enum and others</td>
</tr>
<tr>
<td>circledCharacterFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>circledCharacterFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>circledCharacterFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>circledCharacterRadius</td>
<td>Courier 11</td>
<td></td>
</tr>
<tr>
<td>classArrowFontColor</td>
<td>black 10 plain</td>
<td>Used for text on arrows in class diagrams</td>
</tr>
<tr>
<td>classArrowFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>classArrowFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>classArrowFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>classAttributeFontColor</td>
<td>black 10 plain</td>
<td>Class attributes and methods</td>
</tr>
<tr>
<td>classAttributeFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>classAttributeIconSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>classAttributeFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>classAttributeFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>classFontColor</td>
<td>black 12 plain</td>
<td>Used for classes name</td>
</tr>
<tr>
<td>classFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>classFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>classFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>classStereotypeFontColor</td>
<td>black 12 italic</td>
<td>Used for stereotype in classes</td>
</tr>
<tr>
<td>classStereotypeFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>classStereotypeFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>classStereotypeFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>componentFontColor</td>
<td>black 14 plain</td>
<td>Used for components name</td>
</tr>
<tr>
<td>componentFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>componentFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>componentFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>componentStereotypeFontColor</td>
<td>black 14 italic</td>
<td>Used for stereotype in components</td>
</tr>
<tr>
<td>componentStereotypeFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>componentStereotypeFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>componentStereotypeFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Font/Style/Color</td>
<td>Size</td>
<td>Style</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>componentArrowFontColor</td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td>componentArrowFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>componentArrowFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>componentArrowFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>noteFontColor</td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td>noteFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>noteFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>noteFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>packageFontColor</td>
<td>black</td>
<td>14</td>
</tr>
<tr>
<td>packageFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>packageFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>packageFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceActorFontColor</td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td>sequenceActorFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceActorFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceActorFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceDividerFontColor</td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td>sequenceDividerFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceDividerFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceDividerFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceArrowFontColor</td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td>sequenceArrowFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceArrowFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceArrowFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceGroupingFontColor</td>
<td>black</td>
<td>11</td>
</tr>
<tr>
<td>sequenceGroupingFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceGroupingFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceGroupingFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceGroupingHeaderFontColor</td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td>sequenceGroupingHeaderFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceGroupingHeaderFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceGroupingHeaderFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceParticipantFontColor</td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td>sequenceParticipantFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceParticipantFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceParticipantFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceTitleFontColor</td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td>sequenceTitleFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceTitleFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sequenceTitleFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>titleFontColor</td>
<td>black</td>
<td>18</td>
</tr>
<tr>
<td>titleFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>titleFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>titleFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stateFontColor</td>
<td>black</td>
<td>14</td>
</tr>
<tr>
<td>stateFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stateFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stateFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stateArrowFontColor</td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td>stateArrowFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stateArrowFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stateArrowFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stateAttributeFontColor</td>
<td>black</td>
<td>12</td>
</tr>
<tr>
<td>stateAttributeFontSize</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stateAttributeFontStyle</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stateAttributeFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use Case Font Property</td>
<td>Color</td>
<td>Size</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>usecaseFontColor</td>
<td>black</td>
<td>14</td>
</tr>
<tr>
<td>usecaseFontSize</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>usecaseFontStyle</td>
<td>plain</td>
<td></td>
</tr>
<tr>
<td>usecaseFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>usecaseStereotypeFontColor</td>
<td>black</td>
<td>14</td>
</tr>
<tr>
<td>usecaseStereotypeFontSize</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>usecaseStereotypeFontStyle</td>
<td>italic</td>
<td></td>
</tr>
<tr>
<td>usecaseStereotypeFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>usecaseActorFontColor</td>
<td>black</td>
<td>14</td>
</tr>
<tr>
<td>usecaseActorFontSize</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>usecaseActorFontStyle</td>
<td>plain</td>
<td></td>
</tr>
<tr>
<td>usecaseActorFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>usecaseActorStereotypeFontColor</td>
<td>black</td>
<td>14</td>
</tr>
<tr>
<td>usecaseActorStereotypeFontSize</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>usecaseActorStereotypeFontStyle</td>
<td>italic</td>
<td></td>
</tr>
<tr>
<td>usecaseActorStereotypeFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>usecaseArrowFontColor</td>
<td>black</td>
<td>13</td>
</tr>
<tr>
<td>usecaseArrowFontSize</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>usecaseArrowFontStyle</td>
<td>plain</td>
<td></td>
</tr>
<tr>
<td>usecaseArrowFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>footerFontColor</td>
<td>black</td>
<td>10</td>
</tr>
<tr>
<td>footerFontSize</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>footerFontStyle</td>
<td>plain</td>
<td></td>
</tr>
<tr>
<td>footerFontName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>headerFontColor</td>
<td>black</td>
<td>10</td>
</tr>
<tr>
<td>headerFontSize</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>headerFontStyle</td>
<td>plain</td>
<td></td>
</tr>
<tr>
<td>headerFontName</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
12.5 Black and White

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

```plantuml
@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
```
13 Preprocessing

Some minor preprocessing capabilities are included in PlantUML, and available for all diagrams. Those functionalities are very similar to the C language preprocessor, except that the special character (#) has been changed to the exclamation mark (!).

13.1 Including files

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

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

```plantuml
@startuml
!include List.iuml
List <|-- ArrayList
@enduml
```

File List.iuml: interface List

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.

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.

13.2 Including URL

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

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

13.3 Constant definition

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

```plantuml
!define SEQUENCE (S,#AAAAAA) Database Sequence
!define TABLE (T,#FFAAAA) Database Table
```
Of course, you can use the `!include` directive to define all your constants in a single file that you include in your diagram.

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

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

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

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

### 13.4 Macro definition

You can also define macro with arguments.

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

Macro can have several arguments.

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

### 13.5 Macro on several lines

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

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

AUTHEN(Bob,Alice)
@enduml
```
13.6 Conditions

You can use `!ifdef XXX` and `!endif` directives to have conditional drawings. The lines between those two directives will be included only if the constant after the `!ifdef` directive has been defined before.

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

```
@startuml
!ifdef SHOW_METHODS
ArrayList : int size()
ArrayList : void clear()
!endif
@enduml
```

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

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

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

13.7 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.

13.8 Advanced features

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

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

A macro can be defined by another macro.
A macro can be polymorphic with argument count.

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

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

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

The PlantUML language uses letters to define actor, usecase and so on. But letters are not only A-Z latin characters, it could be any kind of letter from any language.

```plantuml
@startuml
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
```

14.1 Charset

The default charset used when reading the text files containing the UML text description is system dependent. Normally, it should just be fine, but in some case, you may want to use another charset. For example, with the command line:

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

Or, with the ant task:
Depending of your Java installation, the following charset should be available: ISO-8859-1, UTF-8, UTF-16BE, UTF-16LE, UTF-16.
15 Color Names

Here is the list of colors recognized by PlantUML. Note that color names are case insensitive.

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

1 **Sequence Diagram**
   1.1 Basic examples ........................................ 1
   1.2 Declaring participant .................................. 1
   1.3 Use non-letters in participants .......................... 3
   1.4 Message to Self .......................................... 3
   1.5 Change arrow style ..................................... 3
   1.6 Change arrow color ..................................... 4
   1.7 Message sequence numbering ............................ 4
   1.8 Splitting diagrams ..................................... 6
   1.9 Grouping message ....................................... 7
   1.10 Notes on messages .................................... 8
   1.11 Some other notes ...................................... 9
   1.12 Changing notes shape .................................. 10
   1.13 Creole and HTML ...................................... 10
   1.14 Divider ................................................ 11
   1.15 Reference .............................................. 12
   1.16 Delay .................................................. 12
   1.17 Space .................................................. 13
   1.18 Lifeline Activation and Destruction .................... 13
   1.19 Participant creation ................................... 15
   1.20 Incoming and outgoing messages ....................... 15
   1.21 Stereotypes and Spots ................................ 16
   1.22 More information on titles ............................ 17
   1.23 Participants encompass ................................. 19
   1.24 Removing Footer ...................................... 19
   1.25 Skinparam ............................................. 19
   1.26 Changing padding ..................................... 21

2 **Use Case Diagram**
   2.1 Usecases ............................................... 23
   2.2 Actors .................................................. 23
   2.3 Usecases description ................................... 23
   2.4 Basic example .......................................... 24
   2.5 Extension ............................................... 24
   2.6 Using notes ............................................ 25
   2.7 Stereotypes ............................................. 26
   2.8 Changing arrows direction ...................... 26
   2.9 Splitting diagrams .................................... 28
   2.10 Left to right direction ................................ 28
   2.11 Skinparam ............................................. 29
   2.12 Complete example .................................... 30
## 3 Class Diagram

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Relations between classes</td>
<td>31</td>
</tr>
<tr>
<td>3.2 Label on relations</td>
<td>32</td>
</tr>
<tr>
<td>3.3 Adding methods</td>
<td>33</td>
</tr>
<tr>
<td>3.4 Defining visibility</td>
<td>34</td>
</tr>
<tr>
<td>3.5 Abstract and Static</td>
<td>35</td>
</tr>
<tr>
<td>3.6 Advanced class body</td>
<td>36</td>
</tr>
<tr>
<td>3.7 Notes and stereotypes</td>
<td>37</td>
</tr>
<tr>
<td>3.8 More on notes</td>
<td>38</td>
</tr>
<tr>
<td>3.9 Note on links</td>
<td>39</td>
</tr>
<tr>
<td>3.10 Abstract class and interface</td>
<td>40</td>
</tr>
<tr>
<td>3.11 Using non-letters</td>
<td>41</td>
</tr>
<tr>
<td>3.12 Hide attributes, methods...</td>
<td>42</td>
</tr>
<tr>
<td>3.13 Hide classes</td>
<td>43</td>
</tr>
<tr>
<td>3.14 Use generics</td>
<td>43</td>
</tr>
<tr>
<td>3.15 Specific Spot</td>
<td>43</td>
</tr>
<tr>
<td>3.16 Packages</td>
<td>44</td>
</tr>
<tr>
<td>3.17 Packages style</td>
<td>44</td>
</tr>
<tr>
<td>3.18 Namespaces</td>
<td>45</td>
</tr>
<tr>
<td>3.19 Automatic namespace creation</td>
<td>46</td>
</tr>
<tr>
<td>3.20 Lollipop interface</td>
<td>47</td>
</tr>
<tr>
<td>3.21 Changing arrows direction</td>
<td>47</td>
</tr>
<tr>
<td>3.22 Association classes</td>
<td>48</td>
</tr>
<tr>
<td>3.23 Skinparam</td>
<td>49</td>
</tr>
<tr>
<td>3.24 Skinned Stereotypes</td>
<td>50</td>
</tr>
<tr>
<td>3.25 Color gradient</td>
<td>50</td>
</tr>
<tr>
<td>3.26 Help on layout</td>
<td>51</td>
</tr>
<tr>
<td>3.27 Splitting large files</td>
<td>52</td>
</tr>
</tbody>
</table>

## 4 Activity Diagram

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Simple Activity</td>
<td>54</td>
</tr>
<tr>
<td>4.2 Label on arrows</td>
<td>54</td>
</tr>
<tr>
<td>4.3 Changing arrow direction</td>
<td>54</td>
</tr>
<tr>
<td>4.4 Branches</td>
<td>55</td>
</tr>
<tr>
<td>4.5 More on Branches</td>
<td>56</td>
</tr>
<tr>
<td>4.6 Synchronization</td>
<td>57</td>
</tr>
<tr>
<td>4.7 Long activity description</td>
<td>58</td>
</tr>
<tr>
<td>4.8 Notes</td>
<td>58</td>
</tr>
<tr>
<td>4.9 Partition</td>
<td>59</td>
</tr>
<tr>
<td>4.10 Skinparam</td>
<td>60</td>
</tr>
<tr>
<td>4.11 Octagon</td>
<td>61</td>
</tr>
<tr>
<td>4.12 Complete example</td>
<td>61</td>
</tr>
</tbody>
</table>
### 5 Activity Diagram (beta)

- 5.1 Simple Activity ........................................... 64
- 5.2 Start/Stop .................................................. 64
- 5.3 Conditional .................................................. 65
- 5.4 Repeat loop .................................................. 66
- 5.5 While loop .................................................... 66
- 5.6 Parallel processing ......................................... 67
- 5.7 Notes ......................................................... 68
- 5.8 Colors ......................................................... 68
- 5.9 Arrows ......................................................... 69
- 5.10 Connector .................................................... 69
- 5.11 Grouping ..................................................... 70
- 5.12 Swimlanes .................................................... 70
- 5.13 Detach ......................................................... 71
- 5.14 SDL .......................................................... 72
- 5.15 Complete example ......................................... 73

### 6 Component Diagram

- 6.1 Components .................................................. 75
- 6.2 Interfaces ...................................................... 75
- 6.3 Basic example ................................................ 75
- 6.4 Using notes ................................................... 76
- 6.5 Grouping Components ...................................... 76
- 6.6 Changing arrows direction ................................ 78
- 6.7 Use UML2 notation ......................................... 79
- 6.8 Long description ............................................ 80
- 6.9 Individual colors ............................................ 80
- 6.10 Using Sprite in Stereotype ............................... 80
- 6.11 Skinparam .................................................... 81

### 7 State Diagram

- 7.1 Simple State ................................................ 83
- 7.2 Composite state ............................................. 83
- 7.3 Long name .................................................... 84
- 7.4 Concurrent state ............................................. 85
- 7.5 Arrow direction .............................................. 86
- 7.6 Note .......................................................... 87
- 7.7 More in notes ............................................... 88
- 7.8 Skinparam .................................................... 88

### 8 Object Diagram

- 8.1 Definition of objects ....................................... 90
- 8.2 Relations between objects ............................... 90
- 8.3 Adding fields ................................................ 90
- 8.4 Common features with class diagrams .................. 91
## 9 Common commands

9.1 Footer and header ............................................. 92  
9.2 Zoom ..................................................................... 92

## 10 Salt

10.1 Basic widgets ..................................................... 93  
10.2 Using grid ................................................................ 93  
10.3 Using separator ................................................... 94  
10.4 Tree widget ......................................................... 94  
10.5 Enclosing brackets ............................................... 95  
10.6 Adding tabs ........................................................ 95  
10.7 Using menu ........................................................ 96  
10.8 Advanced table .................................................... 97

## 11 Creole

11.1 Emphasized text ................................................... 98  
11.2 List ......................................................................... 98  
11.3 Escape character ................................................... 99  
11.4 Horizontal lines ..................................................... 99  
11.5 Headings .............................................................. 99  
11.6 Legacy HTML ........................................................ 99  
11.7 Table ....................................................................... 100  
11.8 Tree ........................................................................ 100  
11.9 Special characters .................................................. 100  
11.10 OpenIconic .......................................................... 101  
11.11 Defining and using sprites ....................................... 102  
11.12 Encoding Sprite ..................................................... 102  
11.13 Importing Sprite ..................................................... 102  
11.14 Examples ............................................................. 102

## 12 Changing fonts and colors

12.1 Usage ....................................................................... 104  
12.2 Nested ..................................................................... 104  
12.3 Color ....................................................................... 105  
12.4 Font color, name and size ........................................ 106  
12.5 Black and White ...................................................... 109

## 13 Preprocessing

13.1 Including files ....................................................... 110  
13.2 Including URL ....................................................... 110  
13.3 Constant definition ................................................ 110  
13.4 Macro definition ..................................................... 111  
13.5 Macro on several lines ............................................. 111  
13.6 Conditions ............................................................. 112  
13.7 Search path ........................................................... 112  
13.8 Advanced features ............................................... 112
14 Internationalization 114
  14.1 Charset ................................................................. 114

15 Color Names 116