

SparkChart Pro - Sequence Diagram

Introduction

Welcome to the **SparkChart Pro** Sequence Diagram Documentation. This guide provides a comprehensive reference for creating sequence diagrams using PlantUML syntax within SparkChart Pro. It covers everything from basic interactions to advanced features like grouping, note formatting, and custom styling.

Table of Contents

1. [Sequence Diagram Syntax](#)
 2. [Participants & Lifelines](#)
 3. [Messages & Arrows](#)
 4. [Message Sequence Numbering](#)
 5. [Notes & Fragments](#)
 6. [Styling & Skinparams](#)
 7. [Mermaid Support](#)
-

Sequence Diagram Syntax

PlantUML streamlines the creation of sequence diagrams through its intuitive and user-friendly syntax. This approach allows both novices and experienced designers to quickly transition from concept to a polished graphical output.

Basic Examples

The `->` sequence denotes a message sent between two participants. Participants are automatically recognized. Dotted arrows are used for responses.

@startuml

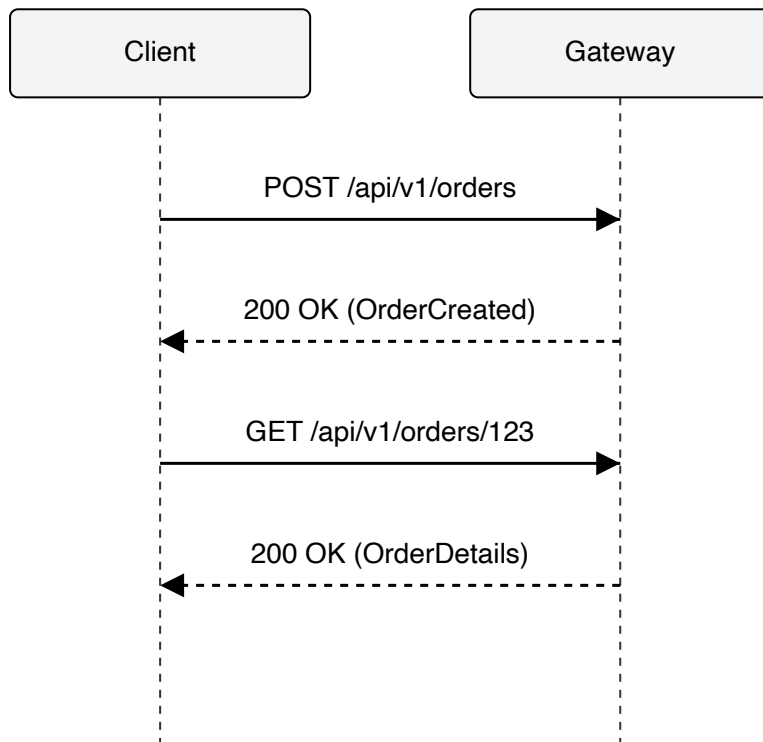
Client -> Gateway: POST /api/v1/orders

Gateway --> Client: 200 OK (OrderCreated)

Client -> Gateway: GET /api/v1/orders/123

Client <-- Gateway: 200 OK (OrderDetails)

@enduml



Participants & Lifelines

Declaring Participant

You can explicitly declare participants to control their order or assign specific shapes like **actor**, **boundary**, **control**, etc.

@startuml

actor User as user
participant "Front End" as fe
control "API Controller" as api
entity "User Service" as service
database "PostgreSQL" as db
queue "Kafka Topic" as queue

user -> fe: Click "Register"

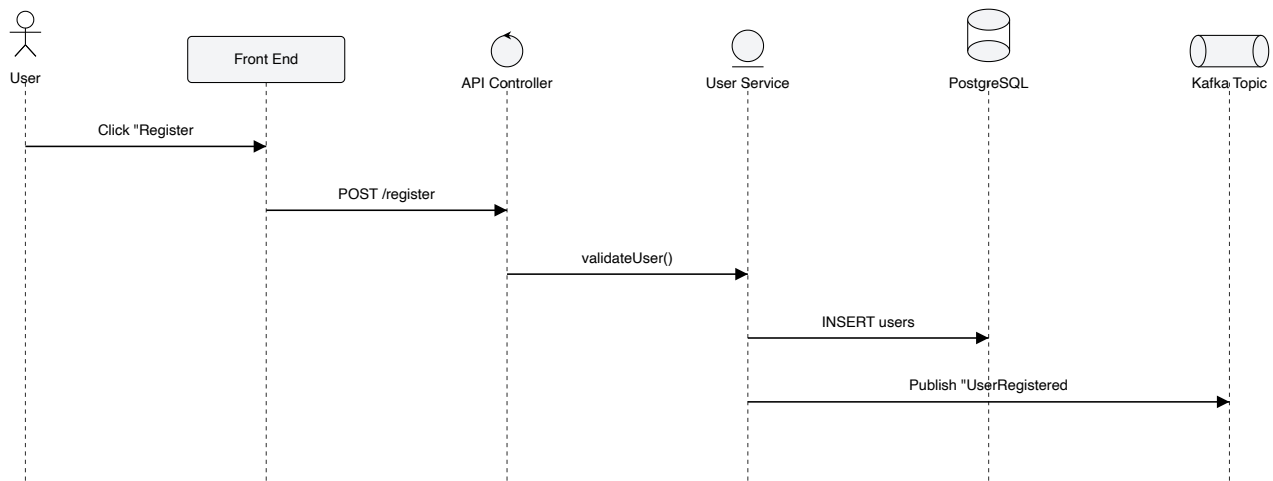
fe -> api: POST /register

api -> service: validateUser()

service -> db: INSERT users

service -> queue: Publish "UserRegistered"

@enduml



You can rename participants using **as** and change background colors.

@startuml

actor "Customer" as c #red
participant "Payment Proxy" as p #99FF99
participant "Bank System" as b

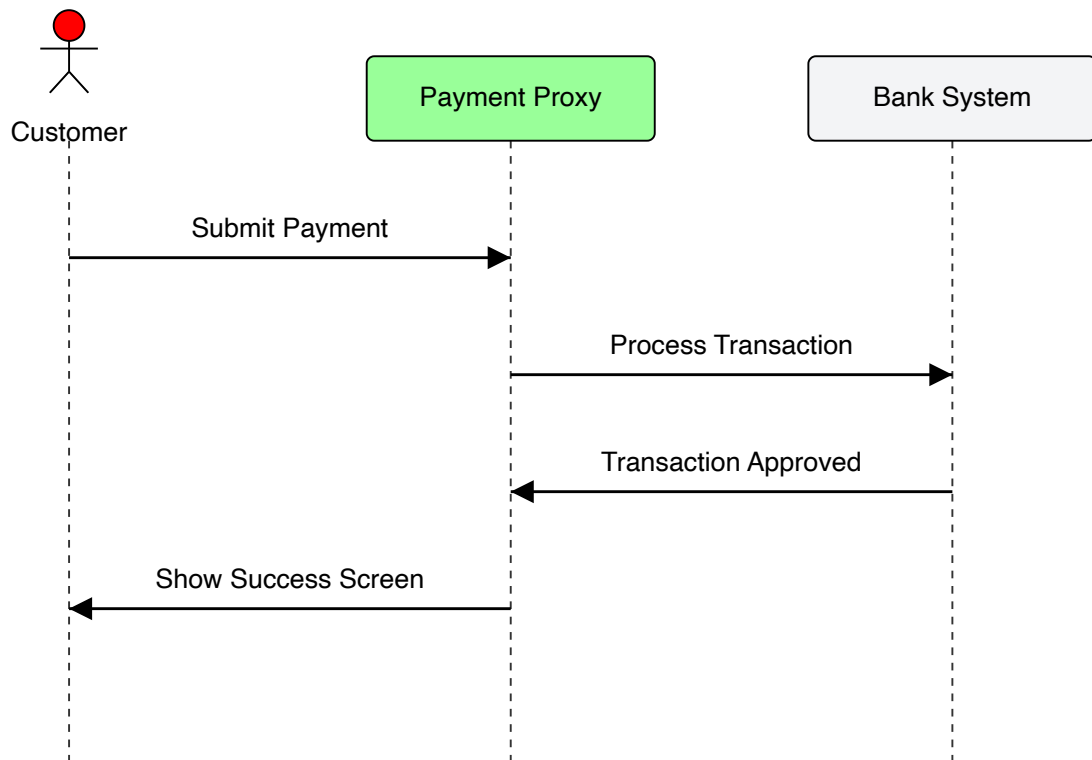
c -> p: Submit Payment

p -> b: Process Transaction

b -> p: Transaction Approved

p -> c: Show Success Screen

@enduml



Participant Creation

You can create participants dynamically using the **create** keyword. The participant's lifeline will start at the point of creation.

@startuml

participant User

User -> "OrderController" as ctrl : createOrder(items)

activate ctrl

create control "OrderService" as service

ctrl -> service : new()

activate service

note right : Service created dynamically

service -> service : validateItems()

create entity "Order" as order

service -> order : create(items)

note right : Order entity created

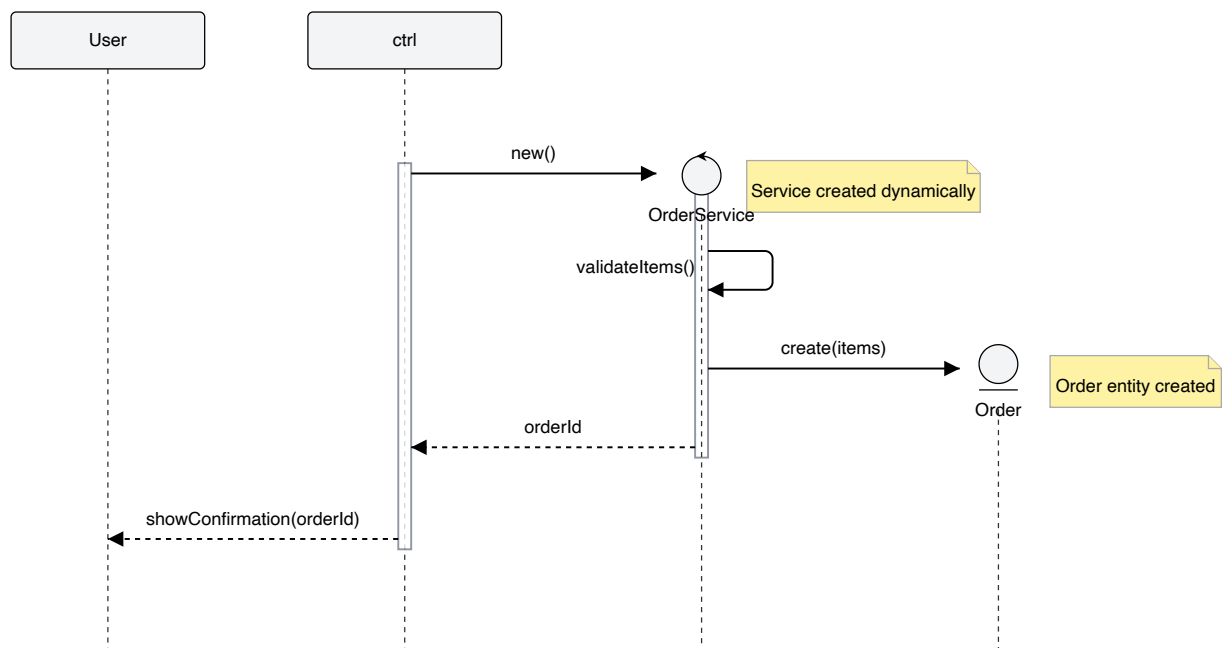
service --> ctrl : orderId

deactivate service

ctrl --> User : showConfirmation(orderId)

deactivate ctrl

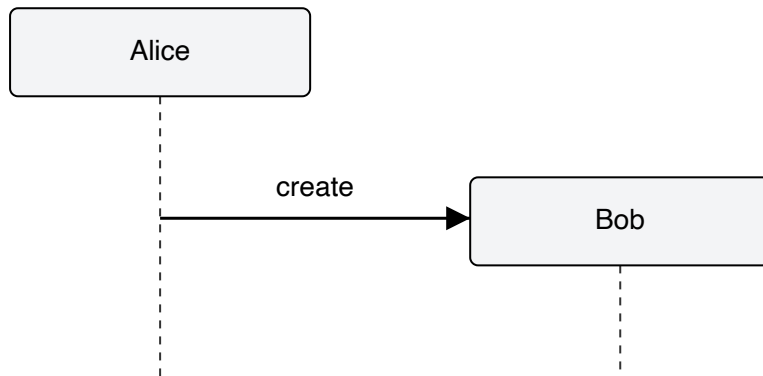
@enduml



Shortcut Syntax for Participant Creation

You can create participants immediately upon sending a message using the ****** suffix.

```
@startuml
participant Alice
Alice -> Bob **: create
@enduml
```



This is equivalent to explicitly declaring **create Bob** followed by the message.

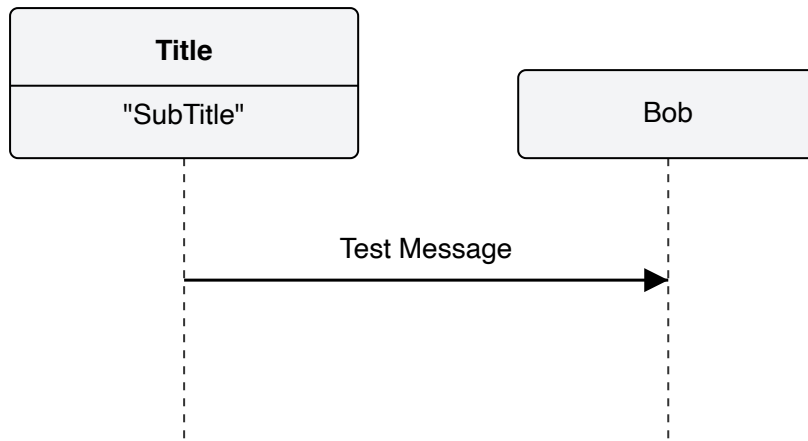
Multiline Definition

You can define participants on multiple lines to include titles, subtitles, or other formatted text.

```
@startuml
participant Participant [
    =Title
    ----
    ""Subtitle""
]

participant Bob

Participant -> Bob: Test Message
@enduml
```

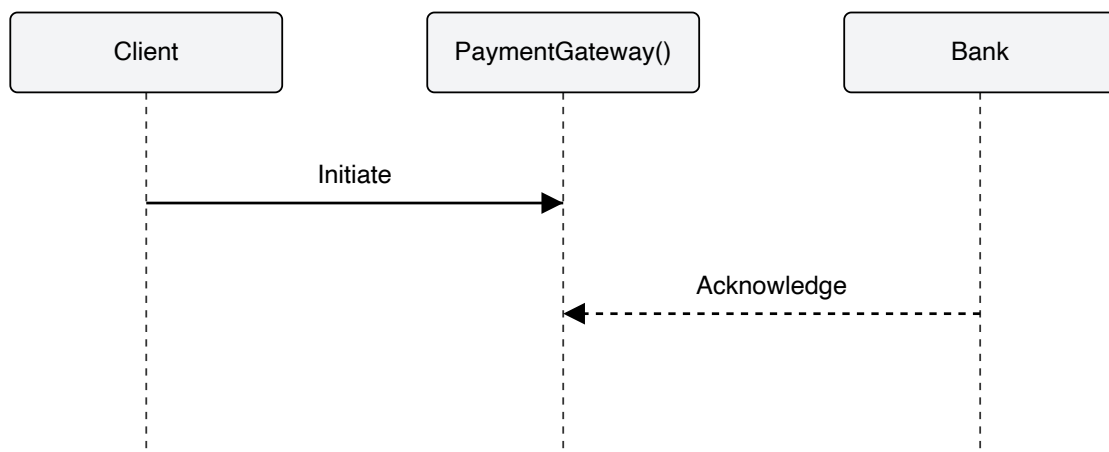


This format allows for richer participant descriptions.

Use Non-letters in Participants

Use quotes to define participants with non-letter characters.

```
@startuml
Client -> "PaymentGateway()" : Initiate
"PaymentGateway()" -> "Bank:Network" as Bank
Bank --> "PaymentGateway()" : Acknowledge
@enduml
```



Lifeline Activation and Destruction

@startuml

participant User

User -> UI: Click "Generate Report"

activate UI

UI -> GenService: Request Generation

activate GenService

GenService -> Worker: Spawn Worker

activate Worker

Worker --> GenService: Done

destroy Worker

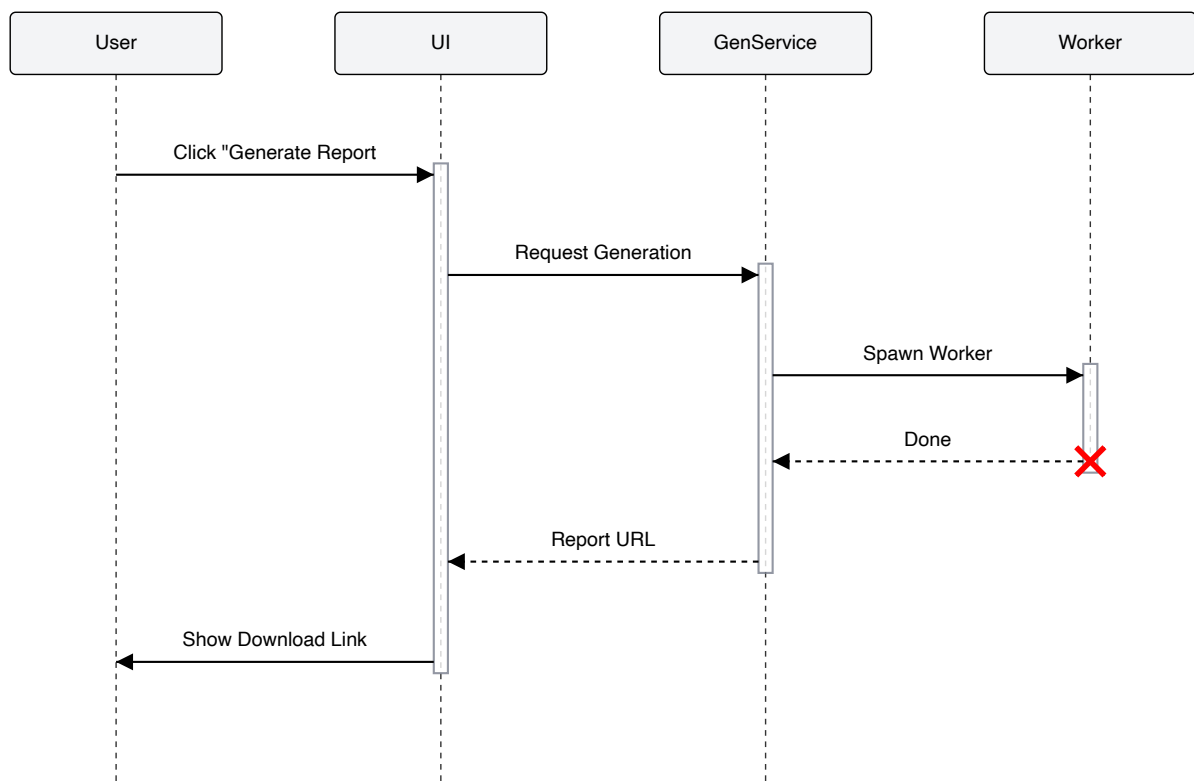
GenService --> UI: Report URL

deactivate GenService

UI -> User: Show Download Link

deactivate UI

@enduml



Shortcut Syntax for Activation

Use ++, --.

```
@startuml
```

```
Client -> Gateway ++ : Request
```

```
Gateway -> Service ++ : Forward
```

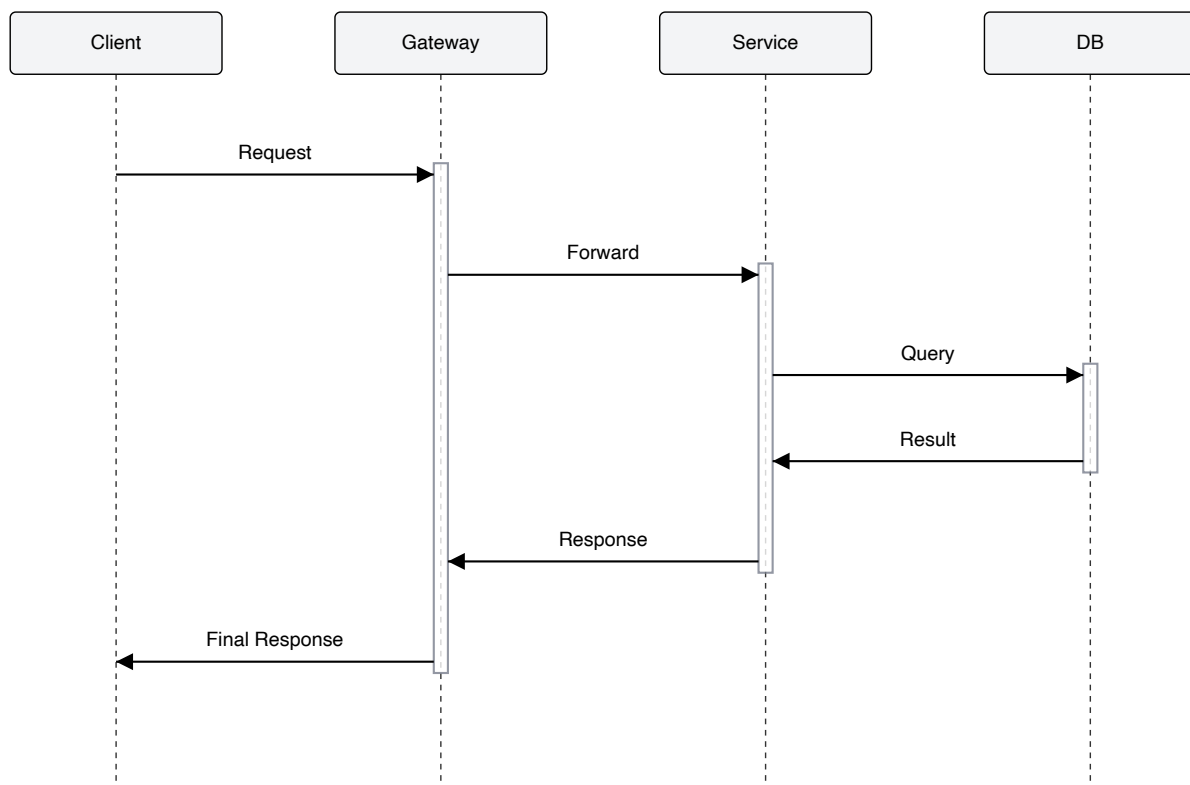
```
Service -> DB ++ #005500 : Query
```

```
DB -> Service --: Result
```

```
Service -> Gateway --: Response
```

```
Gateway -> Client --: Final Response
```

```
@enduml
```



Shortcut Syntax for Destruction

You can automatically destroy a participant after a message using the !! suffix.

```
@startuml
```

```
participant Alice
```

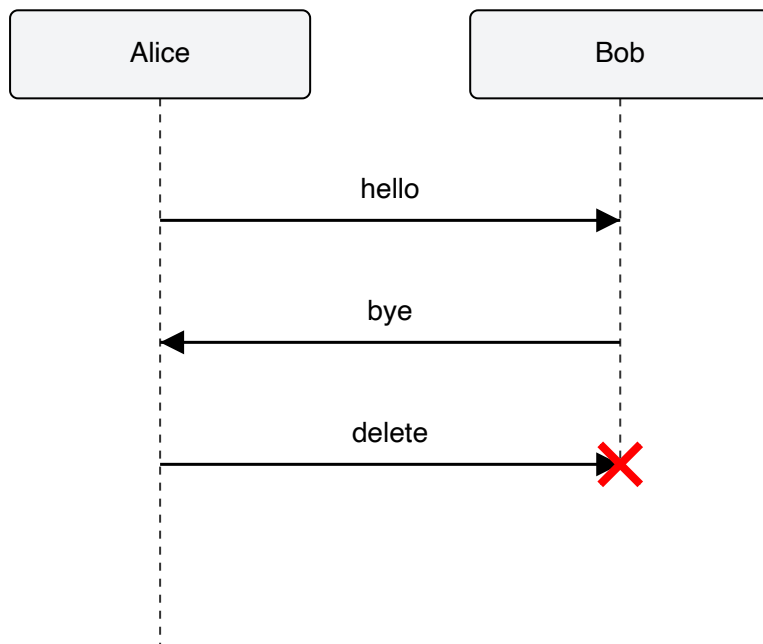
```
participant Bob
```

```
Alice -> Bob : hello
```

```
Bob -> Alice : bye
```

```
Alice -> Bob !! : delete
```

```
@enduml
```



Participants Encompass (Box)

Draw a box around participants to group them visually.

```
@startuml
```

```
box "Internal Network" #LightBlue
```

```
participant Gateway
```

```
participant AuthServer
```

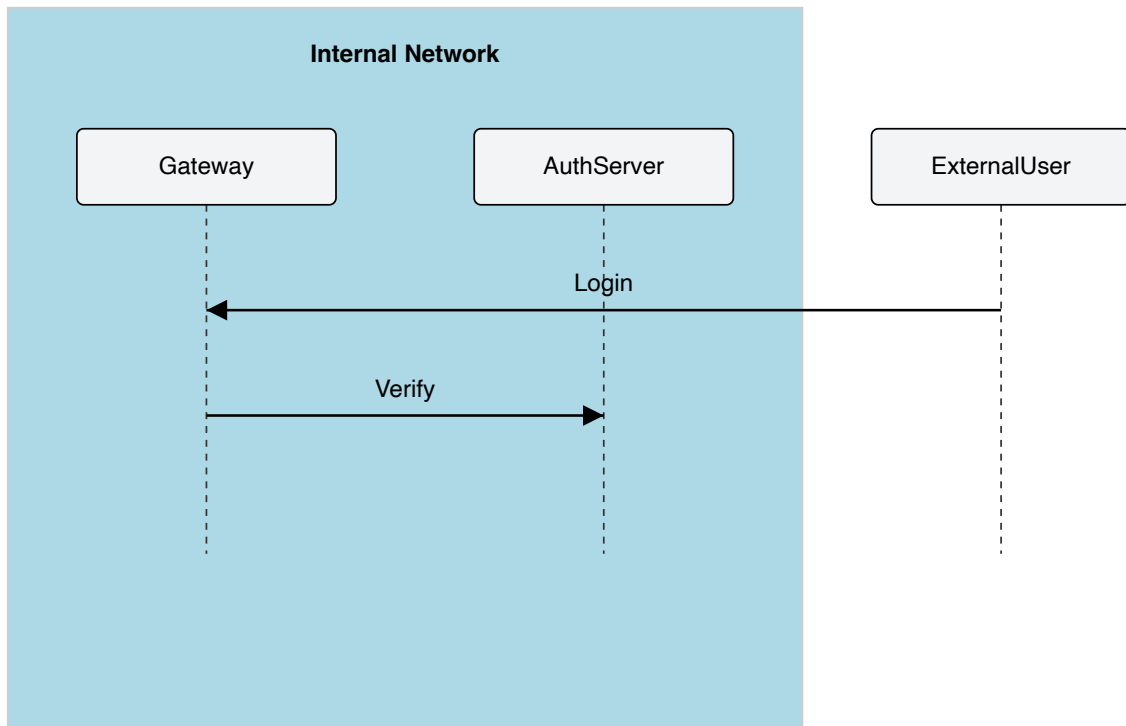
```
end box
```

```
participant ExternalUser
```

```
ExternalUser -> Gateway : Login
```

```
Gateway -> AuthServer : Verify
```

```
@enduml
```

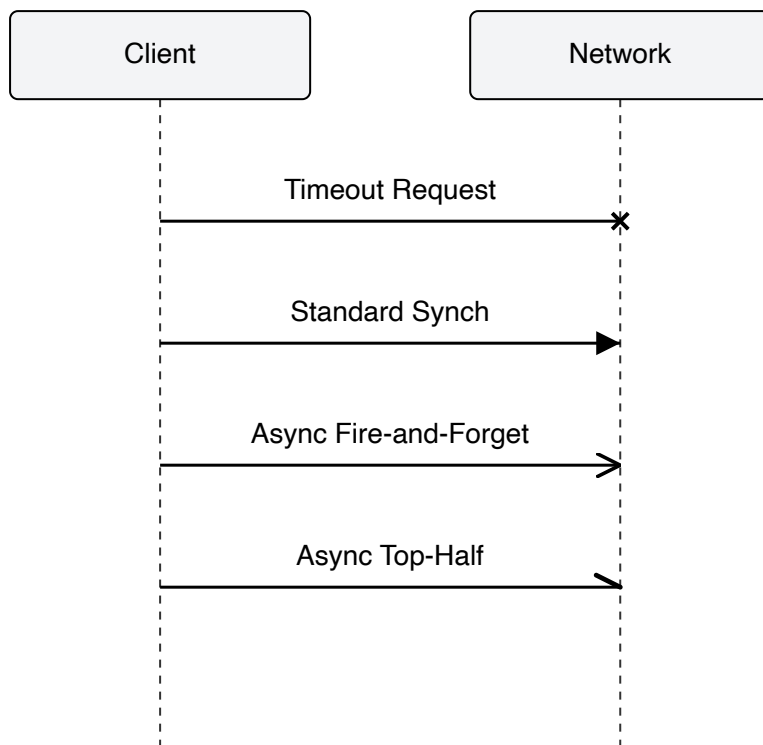


Messages & Arrows

Change Arrow Style

Modify arrows to indicate lost messages, distinct directions, etc.

```
@startuml
Client ->x Network: Timeout Request
Client -> Network: Standard Synch
Client ->> Network: Async Fire-and-Forget
Client -\\ Network: Async Top-Half
@enduml
```



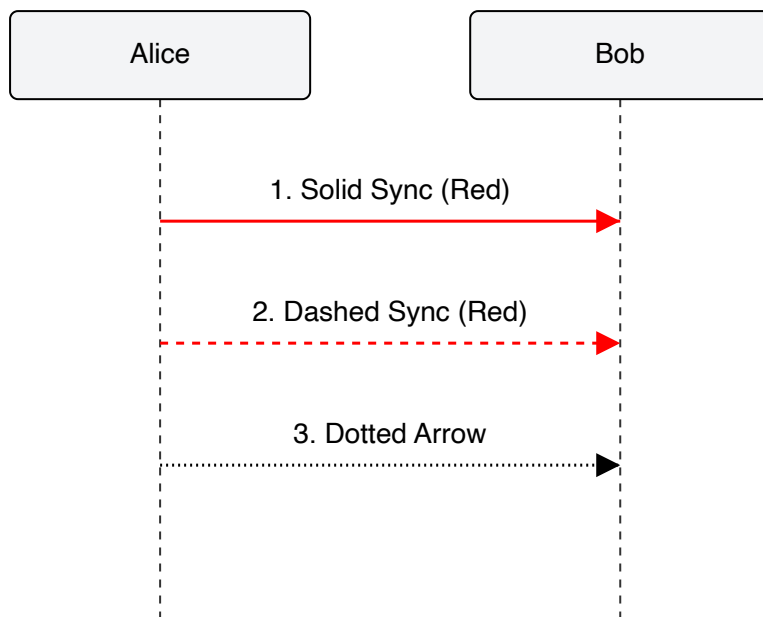
Advanced Arrow Syntax

Complex arrow styles including colored, bidirectional, half-arrows, and external signals.

Colored & Styled Arrows

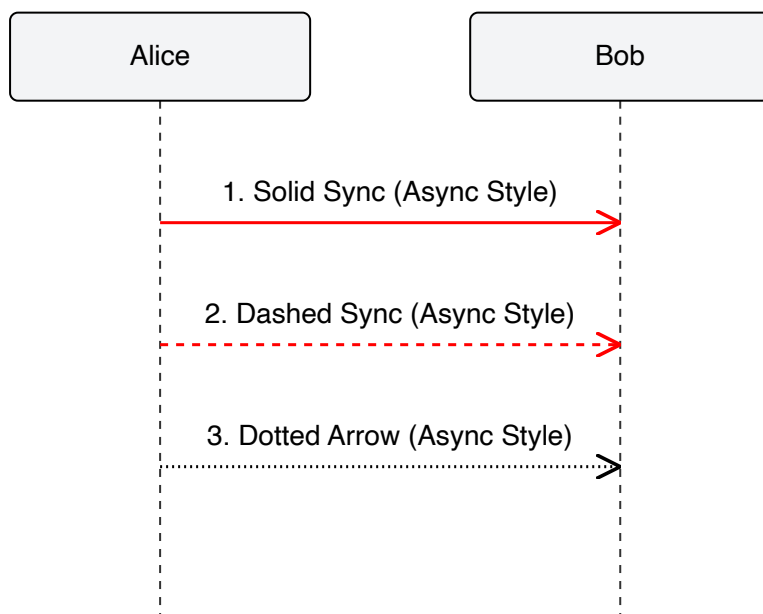
Basic Colored Arrows

```
@startuml
Alice -[#Red]> Bob: Solid Sync (Red)
Alice -[#f00]-> Bob: Dashed Sync (Red)
Alice -[dotted]> Bob: Dotted Arrow
@enduml
```



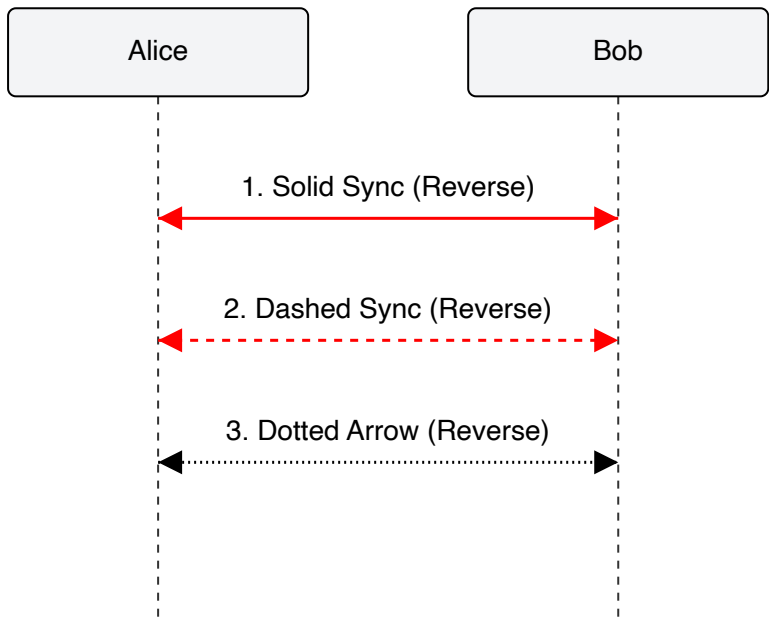
Async Style

```
@startuml
Alice -[#f00]>> Bob: Solid Sync (Async Style)
Alice -[#f00]->> Bob: Dashed Sync (Async Style)
Alice -[dotted]>> Bob: Dotted Arrow (Async Style)
@enduml
```



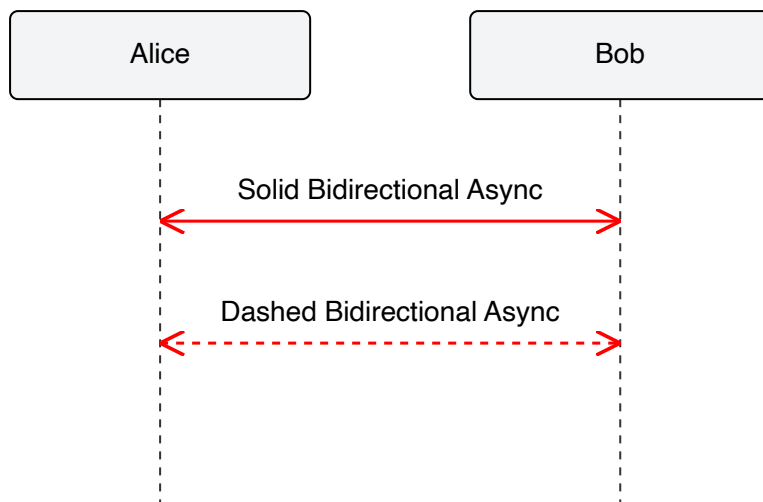
Reverse Arrows

```
@startuml
Alice <-[#f00]> Bob: Solid Sync (Reverse)
Alice <-[#f00]-> Bob: Dashed Sync (Reverse)
Alice <-[dotted]> Bob: Dotted Arrow (Reverse)
@enduml
```



Bidirectional Arrows

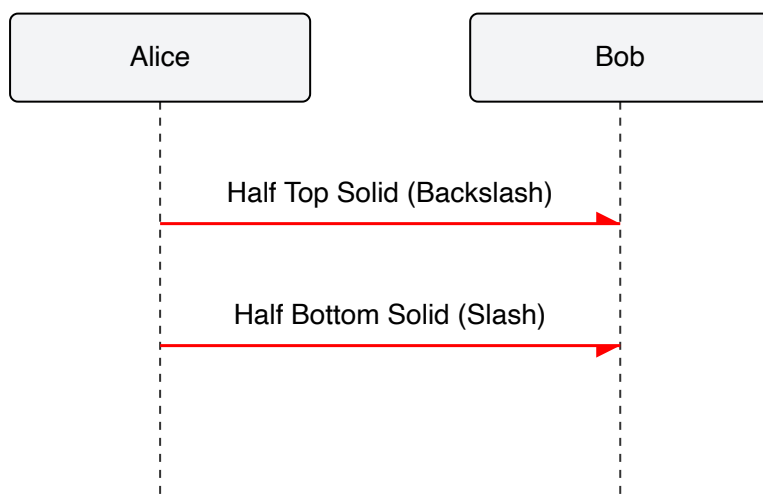
```
@startuml
Alice <<-[#f00]>> Bob: Solid Bidirectional Async
Alice <<-[#f00]->> Bob: Dashed Bidirectional Async
@enduml
```



Half Arrows

Basic Half Arrows

```
@startuml
Alice -[#f00]\ Bob: Half Top Solid (Backslash)
Alice -[#f00]/ Bob: Half Bottom Solid (Slash)
@enduml
```



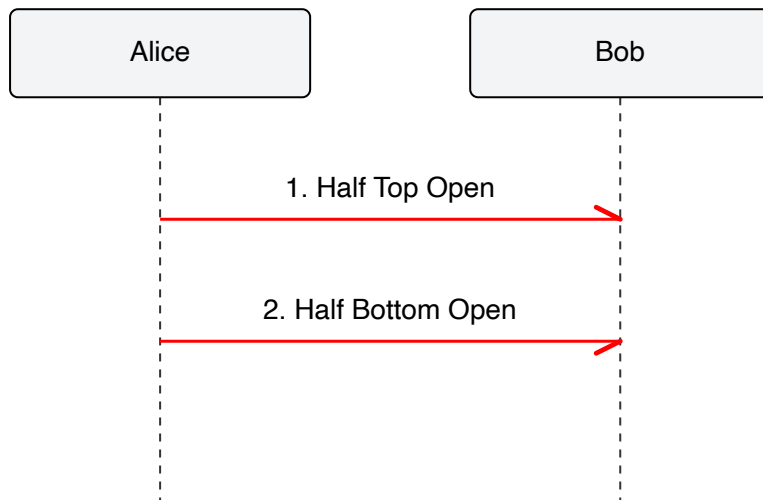
Half Arrows Open

@startuml

Alice -[#f00]\\ Bob: Half Top Open

Alice -[#f00]// Bob: Half Bottom Open

@enduml



Reverse Half Arrows

@startuml

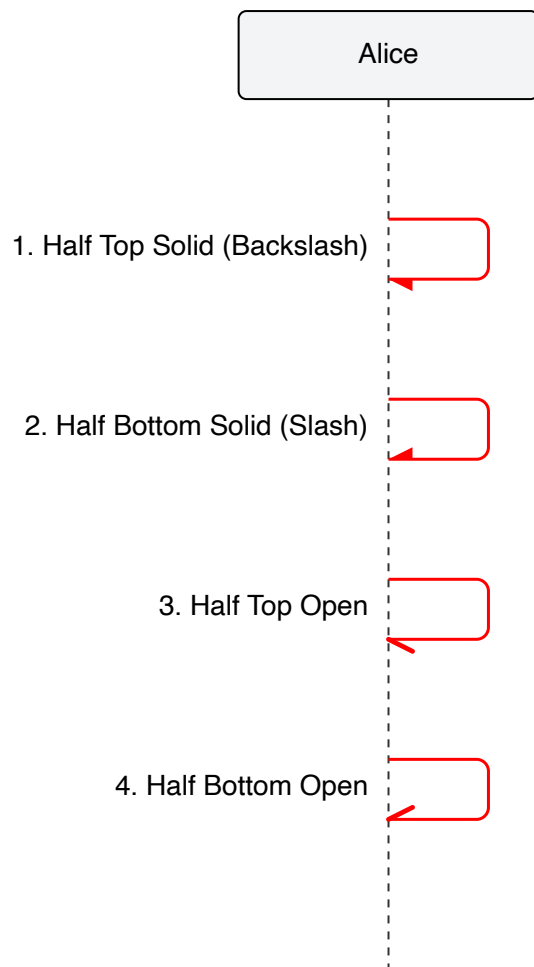
Bob -[#f00]\ Alice: Half Top (Reverse)

Bob -[#f00]/ Alice: Half Bottom (Reverse)

Bob -[#f00]\\ Alice: Half Top Open (Reverse)

Bob -[#f00]// Alice: Half Bottom Open (Reverse)

@enduml



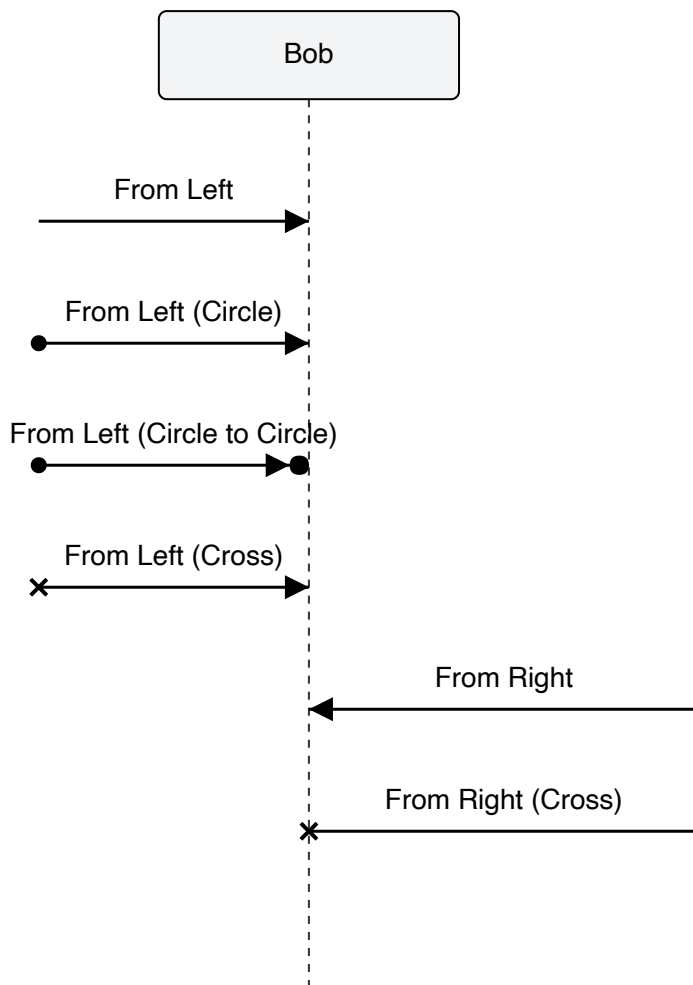
External Signals

Signals starting from or ending at the diagram boundary.

Incoming Messages

```
@startuml
[-> Bob: From Left
[o-> Bob: From Left (Circle)
[o->o Bob: From Left (Circle to Circle)
[x-> Bob: From Left (Cross)

Bob <-]: From Right
Bob x<-]: From Right (Cross)
@enduml
```



Outgoing Messages

@startuml

[<- Bob: To Left

[x<- Bob: To Left (Cross)

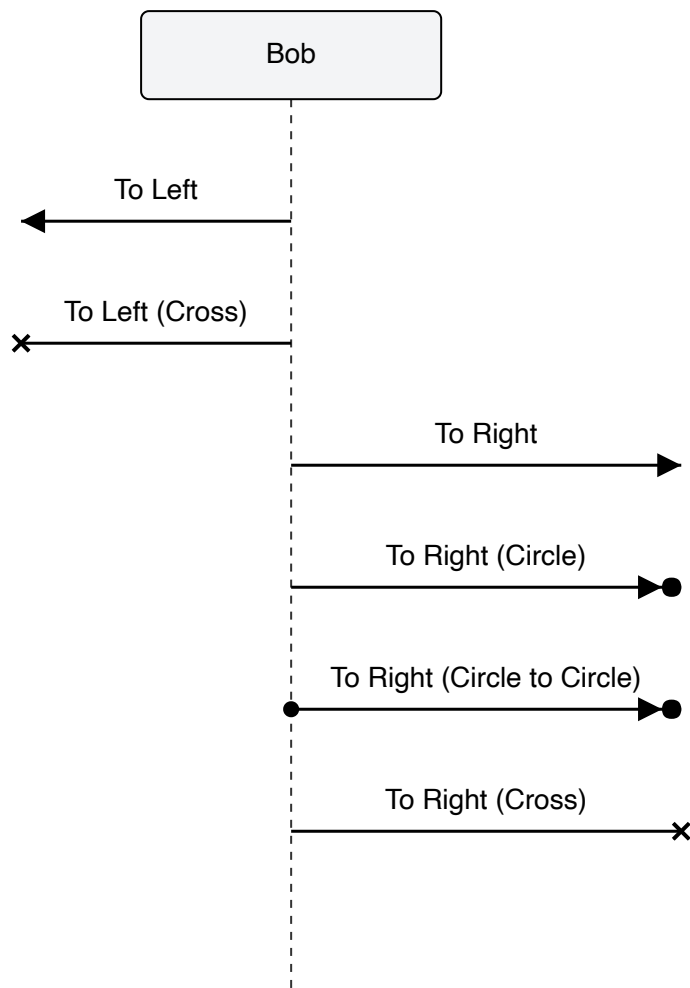
Bob ->]: To Right

Bob ->o]: To Right (Circle)

Bob o->o]: To Right (Circle to Circle)

Bob ->x]: To Right (Cross)

@enduml



Return

The **return** keyword generates a return message with an optional text label. It automatically identifies the point that caused the most recent lifeline activation and draws a dashed arrow back to the caller. The explicit **activate** commands are required for this to track the stack correctly.

@startuml

participant User

participant Server

participant Database

User -> Server: **Login**(username, password)

activate Server

Server -> Database: **getUser**(username)

activate Database

return user_record

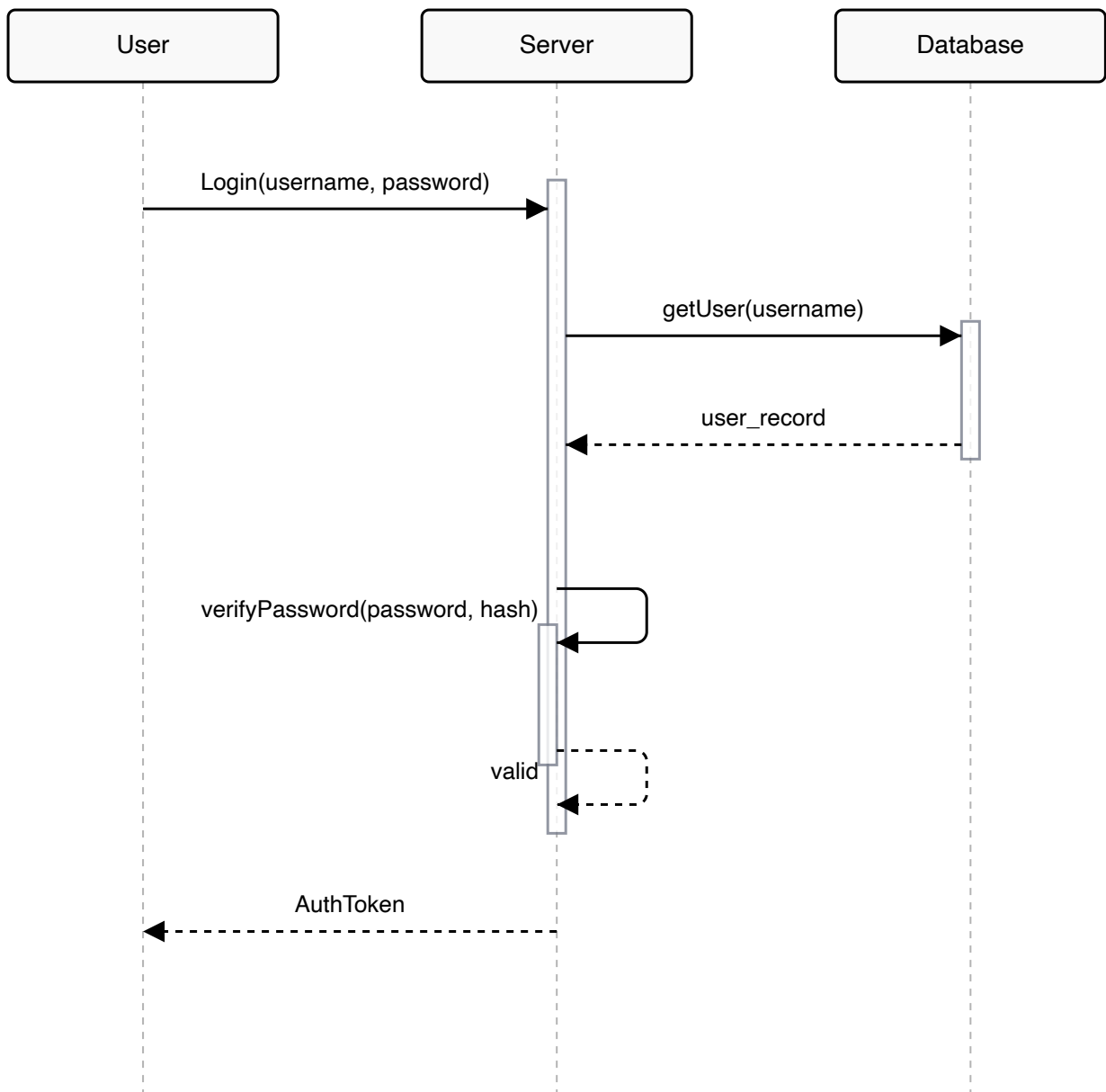
Server -> Server: **verifyPassword**(password, hash)

activate Server

return valid

return AuthToken

@enduml



Slanted or odd arrows

You can use the `(nn)` option (before or after arrow) to make the arrows slanted, where `nn` is the number of shift pixels.

@startuml

' Standard latency example

Client ->(50) Server: Request (High Latency)

Server ->(10) Client: Response (Low Latency)

' Complex routing

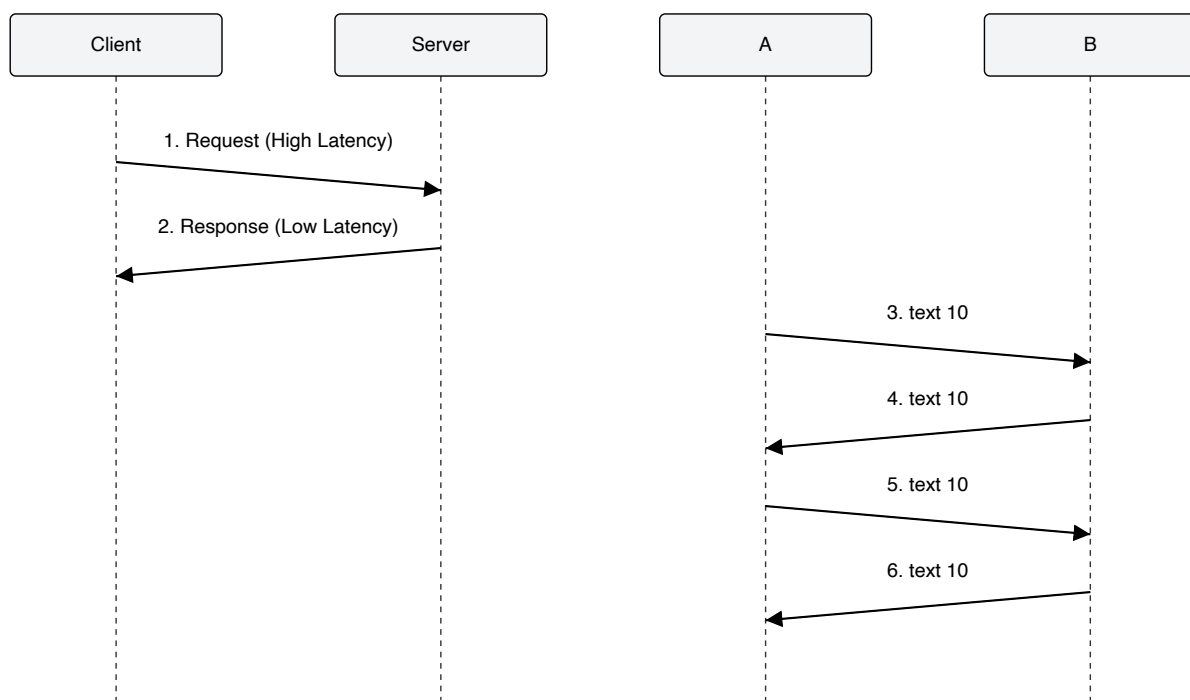
A ->(10) B: text 10

B ->(10) A: text 10

A ->(10) B: text 10

A (10)<- B: text 10

@enduml



Parallel messages

You can use the `&` command to display parallel messages. This is particularly useful for modelling non-blocking operations or concurrent events.

@startuml

participant User

participant "Mobile App" as App

participant "Backend API" as API

participant "Analytics Service" as Analytics

User -> App: Click "Submit Order"

activate App

App -> API: POST /order

& API -> Analytics: Log "Order Submitted" event

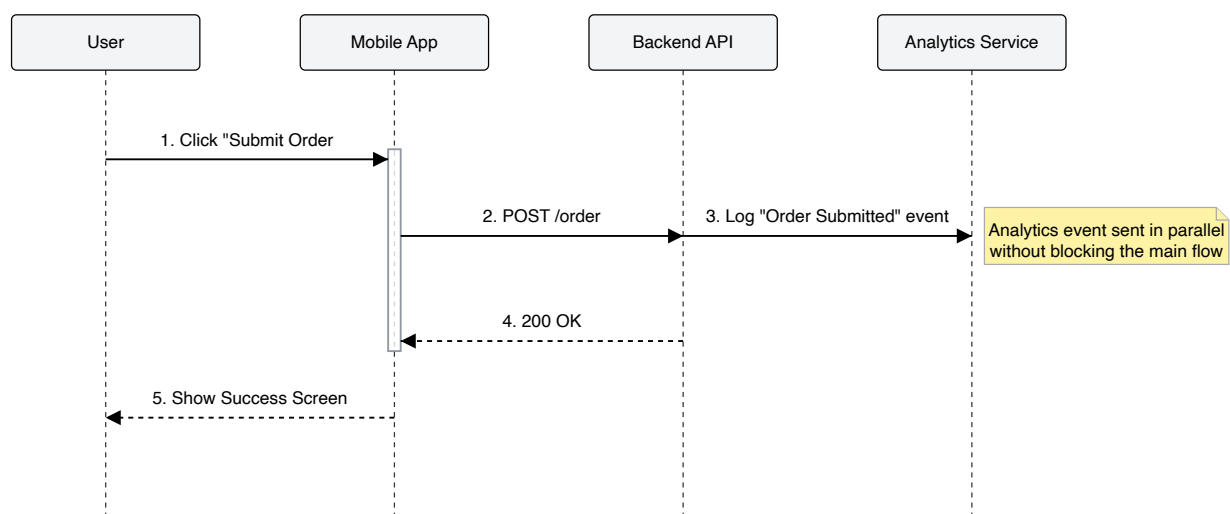
note right: Analytics event sent in parallel\nwithout blocking the main flow

API --> App: 200 OK

deactivate App

App --> User: Show Success Screen

@enduml



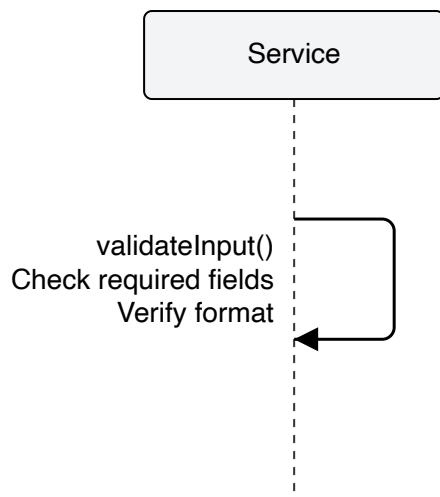
Message to Self

A participant can send a message to itself.

@startuml

Service -> Service: validateInput()\nCheck required fields\nVerify format

@enduml

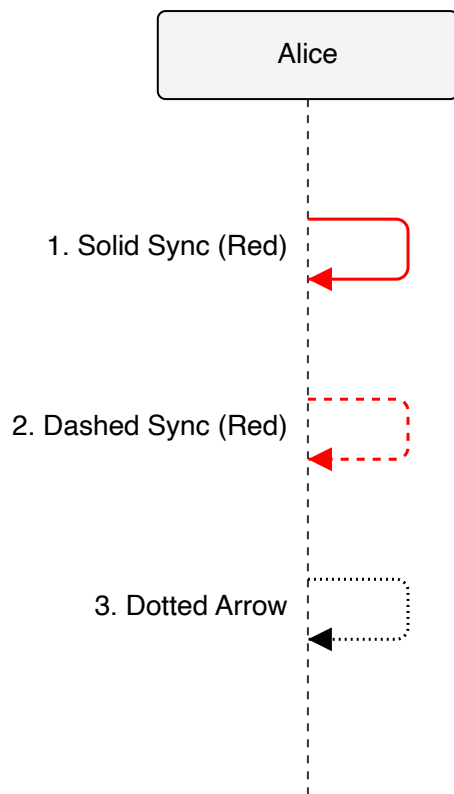


Self-Message Variations

Self-messages support various styles including colors, line types (dashed, dotted), and different arrowheads.

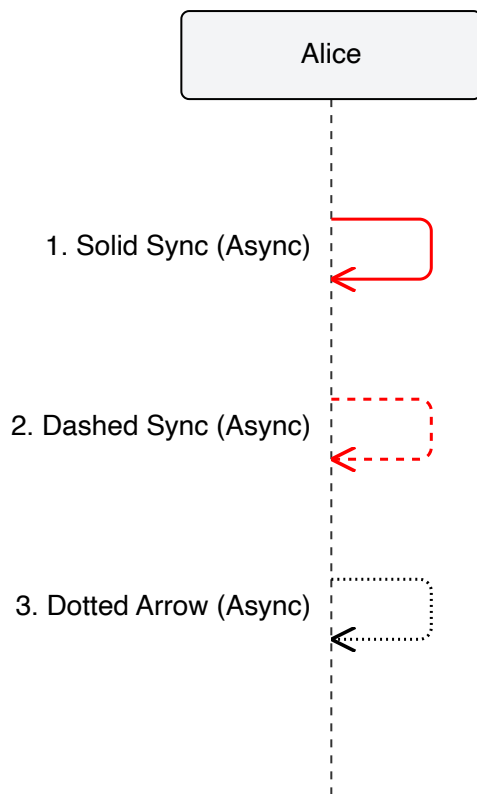
Basic Styles

```
@startuml
Alice -[#Red]> Alice: Solid Sync (Red)
Alice -[#f00]-> Alice: Dashed Sync (Red)
Alice -[dotted]> Alice: Dotted Arrow
@enduml
```

Async Styles

```
@startuml
Alice -[#f00]>> Alice: Solid Sync (Async)
Alice -[#f00]->> Alice: Dashed Sync (Async)
Alice -[dotted]>> Alice: Dotted Arrow (Async)
@enduml
```



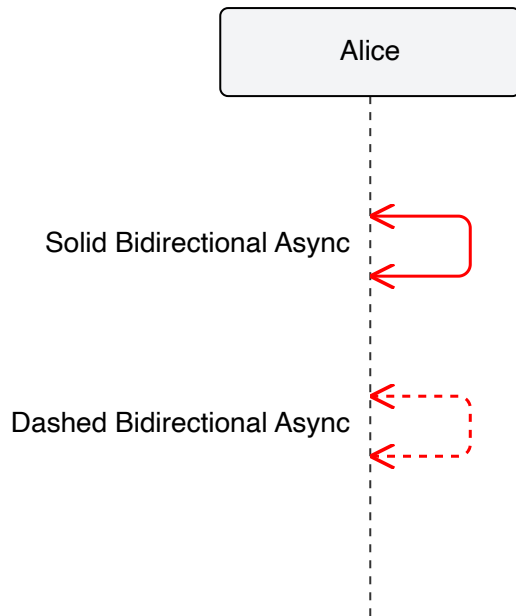
Bidirectional Styles

```
@startuml
```

```
Alice <<-[#f00]>> Alice: Solid Bidirectional Async
```

```
Alice <<-[#f00]->> Alice: Dashed Bidirectional Async
```

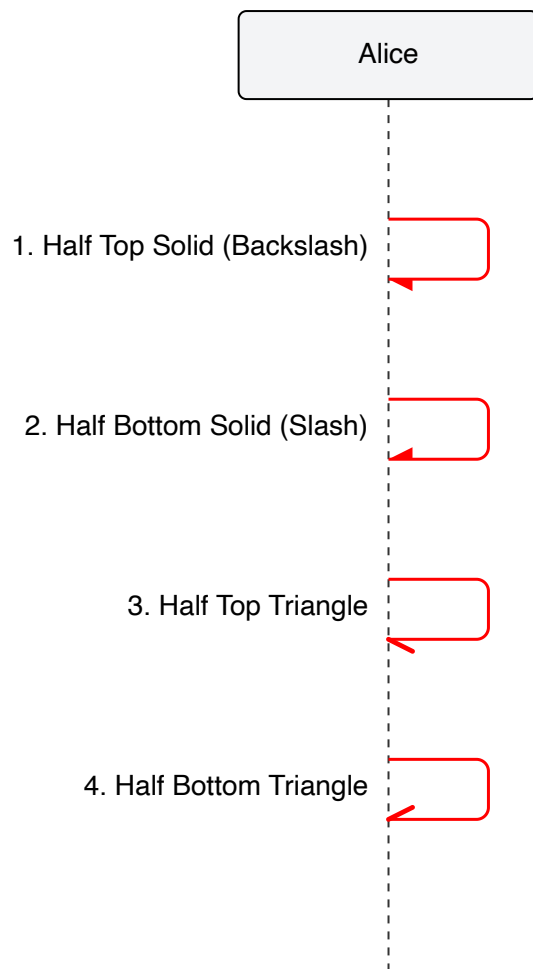
```
@enduml
```



Half Arrow Styles

```
@startuml
Alice -[#f00]\ Alice: Half Top Solid (Backslash)
Alice -[#f00]/ Alice: Half Bottom Solid (Slash)

Alice -[#f00]\\ Alice: Half Top Open
Alice -[#f00]// Alice: Half Bottom Open
@enduml
```



Incoming and Outgoing (Detailed)

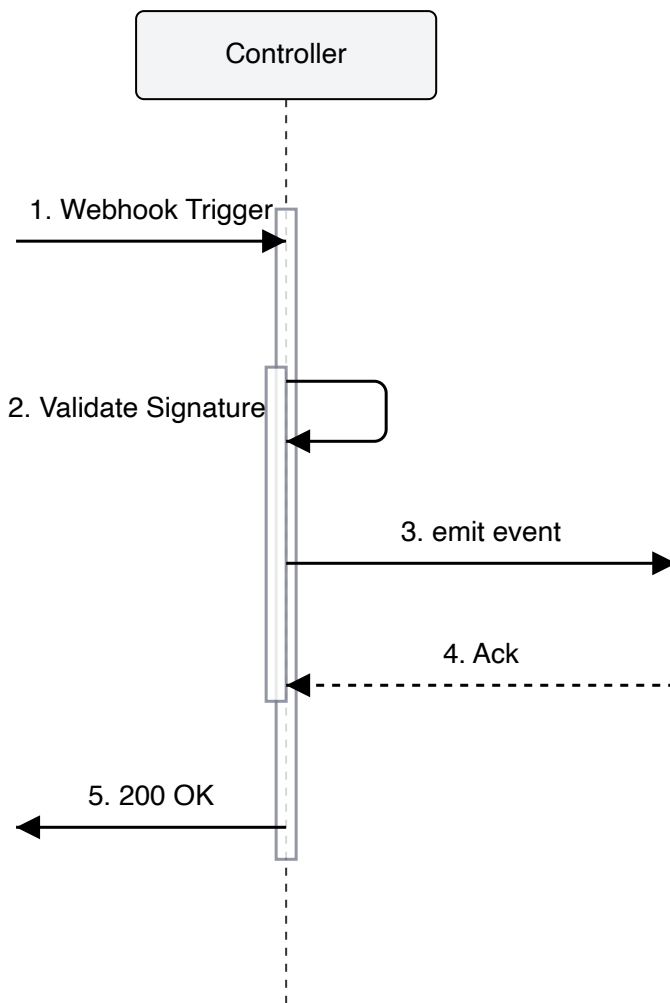
```
@startuml
[-> Controller: Webhook Trigger

activate Controller

Controller -> Controller: Validate Signature
activate Controller

Controller ->] : emit event

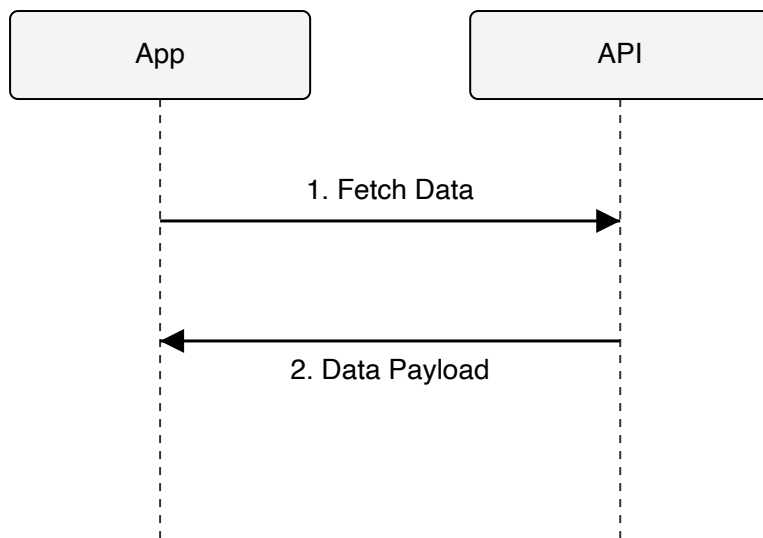
Controller<--] : Ack
deactivate Controller
[<- Controller: 200 OK
deactivate Controller
@enduml
```



Text Alignment & Placement

Response Message Below Arrow

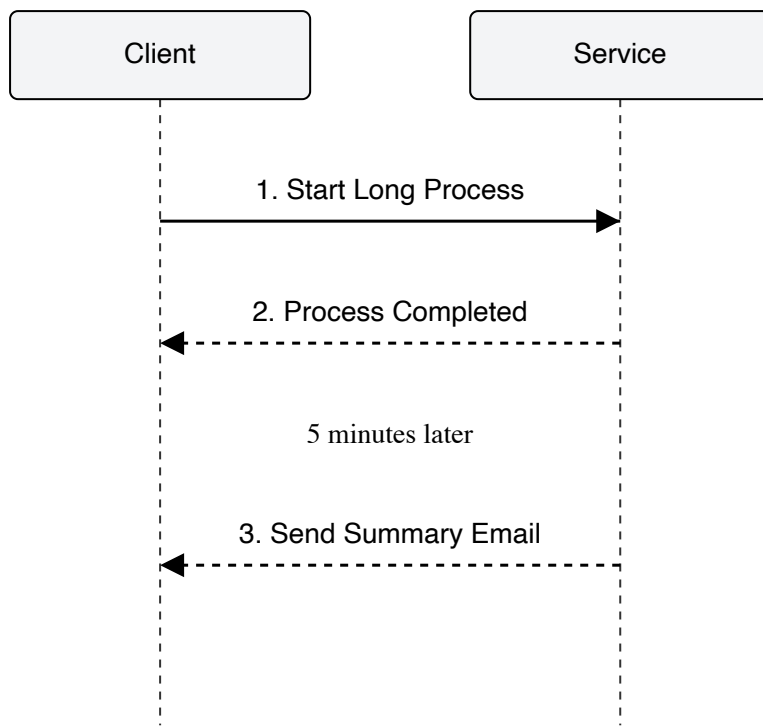
```
@startuml
skinparam responseMessageBelowArrow true
App -> API : Fetch Data
App <- API : Data Payload
@enduml
```



Delay

You can add a delay in the timeline using `...` syntax. This creates a vertical gap with optional centered text.

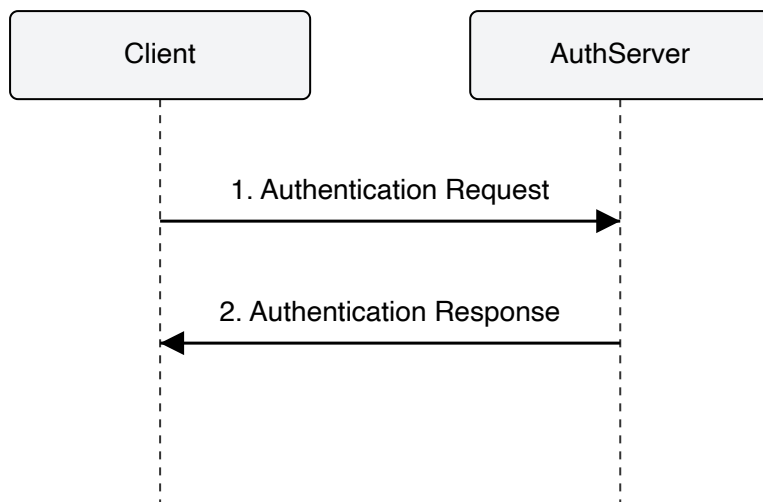
```
@startuml
Client -> Service: Start Long Process
...
Service --> Client: Process Completed
...5 minutes later...
Service --> Client: Send Summary Email
@enduml
```



Message Sequence Numbering

The keyword **autonumber** is used to automatically add an incrementing number to messages.

```
@startuml
autonumber
Client -> AuthServer : Authentication Request
Client <- AuthServer : Authentication Response
@enduml
```

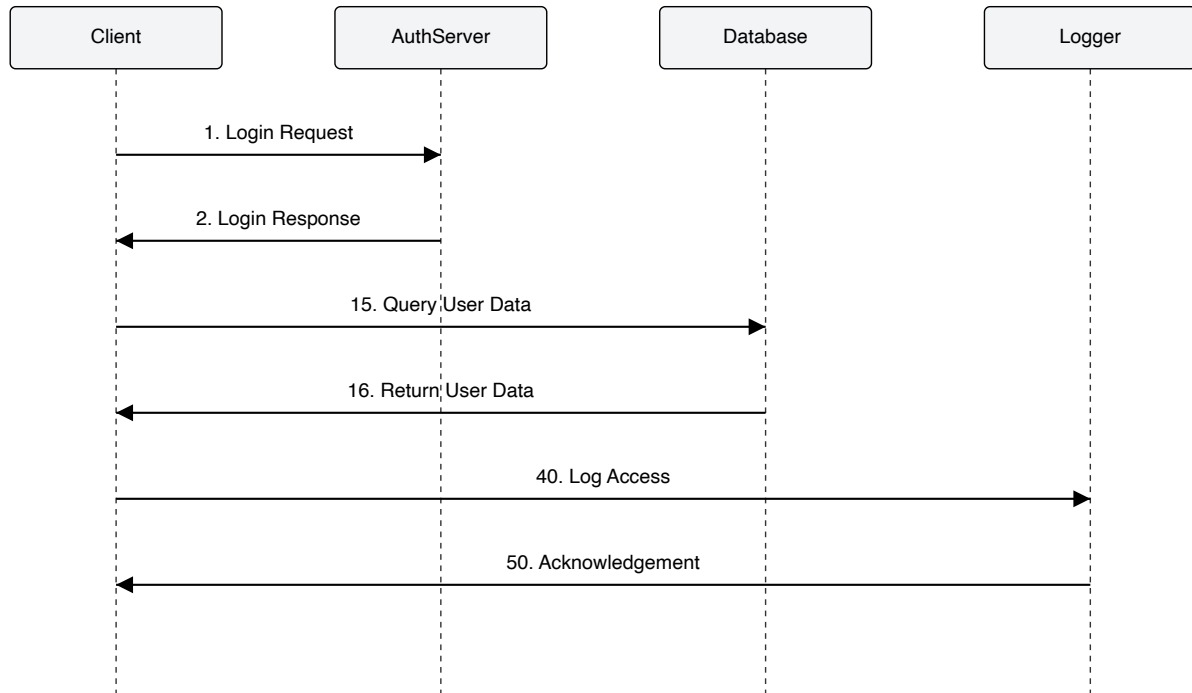


You can specify a start number with `autonumber <start>` , and also an increment with `autonumber <start> <increment>` .

```
@startuml
autonumber
Client -> AuthServer : Login Request
Client <- AuthServer : Login Response

autonumber 15
Client -> Database : Query User Data
Client <- Database : Return User Data

autonumber 40 10
Client -> Logger : Log Access
Client <- Logger : Acknowledgement
@enduml
```

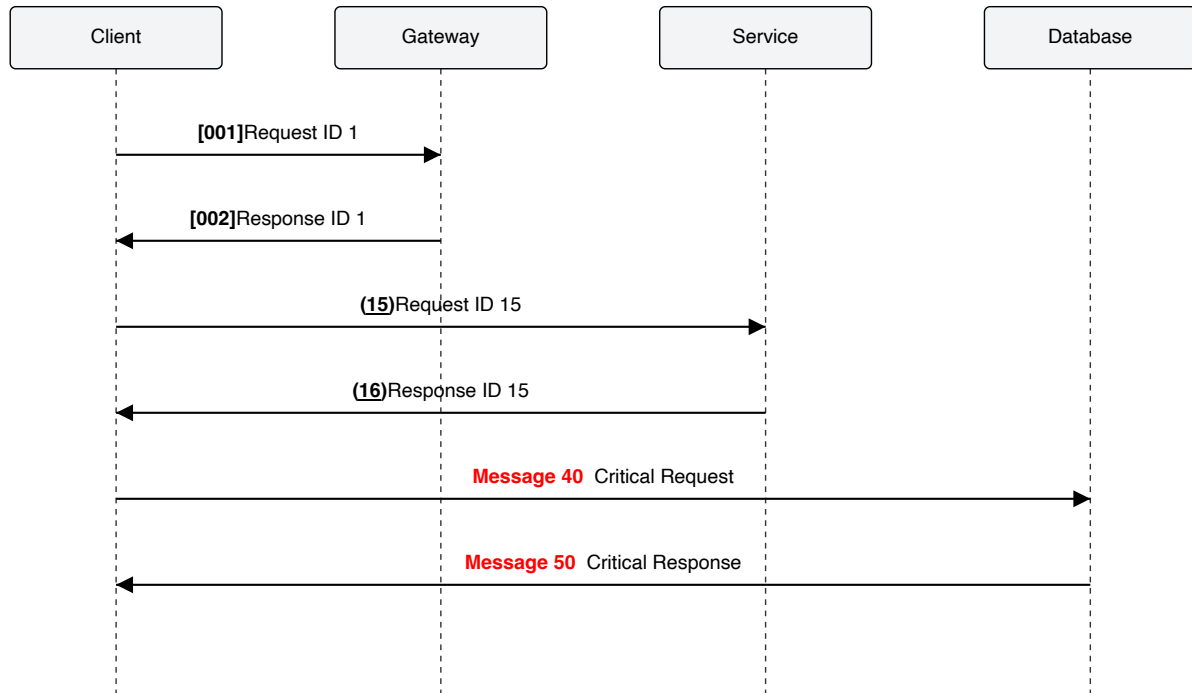
You can specify a format for your number by using double-quotes. The formatting is done with the Java class `DecimalFormat` (`0` means digit, `#` means digit and zero if absent). You can use some HTML tags in the format.

```

@startuml
autonumber "<b>[000]"
Client -> Gateway : Request ID 1
Client <- Gateway : Response ID 1

autonumber 15 "<b>(<u>#</u>)"
Client -> Service : Request ID 15
Client <- Service : Response ID 15

autonumber 40 10 "<font color=red><b>Message 0  "
Client -> Database : Critical Request
Client <- Database : Critical Response
@enduml
  
```



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

```

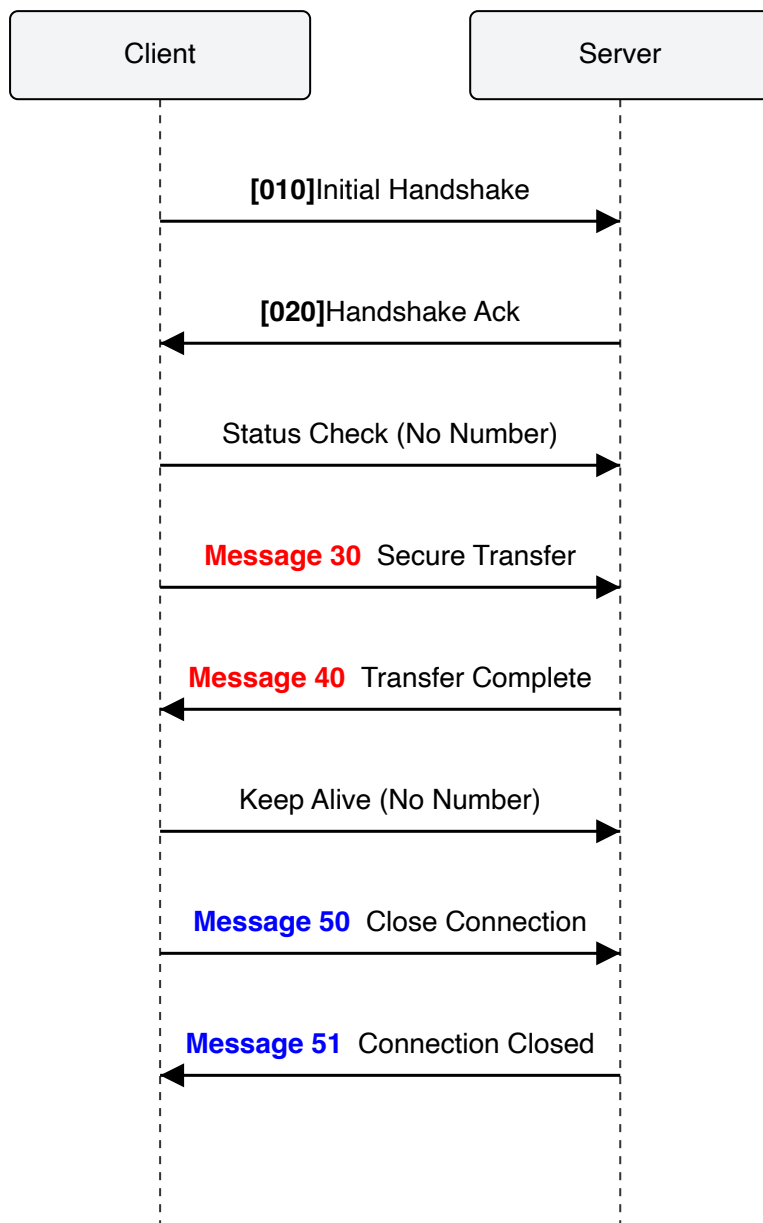
@startuml
autonumber 10 10 "<b>[000]</b>"
Client -> Server : Initial Handshake
Client <- Server : Handshake Ack

autonumber stop
Client -> Server : Status Check (No Number)

autonumber resume "<font color=red><b>Message 0</b></font> "
Client -> Server : Secure Transfer
Client <- Server : Transfer Complete

autonumber stop
Client -> Server : Keep Alive (No Number)

autonumber resume 1 "<font color=blue><b>Message 0</b></font> "
Client -> Server : Close Connection
Client <- Server : Connection Closed
@enduml
  
```



Your start number can also be a 2 or 3 digit sequence using a field delimiter such as ., ;, ,, : or a mix of these. For example: **1.1.1** or **1.1:1**. Automatically the last digit will increment. To increment the first digit, use: **autonumber inc A**. To increment the second digit, use: **autonumber inc B**.

@startuml

autonumber 1.1.1

Client -> Server: Initialize Session

Server --> Client: Session ID

autonumber inc A

'Now we have 2.1.1

Client -> Server: Request Data Package 1

Server --> Client: Data Package 1

autonumber inc B

'Now we have 2.2.1

Client -> Server: Request Data Package 2

Server --> Client: Data Package 2

autonumber inc A

'Now we have 3.1.1

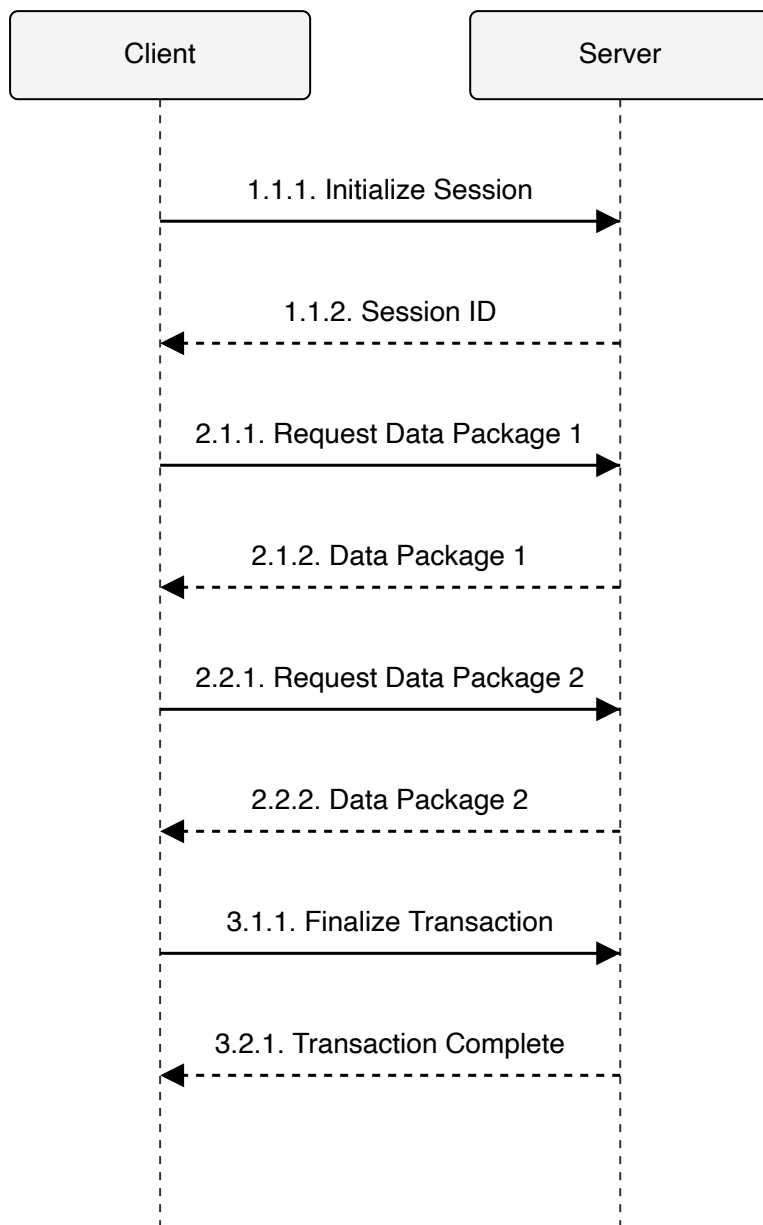
Client -> Server: Finalize Transaction

autonumber inc B

'Now we have 3.2.1

Server --> Client: Transaction Complete

@enduml



Notes & Fragments

Group messages, add notes, and use fragments like loop/alt/opt.

Comprehensive Notes Example

You can place notes relative to the timeline (**left** , **right**), over specific participants (**over**), or across the entire diagram (**across**). You can also apply specific background colors to notes using **#ColorName** .

@startuml

title Notes

User -> PaymentGateway: Submit Payment

note left: Note left text

note right #Gold: Note right text

note over User #Aqua

Note Over

end note

note across #Bisque: Note Accross

note right of PaymentGateway #SandyBrown: note right of

note left of PaymentGateway #YellowGreen: note left of

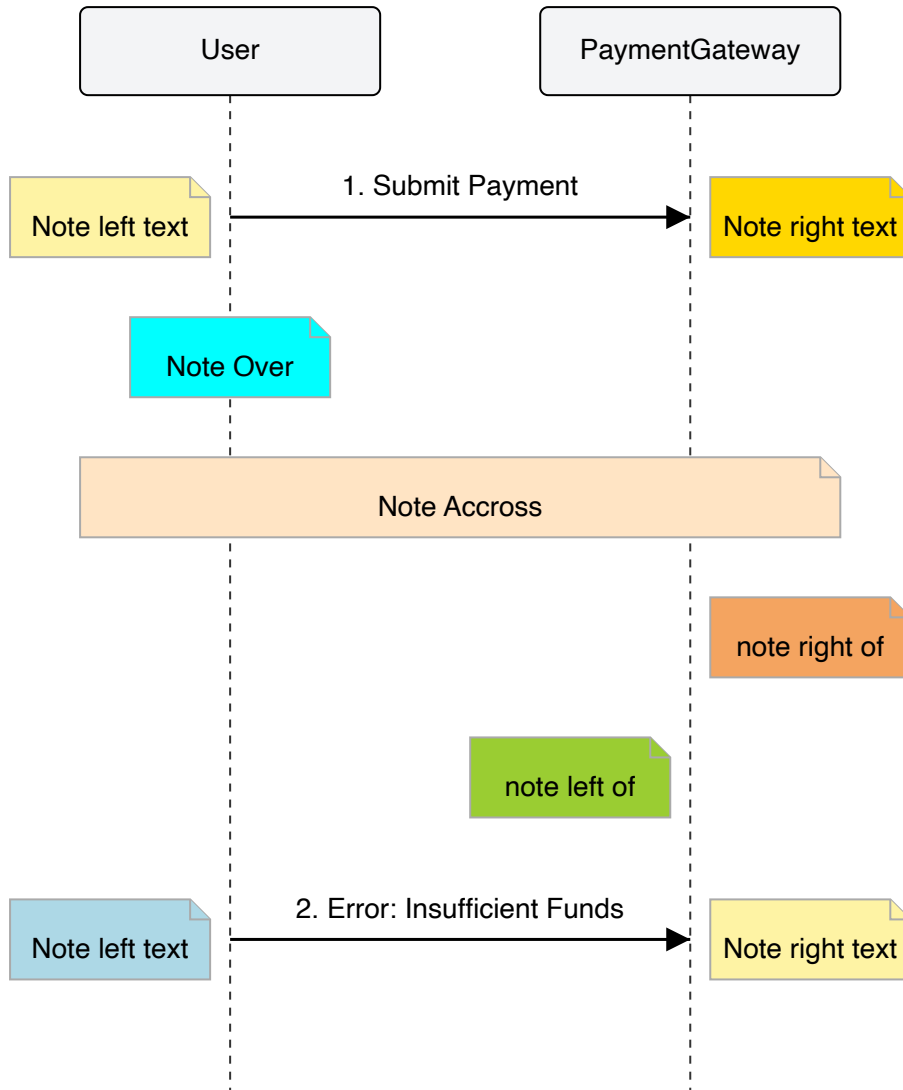
User -> PaymentGateway: Error: Insufficient Funds

note right: Note right text

note left #LightBlue: Note left text

@enduml

Notes



Activation Coloring

You can color activation boxes to distinguish different types of processing or highlight specific interactions.

@startuml

participant User

participant "Payment System" as Payment

participant "Risk Engine" as Risk

participant "Bank" as Bank

User -> Payment: **Process Payment**

activate Payment #LightBlue

Payment -> Risk: **Analyze Risk**

activate Risk #Salmon

Risk --> Payment: **Approved**

deactivate Risk

Payment -> Bank: **Charge Credit Card**

activate Bank #LightGreen

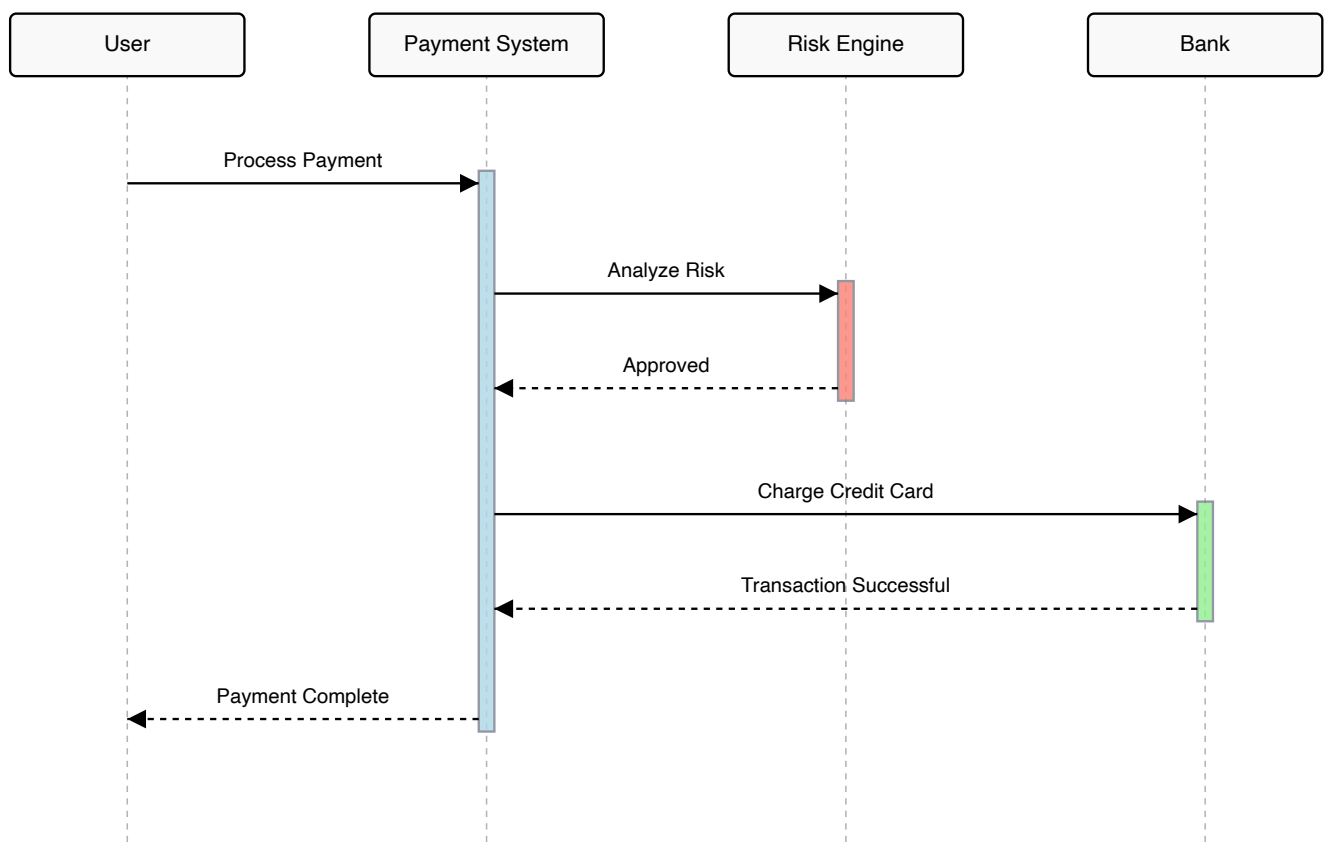
Bank --> Payment: **Transaction Successful**

deactivate Bank

Payment --> User: **Payment Complete**

deactivate Payment

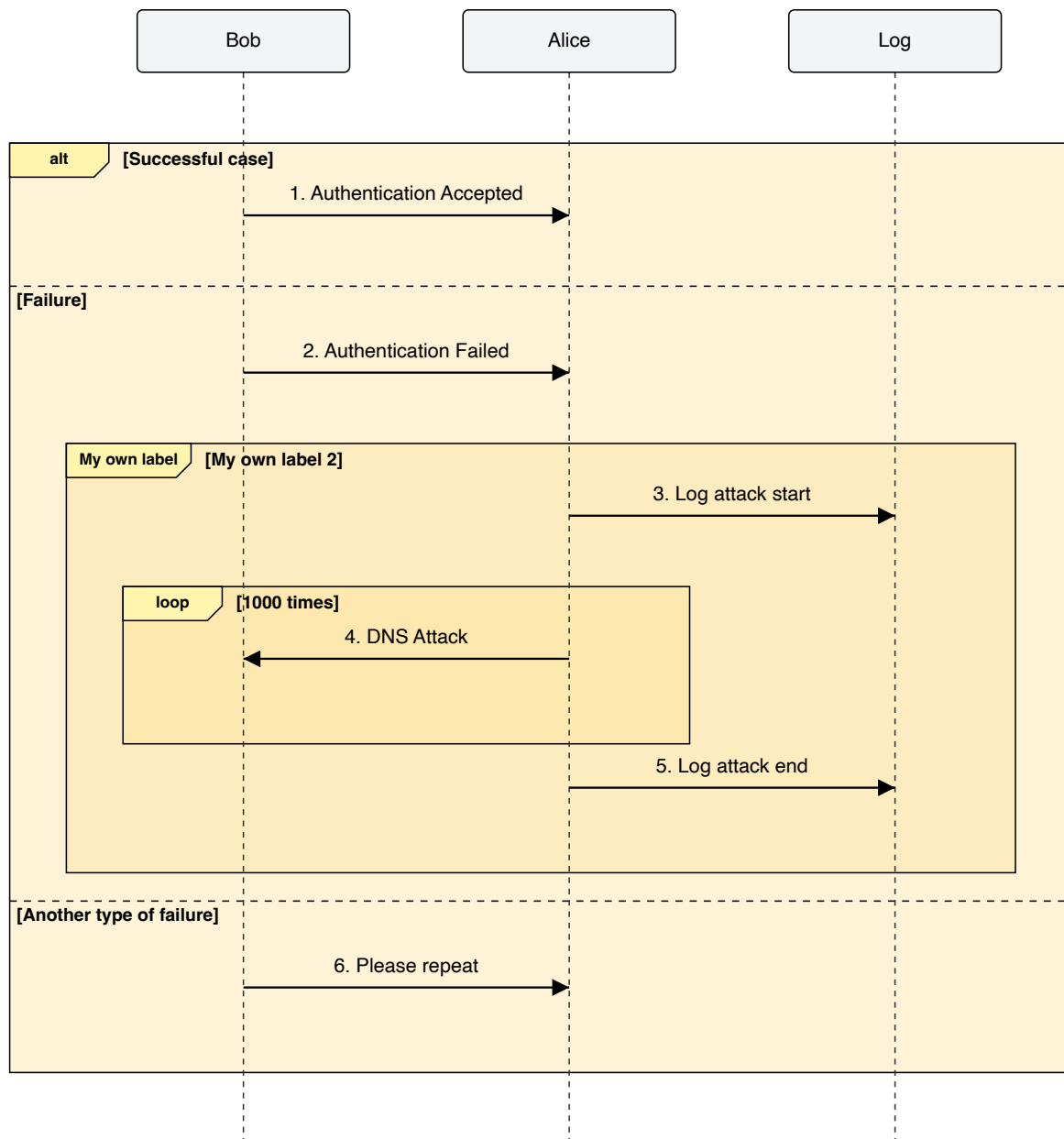
@enduml



Grouping Message

Group messages using keywords like `alt/else`, `opt`, `loop`, `par`, `break`, `critical`, and `group`.

```
@startuml
alt Successful case
    Bob -> Alice: Authentication Accepted
else Failure
    Bob -> Alice: Authentication Failed
    group My own label [My own label 2]
        Alice -> Log : Log attack start
        loop 1000 times
            Alice -> Bob: DNS Attack
        end
        Alice -> Log : Log attack end
    end
else Another type of failure
    Bob -> Alice: Please repeat
end
@enduml
```



Group Coloring

You can add colors to the group header, body, and even specific `else` or `option` blocks.

Syntax

- `type#HeaderColor #BodyColor Label`
- `group#HeaderColor #BodyColor Label [Description]`
- `else #BgColor Label`

@startuml

critical #Gold #LightBlue Critical Transaction

User -> PaymentGateway: **Submit Payment**

else #Pink Payment Failed

User -> PaymentGateway: Error: **Insufficient Funds**

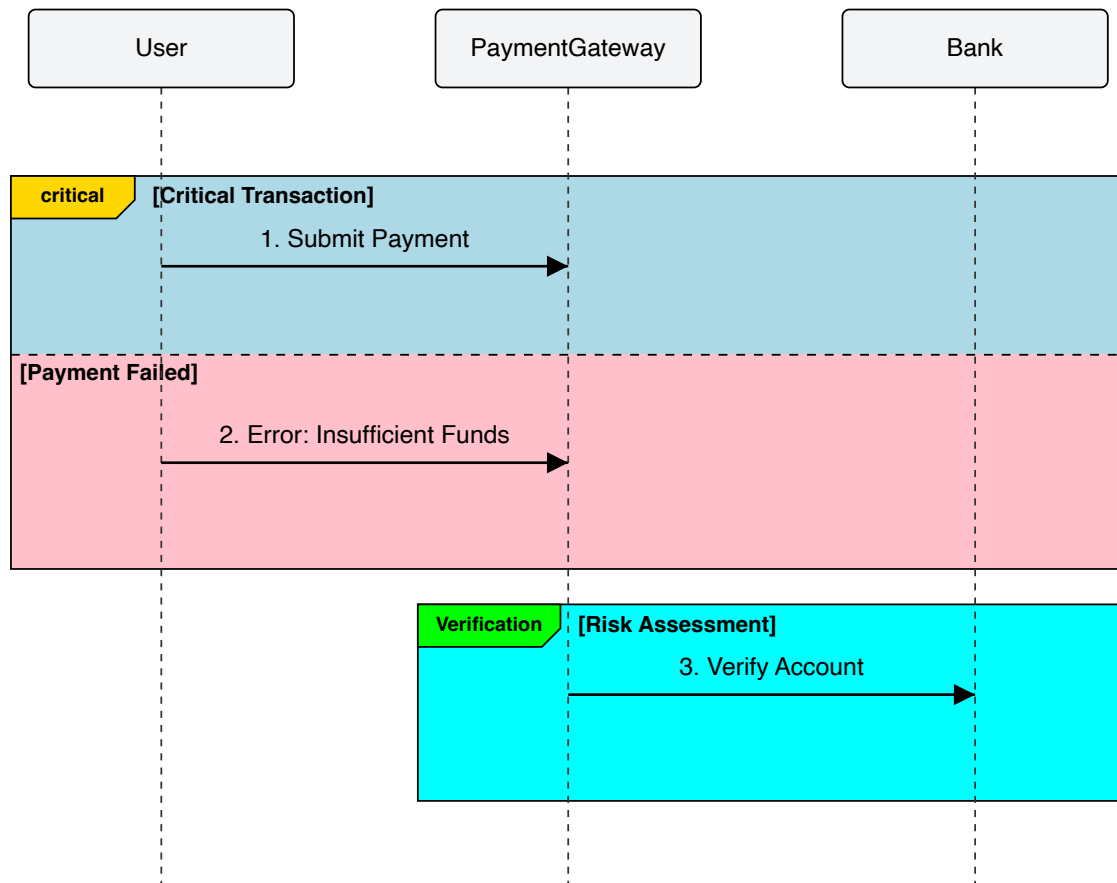
end

group #Lime #Cyan Verification [Risk Assessment]

PaymentGateway -> Bank: **Verify Account**

end

@enduml



Complex Alt/Else Coloring

You can colorize complex decision trees:

@startuml

alt #Gold #LightBlue Successful Login

User -> Auth: Credentials

Auth -> User: Token

else #Pink Invalid Credentials

User -> Auth: Credentials

Auth -> User: Error 401

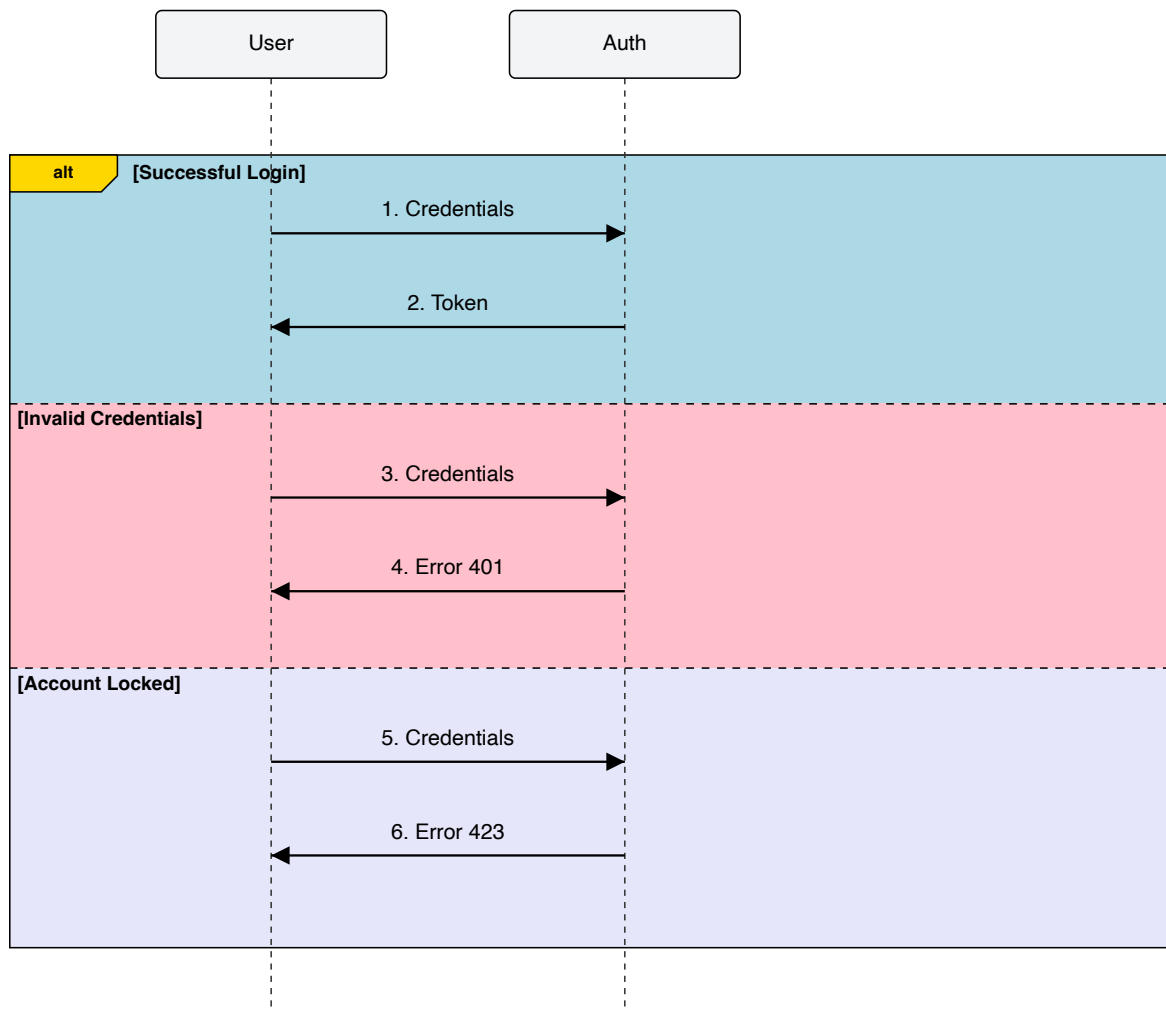
else #Lavender Account Locked

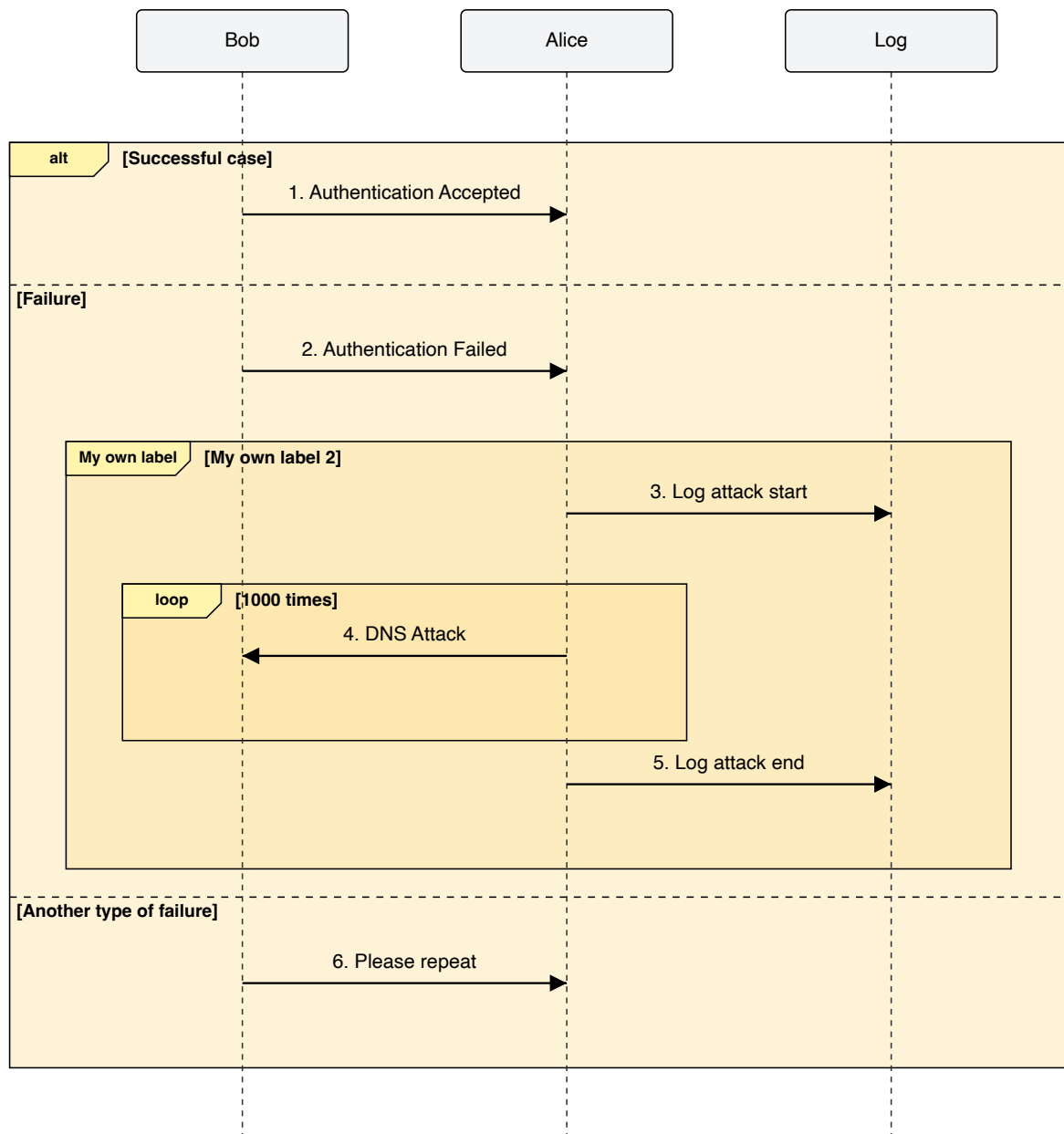
User -> Auth: Credentials

Auth -> User: Error 423

end

@enduml





Notes on Participants (Relative)

Place notes relative to participants using **left of** , **right of** , or **over** .

@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

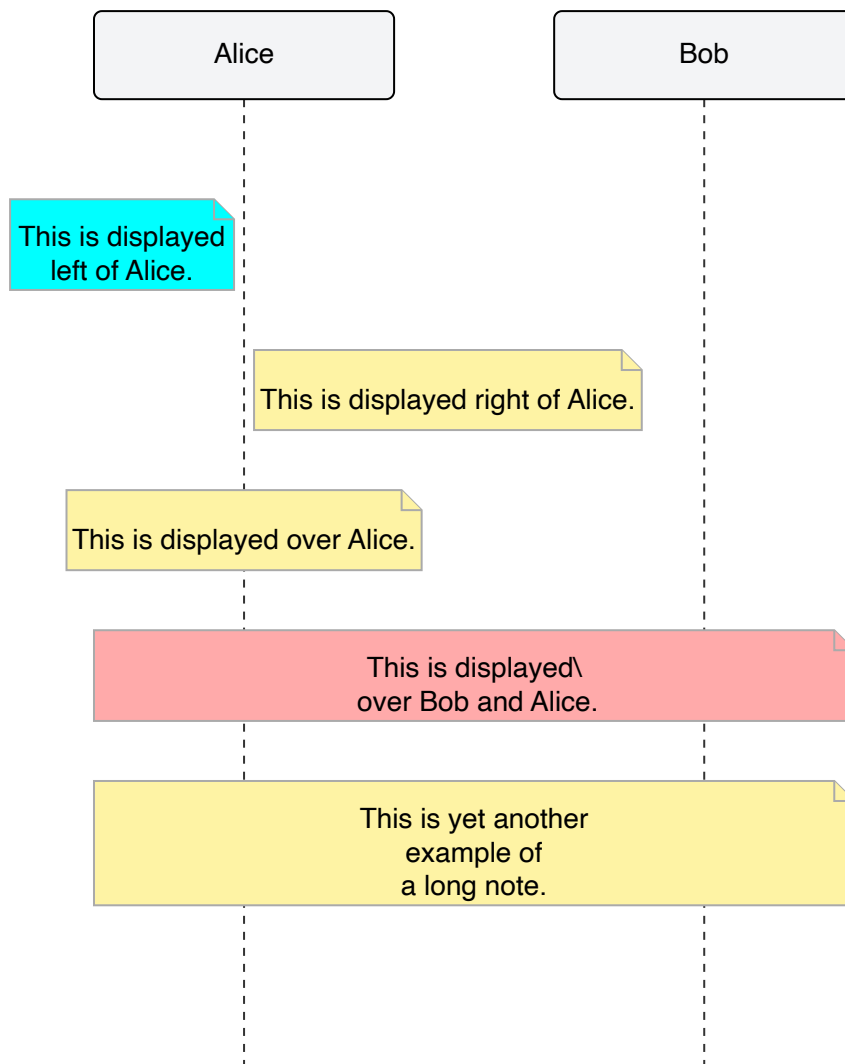
This is yet another

example of

a long note.

end note

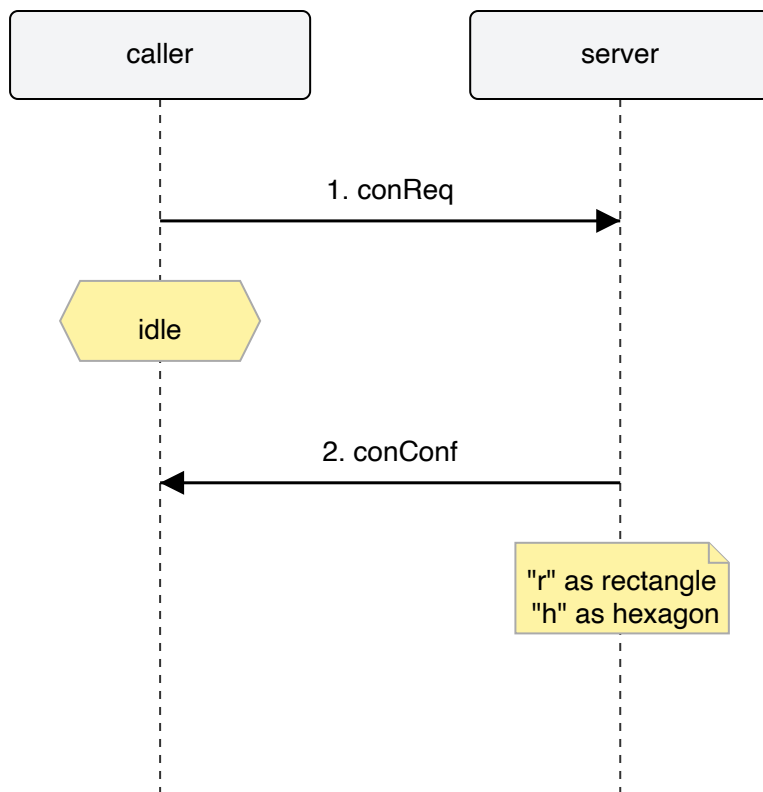
@enduml



Formatting Notes

You can shape notes using keywords like **hnote** (hexagon) and **rnote** (rectangle).

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



Divider or Separator

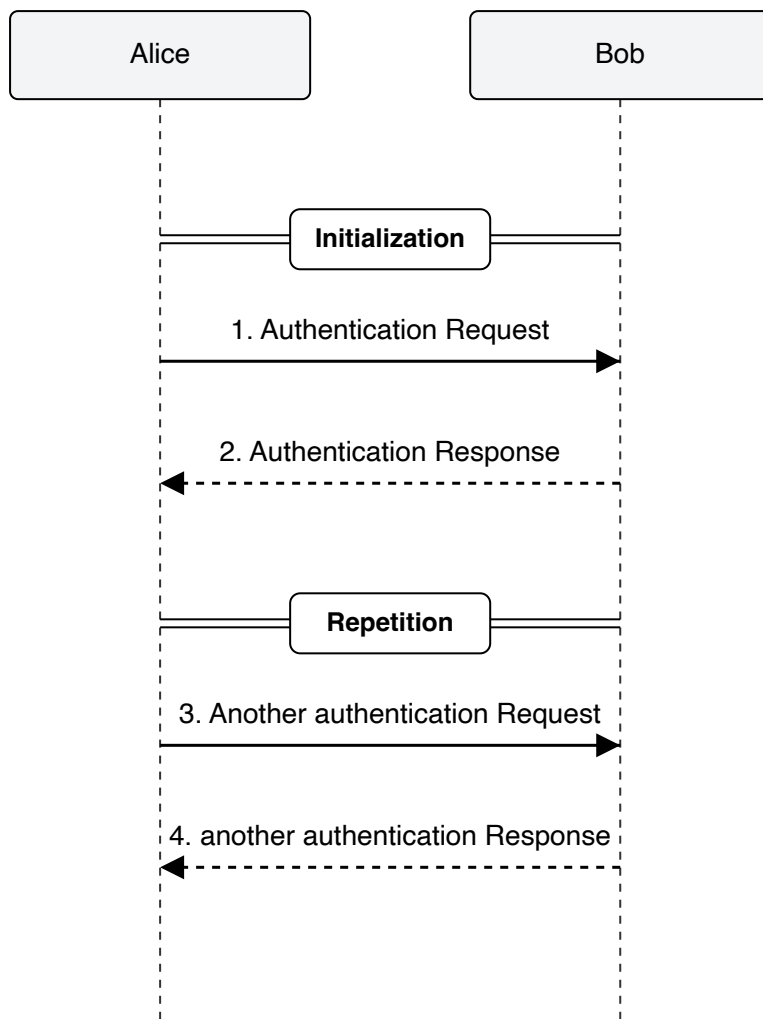
Split diagrams into logical phases using `== Separator ==`.

```

@startuml
== Initialization ==
Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response

== Repetition ==
Alice -> Bob: Another authentication Request
Bob --> Alice: another authentication Response
@enduml

```

Reference

Link to other parts of the diagram or external interactions using **ref over** .

@startuml

participant Alice

actor Bob

ref over Alice, Bob : init

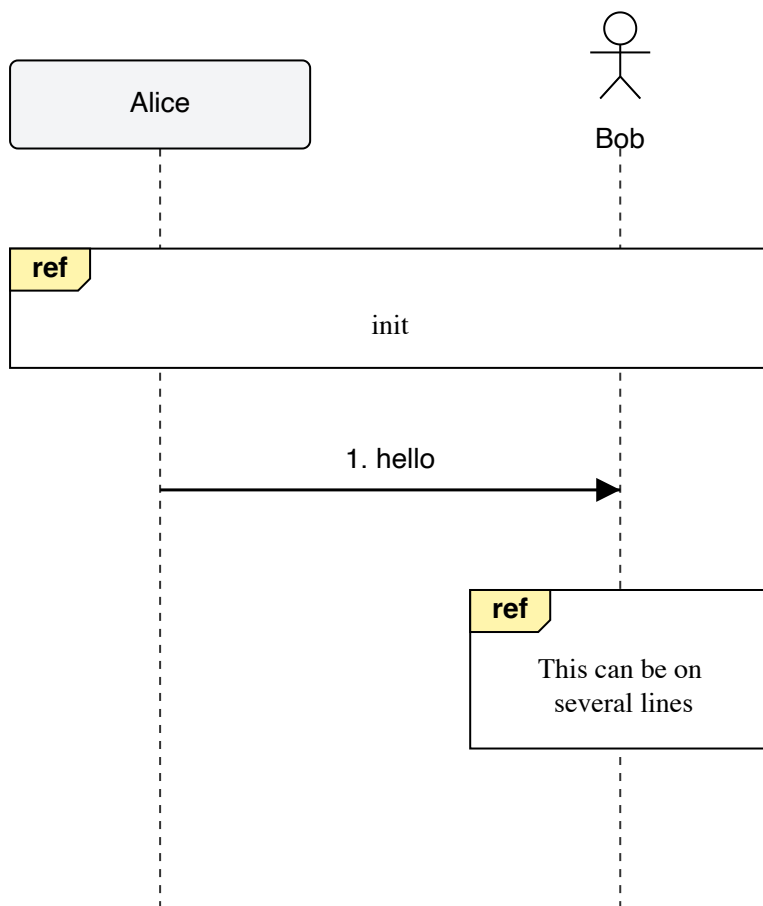
Alice -> Bob : hello

ref over Bob

This can be on
several lines

end ref

@enduml



Title, Header and Footer, Caption

Add metadata to your diagram. Keywords: **title**, **header**, **footer**, **caption**.

@startuml

header Page Header

footer Page %page% of %lastpage%

title

<u>Simple</u> communication example

on <i>several</i> lines and using html

end title

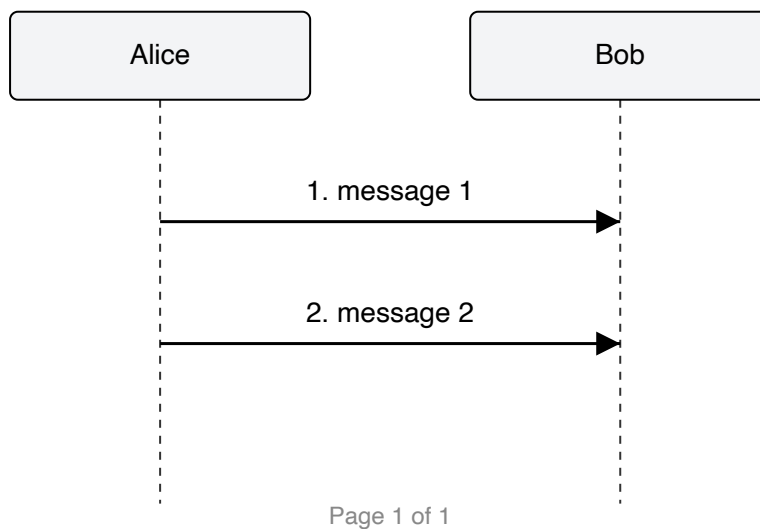
Alice -> Bob : message 1

Alice -> Bob : message 2

@enduml

Page Header

Simple communication example on *several* lines and using **html**



Styling & Skinparams

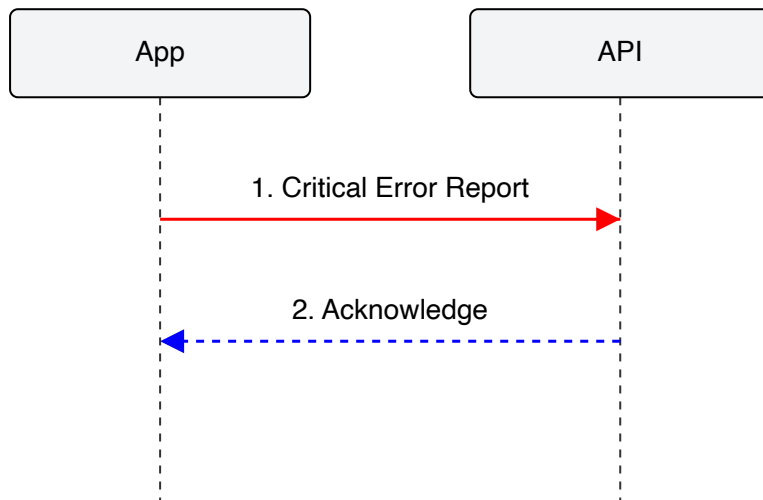
Change Arrow Color (Inline)

```
@startuml
```

```
App -[#red]> API : Critical Error Report
```

```
API -[#0000FF]-> App : Acknowledge
```

```
@enduml
```



Styling & Skinparams

You can customize the appearance of diagrams using **skinparam** commands. Use specific parameters to style participants, actors, notes, and more.

@startuml

skinparam sequenceArrowThickness 2

skinparam maxMessageSize 100

skinparam participant {
 BorderColor #2C3E50
 BackgroundColor #ECF0F1
 FontColor #2C3E50
}

skinparam actor {
 BorderColor #D35400
 BackgroundColor #E67E22
 FontColor #D35400
 FontSize 14
}

skinparam note {
 BackgroundColor #F1C40F
 BorderColor #D4AC0D
 FontColor black
}

actor User

participant "Service A" as A

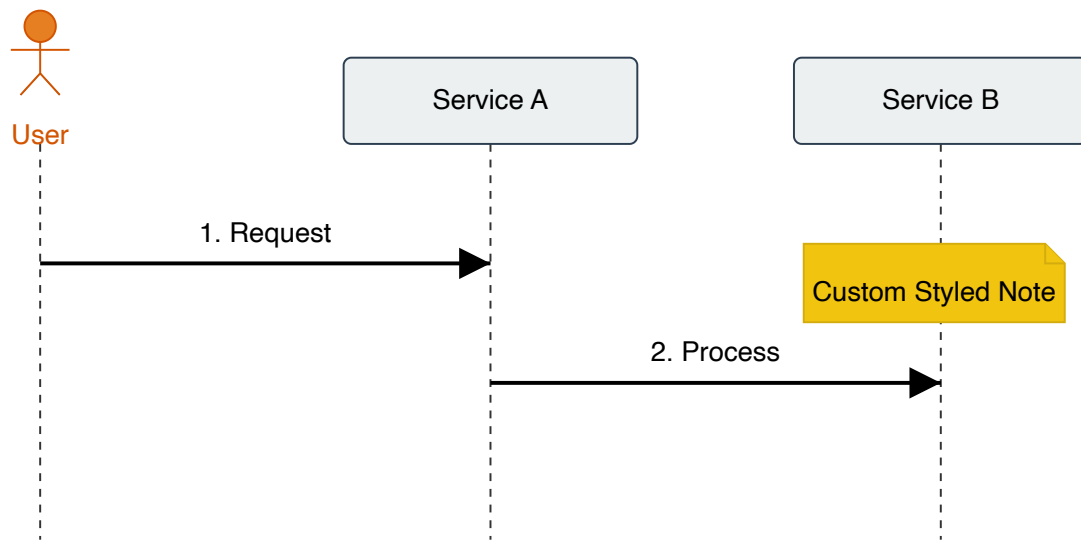
participant "Service B" as B

User -> A: Request

note right: Custom Styled Note

A -> B: Process

@enduml



Common Skinparams

Global:

- `sequenceArrowThickness`
- `roundCorner`
- `maxMessageSize`
- `monochrome true/false`

Participants & Actors:

- `participantBorderColor`
- `participantBackgroundColor`
- `actorBorderColor`
- `actorBackgroundColor`

Message & Line Styling

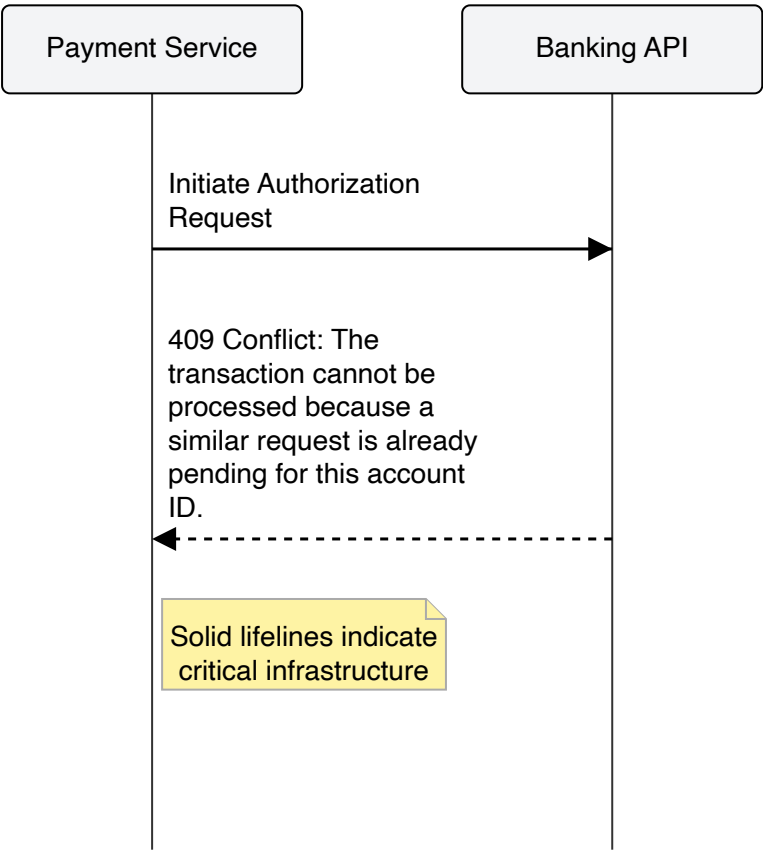
You can control message text wrapping, alignment, and lifeline style using specific skinparams.

```
@startuml
skinparam maxMessageSize 150
skinparam sequenceMessageAlign left
skinparam lifelineStrategy solid

participant "Payment Service" as Pay
participant "Banking API" as Bank

Pay -> Bank : Initiate Authorization Request
Bank --> Pay : 409 Conflict: The transaction cannot be processed because a similar request is already pending for this account ID.

note right of Pay: Solid lifelines indicate critical infrastructure
@enduml
```



Parameter	Syntax	Description
Max Message Size	<code>skinparam maxMessageSize <size></code>	Wraps message text if it exceeds the specified width in pixels.

Parameter	Syntax	Description
Message Alignment	<code>skinparam sequenceMessageAlign left right center</code>	Aligns message text relative to the arrow.
Lifeline Strategy	<code>skinparam lifelineStrategy solid nosolid</code>	Sets participant lifelines to be solid or dashed (nosolid).

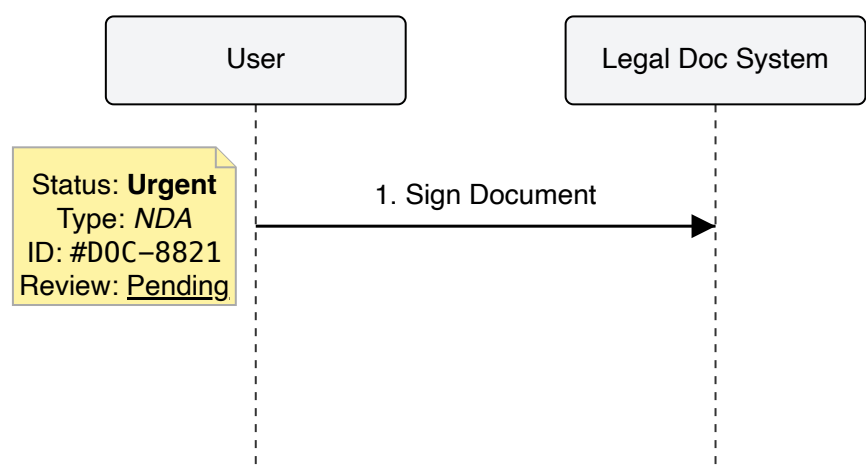
Creole and HTML

You can use Creole formatting for bold, italic, and more.

```
@startuml
participant User
participant "Legal Doc System" as Legal

User -> Legal : Sign Document

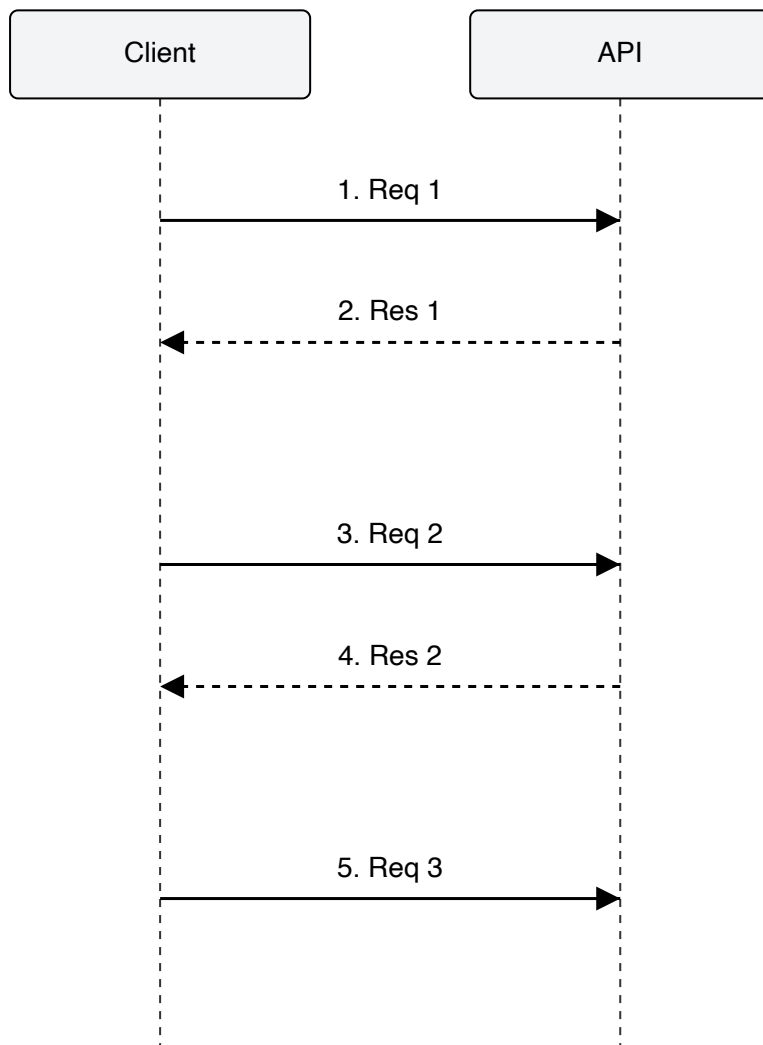
note left
    Status: Urgent
    Type: NDA
    ID: "#DOC-8821"
    Review: Pending
end note
@enduml
```



Space

Use `|||` for spacing.

```
@startuml
Client -> API: Req 1
API --> Client: Res 1
|||
Client -> API: Req 2
API --> Client: Res 2
||45||
Client -> API: Req 3
@enduml
```



Mermaid Support

The app has limited support to mermaid sequence diagram format. The code has to start with "sequenceDiagram" to get the mermaid parser.

For detailed Mermaid sequence diagram syntax and examples, please refer to the Mermaid Sequence Diagrams Documentation available in the Help section.

Support

For additional help or questions about SparkChart Pro:

- Website: <https://sparkchart.pro>
 - Email: sparkchartpro@gmail.com
-

Version 1.0

SparkChart Pro - Professional UML Sequence Diagrams for macOS