Use Cases

Version 2.1
March 2023

This section is dedicated for use cases which can be done by different API and initiated from different sources.

Wallet Server LC API Initiated

This section describes use cases which can be initiated using Wallet Server LC API. This API should be used by Customers to manage Users and cards data on Wallet Server.

User with Card Registration

User with Card Registration is process when user and cards are transferred to Wallet Server to make possible use them in different processes (e.g. digitization) later in the application. Registration needs to be done as the first step

@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA

```
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant Issuer
Issuer -> WS: 1. addUserWithCard(userData, cardData)
activate Issuer
activate WS
WS --> Issuer: 2. response(cardId)
deactivate WS
Issuer -> Issuer: 3. storeCardId
deactivate Issuer
@enduml
```

Card Reissuing

Since plastic cards have expiration date, Issuers may want to reissue them. In most cases PAN remains the same and only expiration date is extended. It is worth to mention that MDES does not check PAN expiry date for transactions performed with Payment Token. MDES offers API for Issuers where is possibility to update expiration date or even whole PAN for already provisioned Payment Tokens. In context of integration there are a few options:

- Issuer has own processor which is integrated with Customer Service MDES API and there is a possibility to update PAN or expiration date. Then reissuing is done only on processor side. Wallet Server will be notified by MDES about the change.
- Issuer uses Verestro Token Management Platform and reissuing is possible using LC API.
- Issuer uses only Wallet Server which acts as Token Requestor and there is no possibility to update PAN or expiration date for given Payment Token. Issuer needs to delete previous card and add new one. Users need to make digitization again.

All options needs to be considered individually and discussed with Verestro representative.

User Deletion

During this process all data related to given User are deleted. Payment Tokens are deleted asynchronously.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
```

```
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant Issuer
Issuer -> WS: 1. deleteUser(userId)
activate Issuer
activate WS
WS -> WS: 2. delete cards, devices
WS --> Issuer: 3. response
deactivate Issuer
loop All Payment Tokens for given User
WS -> MDES: 4. delete token
activate MDES
MDES --> WS: 5. response
opt
WS ->> Issuer: 6. send Payment Token event
end
deactivate WS
deactivate MDES
end
@enduml
```

Card Deletion

During card deletion process all data related to given card are deleted. Payment Tokens are deleted asynchronously.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
```

```
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES
Issuer -> WS: 1. deleteCard(cardId)
activate WS
activate Issuer
WS -> WS: 2. deleteCard
WS --> Issuer: 3. response
deactivate Issuer
loop All Payment Tokens for card
WS -> MDES: 4. Delete token
activate MDES
MDES -> MDES: 5. Delete token mapping
MDES --> WS: 6. response
opt
WS ->> Issuer: 7. send Payment Token event
end
deactivate MDES
WS -> SDK: 8. deliverMessage(paymentTokenDelete)
NOTE LEFT: See: Handle Message From Server
deactivate WS
deactivate MPA
activate SDK
alt Request session if required
SDK -> MDES: 9. request session
activate MDES
MDES --> SDK: 10. response
MDES -> WS: 11. sendRemoteNotificationMessage
deactivate MDES
activate WS
WS -> SDK: 12. deliverMessage(mdesRemoteMessage)
NOTE LEFT: See: Handle Message From Server
deactivate WS
```

deactivate MPA
end
SDK -> MDES: 13. delete(tokenUniqueReference)
activate MDES
MDES --> SDK: 14. response
deactivate MDES
SDK -> SDK: 15. Delete transaction credentials, card profile
SDK -> MPA: 16. onPaymentInstrumentStatusChanged(id, status)
deactivate SDK
end
deactivate MPA
@enduml

Wallet SDK Initiated

Wallet SDK Setup

Setup of Wallet SDK (both modules VCP SDK and MDC SDK) is main step which needs to be made at very beginning. During setup main configuration should be provided. Moreover there is some configuration which is related with HCE payments: MPA should be registered as default application for payment (Tap & Pay) and also should implement HostApduService to emulate an NFC card inside an Android service component. Please find more details in *Wallet SDK API* document.

@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
participant MPA

participant "Wallet SDK" as SDK
MPA -> SDK: 1. MDC:setup(configuration)
activate MPA
activate SDK
SDK --> MPA: 2. result
MPA -> SDK: 3. UCP:setup(configuration)
SDK --> MPA: 4. result
deactivate MPA
deactivate SDK
@enduml

Pair Device on Wallet Server

This section describes pairing device process. Device pairing is process which authenticates device in context of given user. During this process device data and keys used in communication are exchanged with Wallet Server. To make possible device pairing, user needs to be already registered on the Wallet Server. Every device is identified by unique identifier. After every pairing device request, Wallet Server gives unique installation identifier. It means that particular installation of the application installed on particular device belongs to given User. Different Users can use same device for separate installations. If any active installation on given device already exists during pairing device, Wallet Server will delete and create new installation in context of new user. Only one active installation is possible on particular device. Device pairing is the first process which should be done after user registration. Pairing device should be done only once for individual installation.

Pair Device By Trusted Identity

In the integrated model, Trusted Identity is used to proof User authenticity. User is firstly authenticated on the Customer side. Trusted Identity should be generated on Issuer backend side and pass via Wallet SDK, since mobile environment is treated as unsecure. Algorithm of generating Trusted Identity is placed in Wallet SDK API specification.

Access to User data stored on Wallet Server is possible only when session is established. After paring device session is automatically generated for particular User.

If previously on given device was installation which had Payment Tokens, during pairing these Payment Tokens are deleted asynchronously.

@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {

```
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant "Issuer" as Issuer
participant "MDES" as MDES
MPA->Issuer: 1. authenticate
activate MPA
activate Issuer
Issuer->Issuer: 2. generateTrustedIdentity - signed user id
Issuer--> MPA: 3. response(trustedIdentity)
deactivate Issuer
MPA -> MPA: 4. obtainRNSToken(walletFirebase)
MPA-> SDK: 5. MDC:pairDeviceByTrustedIdentity\r(trustedIdentity, rnsRegistrationToken)
activate SDK
SDK -> SDK: 6. isDevicePaired
alt isDevicePaired=true
SDK --> MPA: 7. result
else isDevicePaired=false
SDK-> WS: 8. pairDeviceByTrustedIdentity\r(trustedIdentity, rnsRegistrationToken, deviceInfo)
activate WS
WS-> WS: 9. verify trusted identity
alt isActiveInstallationOnGivenDevice
WS -> WS: 10. deleteActiveInstallation
WS -> MDES: 11. deleteDeviceTokens
activate MDES
deactivate MDES
note left: asynchronous
end
WS -> WS: 12. createNewDeviceInstallationRecordForUser
WS --> SDK: 13. response\r (userSessionToken, installationId)
deactivate WS
SDK -> SDK: 14. store userSessionToken
SDK-->MPA: 15. result
```

deactivate MPA
deactivate SDK
end
@enduml

Card Digitization

Card digitization is process which allows to transform plastic card into Payment Token. To perform digitization, card data should be placed already on Wallet Server and device should be already paired. Card digitization process uses Wallet Server identifier of the card. There are two ways of performing card digitization. Usage of the API's depends on needs in given implementation.

Card Digitization Ways

Depending on implementation card can be digitized in different way using different approach. There are following ways of card digitization:

- one step - which is the simplest way of card digitization and should be used in project implementation where card can be just digitized without any additional staff like additional User authentication or showing T&C or showing card art from MDES,
- multi step - which should be used when digitization will require additional User authentication, showing T&C or showing card art from MDES.

Card Digitization Ways - One Step

One step digitization is dedicated for implementations where there is no additional User authentication and there is no need to show terms and conditions before every digitization or show card art from MDES. Mostly it should be used in Issuer applications where NFC payments is added as a new functionality without additional staff which is used in dedicated wallets. Only transaction outcomes APPROVED(see APPROVED) or DECLINE(see DECLINE) are expected in this type of digitization.

@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F

```
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant "MPA" as MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES as MDES
participant Issuer
MPA -> SDK: 1. UCP:Cards#digitize\r(paymentInstrumentId=verestroCardId, userLanguageCode,
securityCode?)
activate MPA
activate SDK
SDK -> SDK: 2. isDeviceRegisteredForPayment
alt isDeviceRegisteredForPayment=false
SDK -> WS: 3. getPkCertificate
activate WS
WS -> MDES: 4. getPkCertificate
activate MDES
MDES --> WS: 5. response(pkCertificate)
WS --> SDK: 6. response(pkCertificate)
SDK -> SDK: 7. prepareDataForRegistration(pkCertificate)
SDK -> WS: 8. registerDeviceForPayment\r (paymentDeviceInfo, userSessionToken)
WS -> MDES: 9. registerMobilePaymentApplication\r (paymentDeviceInfo)
MDES --> WS: 10. response(mobileKeys,\r remoteManagementUrl)
WS --> SDK: 11. response(mobileKeys,\r remoteManagementUrl)
end
SDK -> SDK: 12. isPaymentInstrumentDigitized
alt isPaymentInstrumentDigitized=true
SDK --> MPA: 13. result
else isPaymentInstrumentDigitized=false
SDK -> WS: 14. digitizeCard\r (cardId, userSessionToken)
WS -> MDES: 15. checkEligibility(cardData, paymentAppId,\r paymentAppInstanceId)
MDES --> WS: 16. response (eligibilityReceipt)
WS -> MDES: 17. digitize(eligibilityReceipt)
MDES --> WS: 18. response
deactivate MDES
WS --> SDK: 19. response(paymentTokenInfo)
deactivate WS
SDK --> MPA: 20. result
deactivate SDK
deactivate MPA
end
```

deactivate SDK
deactivate MPA
note over SDK, WS #1C1E3F: See Card digitization outcome Approved or Decline diagram
@enduml

Card Digitization Ways - Multi step

Multi step digitization should be used in implementations where terms and conditions are displayed before every digitization, additional user authentication is needed or card art need to be used from MDES. Multi step digitization consists of following steps:

- checking card eligibility
- showing T&C
- digitizing card

Digitization may finished with different outcomes(see Card Digitization Outcomes).

@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES
participant Issuer
MPA -> SDK: 1. UCP:Cards#checkEligibility(paymentInstrumentId=verestroCardId, locale)
activate MPA
activate SDK

SDK -> SDK: 2. isDeviceRegisteredForPayment
alt isDeviceRegisteredForPayment=false
SDK -> WS: 3. getPkCertificate
activate WS
WS -> MDES: 4. getPkCertificate
activate MDES
MDES --> WS: 5. response(pkCertificate)
WS --> SDK: 6. response(pkCertificate)
SDK -> SDK: 7. prepareDataForRegistration(pkCertificate)
SDK -> WS: 8. registerDeviceForPayment\r (paymentDeviceInfo, userSessionToken)
WS -> MDES: 9. registerMobilePaymentApplication\r (paymentDeviceInfo)
MDES --> WS: 10. response(mobileKeys,\r remoteManagementUrl)
WS --> SDK: 11. response(mobileKeys,\r remoteManagementUrl)
end
SDK -> WS: 12. checkEligibility(paymentInstrumentId,userSessionToken)
WS -> MDES: 13. checkEligibility(card)
MDES -->WS: 14. response(termsAndConditionsAssetId, eligibilityReceipt)
WS --> SDK: 15. response(termsAndConditionsAssetId, digitizationRef)
SDK --> MPA: 16. result(termsAndConditionsAssetId)
MPA -> SDK: 17. UCP:getAsset(termsAndConditionsAssetId)
SDK -> WS: 18. getAsset(termsAndConditionsAssetId)
WS -> MDES: 19. getAsset(termsAndConditionsAssetId)
MDES --> WS: 20. response(content)
deactivate MDES
WS --> SDK: 21. response(content)
deactivate WS
SDK --> MPA: 22. result
deactivate SDK
MPA -> User: 23. show T&C
deactivate MPA
User -> MPA: 24. accept
activate MPA
MPA -> SDK: 25. UCP:Cards#digitize(paymentInstrumentId, cvc?)
activate SDK
SDK -> WS: 26. digitize(digitizationRef, cvc?, userSessionToken)
activate WS
WS ->MDES: 27. digitize(eligibilityReceipt, cvc?)
activate MDES
MDES --> WS: 28. response(additionalAuthenticationRequired, authenticationMethods?, tokenInfo, productConfig)
deactivate MDES
WS --> SDK: 29. response(additionalAuthenticationRequired, authenticationMethods?, tokenInfo, productConfig)
deactivate WS
SDK --> MPA: 30. result
deactivate SDK

MPA -> User: 31. please wait
deactivate MPA
note over SDK, WS #1C1E3F: See Card digitization outcome Approved or Decline or Require Additional Authentication diagram
@enduml

Card Digitazation Outcomes

There are following digitization outcomes which should be handled differently:

- APPROVED
- DECLINED
- REQUIRE_ADDITIONAL_AUTHENTICATION

APPROVED

This digitization outcome always refers to green path digitization decision. When digitization is APPROVED then profile provisioning(See Profile Provisioning) starts automatically. According to MDES architecture just after digitization Payment Token is in INACTIVE state even though does not require additional User authentication. Due that fact new flag was introduced which informs which Payment Token exactly needs to be additionally authenticated. After successful provisioning token is activated and then SDK will perform Transaction Credentials - Initial Replenishment.

@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK

```
participant "Wallet Server" as WS
participant MDES as MDES
participant Issuer
group APPROVED
MDES --> WS : 1. responseFromDigitization(APPROVED, cardDigitizationData)
activate MDES
activate WS
opt
WS ->>Issuer: 2. send Payment Token Event
end
WS --> SDK : 3. response(digitizatonData, paymentToken)
deactivate WS
activate SDK
SDK --> MPA : 4. result
activate MPA
MPA --> User : 5. result
deactivate MPA
... Profile Provisioning ...
note over MPA, MDES #1C1E3F: See Profile Provisioning diagram
... MDES Token Activation ...
SDK -> MDES: 6. notifyProvisioningResult
MDES -> MDES: 7. createTokenMapping
MDES -> WS: 8. notifyTokenUpdated(Active)
deactivate MDES
activate WS
opt
WS ->>Issuer: 9. send Payment Token Event
end
WS -> SDK: 10. deliverMessage(paymentTokenActive)
NOTE LEFT: See: Handle Message From Server
deactivate WS
SDK -> SDK: 11. updateTokenStatus
SDK -> MPA: 12. onPaymentInstrumentStatusChanged(id, status)
deactivate MPA
deactivate SDK
... Initial Replenishment ...
note over SDK, MDES #1C1E3F: Just after token activation transaction credentials initial replenishment is performed by SDK\r. See Transaction Credentials Initial Replenishment diagram
end
@enduml
```

DECLINE

This digitization outcome refers to red digitization decision.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES as MDES
group APPROVED
MDES --> WS : 1. responseFromDigitization(DECLINED)
activate MDES
deactivate MDES
activate WS
WS --> SDK : 2. response(error)
deactivate WS
activate SDK
SDK --> MPA : 3. result
deactivate SDK
activate MPA
MPA --> User : 4 result
deactivate MPA
@enduml
```

REQUIRE_ADDITIONAL_AUTHENTICATION

This digitization outcome always refers to yellow path digitization decision. When digitization is REQUIRE_ADDITIONAL_AUTHENTICATION then profile provisioning(See Profile Provisioning) starts automatically but remain INACTIVE until the User is authenticated(see Card Digitization Activation). Flag *additionalAuthenticationRequired* informs if additional user authentication is needed or not.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES as MDES
participant Issuer
group REQUIRE_ADDITIONAL_AUTHENTICATION
MDES --> WS : 1. responseFromDigitization(response(additionalAuthenticationRequired=true,
authenticationMethods?, tokenInfo, productConfig))
activate MDES
deactivate MDES
activate WS
opt
WS ->> Issuer: 2. send Payment Token event
end
WS --> SDK : 3. response(response(additionalAuthenticationRequired=true,
authenticationMethods?, tokenInfo, productConfig))
deactivate WS
activate SDK
SDK --> MPA : 4. result
deactivate SDK
activate MPA
MPA --> User : 5. result
deactivate MPA
... Profile Provisioning ...
note over MPA, MDES #1C1E3F: See Profile Provisioning diagram
```

... Card Digitization Activation ...
note over MPA, MDES #1C1E3F: See Card Digitization Activation diagram
end
@enduml

Card Digitization Activation

This process is applicable only if digitization has infomation that additional authentication is required. During this process User can choose one of the additional authentication methods. If user-entered authentication code is chosen then MDES will send authentication code which later should be provided by User for submission. Once user enters correct authentication code, Payment Token is activated by MDES asynchronously. After activation Wallet Server is notified that Payment Token is activated and this information is passed to Wallet SDK. After Payment Token activation, Wallet SDK start Transaction Credentials - Initial Replenishment.

NOTE: Submit authentication value is allowed only if provisioning is finished.

@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES as MDES
alt Just after digitization
MDES --> WS : 1. responseFromDigitization(additionalAuthenticationRequired=true,
\rauthenticationMethods?, tokenInfo,productConfig)
activate MDES

activate WS
WS --> SDK : 2. response(additionalAuthenticationRequired=true, \rauthenticationMethods?, tokenInfo, productConfig)
activate SDK
SDK --> MPA: 3. result(authenticationMethods)
activate MPA
else Activation not finished after digitization
User -> MPA: 4. activate token
MPA -> SDK: 5. UCP::getAdditionalAuthenticationMethods(paymentInstrumentId)
SDK -> WS: 6. getAuthenticationMethods(cardId, userSessionToken)
WS --> SDK: 7. response(authenticationMethods)
SDK --> MPA: 8. result(authenticationMethods)
end
MPA --> User: 9. show authentication methods
User -> MPA: 10. select authentication method
MPA -> SDK: 11. UCP::submitTokenAuthenticationMethod(authenticationMethod)
deactivate MPA
SDK -> WS: 12. submitTokenAuthenticationMethod(authenticationMethod, userSessionToken)
deactivate SDK
WS -> MDES: 13. submitAuthenticationMethod(authenticationMethod)
deactivate WS
MDES -> Issuer: 14. deliverAuthCode(authCode)
deactivate MDES
activate Issuer
Issuer -> User: 15. deliver authCode
deactivate Issuer
alt Provisioning finished - onProvisioningSuccess(paymentInstrumentId) was called
User -> MPA: 16. enter authCode
NOTE LEFT: User should have possibility to enter code\n only when provisioning finished
activate MPA
MPA -> SDK: 17. UCP::submitTokenAuthenticationValue(authCode)
activate SDK
SDK -> WS: 18. submitTokenAuthenticationValue(authCode, userSessionToken)
activate WS
WS -> WS: 19. checkProvisioningStatus
WS -> MDES: 20. submitAuthenticationValue(authCode)
activate MDES
MDES -> MDES: 21. verify authCode
MDES --> WS: 22. response
WS --> SDK: 23. response
deactivate WS
SDK --> MPA: 24. response
deactivate SDK
deactivate MPA
MDES -> MDES: 25. createTokenMapping
MDES -> WS: 26. notifyTokenUpdated(Active)

deactivate MDES
activate WS
opt
WS ->>Issuer: 27. send Payment Token event
end
WS -> SDK: 28. deliverMessage(paymentTokenActive)
NOTE LEFT: See: Handle Message From Server
deactivate WS
SDK -> SDK: 29. updateTokenStatus
SDK -> MPA: 30. onPaymentInstrumentStatusChanged(id, status)
deactivate MPA
deactivate SDK
end
... Initial Replenishment ...
note over SDK, MDES #1C1E3F: Just after token activation transaction credentials initial replenishment is performed by SDK\r. See Transaction Credentials Initial Replenishment diagram
@enduml

Handle Message From Server

In whole system there are processes where server needs to send messages to the device. Wallet Server has separate component which is responsible for sending messages to the device. This component uses different channels for message delivery. There are two channels: SSE(Server Sent Events) and RNS(Remote Notification Service). When message is ready for delivery, Wallet Server uses both channels to deliver such message. In first versions of Wallet Server only RNS was used, however sometimes messages were not delivered and to improve delivery new SSE channel was introduced. This channel helps in processes which start from the device and device expects message from the server. Moreover device checks messages which are still not delivered on actions where such messages are expected. Below diagram describes how delivery message process works and how needs to be handled on MPA side.

@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F

```
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant "MPA" as MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant "RNS" as RNS
opt
SDK -> WS: 1. callActionAfterWhichMessageIsExpected
WS --> SDK: 2. response
SDK -> WS: 3. openSSEConnection
end
WS -> WS: messageReadyForDelivery
opt If device has opened connection
WS -> SDK: 4. deliverUsingSSE
end
WS -> RNS: 5. deliverMessage
RNS --> WS: 6. response
RNS -> MPA: 7. deliverMessage
MPA -> MPA: 8. checkWalletSenderId
MPA-> SDK: 9. MDC:CloudMessage#process(pushData)
SDK -> SDK: 10. deduplicateMessage
SDK -> WS: 11. acknowledgeMessage
WS --> SDK: 12. response
SDK -> SDK: 13. processMessage
...Obtain pending messages...
MPA -> SDK: 14. someActionWhereMessageMayBeStillPending
SDK -> SDK: 15. doAction
SDK ->> WS: 16. getPendingMessages
WS --> SDK: 17. response
SDK -> SDK: do actions from 10 to 13

@enduml
```

Update RNS Token

Wallet Server is responsible for sending push notifications to the Wallet SDK. For that reason RNS token is passed to Wallet Server during pairing device or in some cases is obtained by SDK from MPA whenere is needed. However this token can be updated. MPA will be notified when token is being updated and then needs to obtain new RNS token and update via Wallet SDK on Wallet Server. Retrieving push notifications and RNS tokens is responsibility of the MPA.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant "Remote Notification Service" as RNS
activate RNS
RNS -> MPA: 1. onTokenRefresh
deactivate RNS
activate MPA
MPA -> MPA: 2. obtainNewToken(walletFirebase)
MPA -> SDK: 3. MDC::updateRegistrationToken(newRNSToken)
activate SDK
SDK -> WS: 4. updateRNSToken(deviceInstallationId, newRNSToken)
activate WS
WS -> WS: 5. updateRNSToken
WS --> SDK: 6. response
deactivate WS
SDK --> MPA: 7. result
deactivate SDK
deactivate MPA
deactivate RNS
@enduml
```

Profile Provisioning

During this process digitized card profile is delivered to the device. This process is triggered automatically after successful digitization where outcome is APPROVED or REQUIRE_ADDITIONAL_AUTHENTICATION. It is not possible to retry provisioning itself. To retry provisioning, previous token needs to be deleted and new digitization called hence when SDK reports that provisioning has failed then given token is automatically deleted and User can perform digitization once again. During process there is few point of failures and provisioning can be not finished at all. In this scenario Payment Tokens which are not provisioned for long period of time are delete by Wallet Server (see Removing Not Provisioned Tokens). From User perspective it can be good approach to treat digitization and provisioning as one process and inform User about steps(if User has to wait long time without any information then can treat this as some failure). From MPA perspective can be also good approach to wait for provisioning status as long as User stays on view dedicated to it. If User wants to cancel the process because provisioning status is not available for long period of time it is recommended to delete Payment Token(see Delete Payment Token via SDK) once User click cancel or back. Thanks to deletion, new digitization can be called and User does not have to wait until Payment Token is deleted by Wallet Server due to lack of provisioning.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES as MDES
MDES->WS: 1. sendRemoteNotificationMessage(mdesRemoteMessage)
activate MDES
deactivate MDES
activate WS
```

WS-> SDK: 2. deliverMessage(mdesRemoteMessage)
NOTE LEFT: See: Handle Message From Server
deactivate WS
activate SDK
SDK-> MDES: 3. provision
activate MDES
MDES --> SDK: 4. response(cardProfile)
SDK-> MDES: 5. notify provisioning result
MDES --> SDK: 6. response
deactivate MDES
SDK -> SDK: 7. store card profile
SDK -> WS: 8. confirmProvisioningStatus(SUCCESS/FAILURE)
activate WS
alt FAILURE
WS -> MDES: 9. deleteToken
activate MDES
MDES --> WS: 10. response
deactivate MDES
WS --> SDK: 11. response
SDK -> SDK: 12. deleteToken
SDK -> MPA: 13. onProvisioningFailure
activate MPA
MPA -> User: 14. please try again
deactivate MPA
else SUCCESS
WS --> SDK: 15. response
deactivate WS
SDK -> MPA: 16. onProvisioningSuccess(paymentInstrument)
deactivate SDK
activate MPA
MPA -> User: 17. card digitized successfully
deactivate MPA
end
@enduml

Getting Asset

Every field which's name consists of *assetId* is an identifier to some static asset. This asset can be:

- Card art
- Mastercard brand logos
- Issuers's logos
- Terms and Conditions

There are different types of assets and multiple formats may be supported. For example a single image may be supported in various file formats or variant sizes and most appropriate format to use

for a particular device.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant "MDES" as MDES
MPA -> SDK: 1. UCP::getAsset(assetId)
activate SDK
SDK -> WS: 2. getAsset(assetId)
activate WS
WS -> MDES: 3. getAsset(assetId)
activate MDES
MDES --> WS: 4. response(content)
deactivate MDES
WS --> SDK: 5. response(content)
deactivate WS
SDK --> MPA: 6. result(content)
deactivate SDK
@enduml
```

Transaction Credentials Replenishment

Transaction Credentials are unique per transactions keys that are used to calculate cryptograms in transactions. Each set of credentials is linked with a unique Application Transaction Counter (ATC). Each set of credentials can only be used for one transaction. There is a limit (set on MDES

onboarding) of transaction credentials stored on device.

Transaction Credentials - Automatic Replenishment

After every transaction Wallet SDK checks if number of transaction credentials is below, preconfigured during SDK setup, threshold. If yes then SDK will call replenish. During replenish process transaction credentials are being delivered to mobile application.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES as MDES
SDK->SDK: 1. detectTransactionCredentialsRemainingBelowThreshold
alt Request session if required
activate SDK
SDK-> MDES: 2. requestSession
activate MDES
MDES--> SDK: 3. response
MDES-> WS: 4. sendRemoteNotificationMessage(mdesRemoteMessage)
deactivate MDES
activate WS
WS -> SDK: 5. deliverMessage(mdesRemoteMessage)
NOTE LEFT: See: Handle Message From Server
deactivate WS
```

```
end
SDK-> MDES: 6. replenish
activate MDES
MDES-> MDES: 7. checkIfPaymentTokenIsActive
MDES-->SDK: 8. response(transactionCredentials)
deactivate MDES
SDK->MPA: 9. onReplenishSuccess(paymentInstrument)
deactivate SDK
@enduml
```

Transaction Credentials - Initial Replenishment

Initial replenishment is process which starts directly after successful token activation. Wallet SDK is notified by Wallet Server using push notification or refreshing payment instruments. No action is needed by MPA.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES as MDES
WS -> SDK: 1. notifyTokenUpdated(Active)
activate WS
deactivate WS
activate SDK
alt Request session if required
```

```
SDK-> MDES: 2. requestSession
activate MDES
MDES--> SDK: 3. response
MDES-> WS: 4. sendRemoteNotificationMessage(mdesRemoteMessage)
deactivate MDES
activate WS
WS-> SDK: 5. deliverMessage(mdesRemoteMessage)
NOTE LEFT: See: Handle Message From Server
deactivate WS
end
SDK-> MDES: 6. replenish
activate MDES
MDES-> MDES: 7. checkIfPaymentTokenIsActive
MDES-->SDK: 8. response(transactionCredentials)
deactivate MDES
SDK -> MPA: 9. onReplenishSuccess(paymentInstrument)
deactivate SDK
@enduml
```

Transaction Credentials - Manual Replenishment

There are scenarios when automatic replenish is not possible (user is not able to connect with Internet) and after some number of transactions, transaction credentials number will decrease to 0. In such case MPA should handle NO_TRANSACTION_CREDENTIALS error from transaction listener, show user proper alert and call replenish method manually. MPA can also check number of transaction credentials at any other time.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
```

```
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES as MDES
MPA -> SDK: 1. UCP::replenishCredentials(paymentInstrumentId)
activate MPA
deactivate MPA
activate SDK
alt Request session if required
SDK-> MDES: 2. requestSession
activate MDES
MDES--> SDK: 3. response
MDES-> WS: 4. sendRemoteNotificationMessage
deactivate MDES
activate WS
WS-> SDK: 5. deliverMessage(mdesRemoteMessage)
NOTE LEFT: See: Handle Message From Server
deactivate WS
end
SDK-> MDES: 6. replenish
activate MDES
MDES-> MDES: 7. checkIfPaymentTokenIsActive
MDES-->SDK: 8. response(transactionCredentials)
deactivate MDES
SDK -> MPA: 9. onReplenishSuccess(paymentInstrument)
deactivate SDK
@enduml
```

Transacting

Wallet SDK provides functionalities to make contactless payments (using HCE) and e-commerce payments.

Contactless Transaction

Contactless transaction uses Android HCE. On MPA side HostApduService should be implemented. Depending on chosen CDCVM and on how transaction is started, user experience is different and MPA should interact with Wallet SDK in different way. The final decision about transaction processing belongs to MPA. Wallet SDK provides transaction information and based on that and User authentication, MPA can advise to proceed, decline or require authentication(if User should be authenticated but was not). For contactless transaction Wallet SDK provides result of transaction. This result is only from the communication between MPA and Terminal. Transaction Processing with Payment Network is done separately (see Transaction Processing). Also after every contactless

transaction, Transaction Credentials Replenishment is performed automatically by SDK if needed(see Transaction Credentials - Automatic Replenishment).

NOTE: The way of authentication depends on MPA. For transaction User may also choose specific card. If no card is chosen, SDK will use the one which is set as default for contactless payments. Whenever user is authenticated or chose card for payment MPA should pass this information when *onContactlessPaymentStarted* is called.

As was described above, the final decision(PROCEED, DECLINE, AUTHENTICATION_REQUIRED) for given transaction is taken on MPA side based on transaction information and User authentication. Because of that reason there could be different scenarios which may occur and transaction experience will be single or double tap.

Sample scenarios:

- User can be already authenticated and if MPA will not decline transaction then will be processed as single tap,
- velocity check counters can be applied and even if User was not authenticated MPA can decide to proceed transaction without authentication, taking decision based on transaction information,
- User was not authenticated but MPA recognised transaction as authentication needed. MPA returns AUTHENTICATION_REQUIRED decision and and SDK informs MPA that authentication is needed.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
```

```
participant "HostApduService" as HAS
participant "Wallet SDK" as SDK
participant Terminal as TER
opt User selects particular card for payment
MPA -> User: 1. showCards
activate MPA
User -> MPA: 2. selectCard
deactivate MPA
end
User -> MPA: 3. pay
User -> TER : 4. contactless 1st tap
activate TER
loop
TER -> HAS: 5. processCommandApdu(commandApdu, extras)
activate HAS
HAS -> SDK: 6. UCP::Pay#processHceApduCommand(apdu, extras)
activate SDK
group Select PPSE
SDK -> MPA: 7. onContactlessPaymentStarted()
activate MPA
MPA -> MPA: 8. checkIsUserAuthenticated
alt isAuthenticated=true
MPA -> SDK: 9. UCP::Pay#setUserAuthenticatedForPayment(paymentInstrumentId, pin?)
end
alt selectedCard == null
SDK -> SDK: 10. useDefaultPaymentInstrumentForContactless
else selectedCard != null
MPA -> SDK: 11. UCP::Pay#selectForPayment(selectedPaymentInstrumentId)
deactivate MPA
end
end
group Generate AC command
SDK -> MPA: 12. getFinalDecisionForTransaction(isUserAuthenticated,recommendedAdvice, trxInfo)
activate MPA
MPA -> MPA: 13. checkTrxAndAuthentication
MPA --> SDK: 14. result(advice)
deactivate MPA
end
SDK --> HAS: 15. responseApdu
HAS --> TER: 16. responseApdu
deactivate HAS
end
alt advice=AUTHENTICATION_REQUIRED
SDK -> MPA: 17. onAuthRequiredForContactless(paymentInstrument, trxInfo)
activate MPA
MPA -> User: 18. show authentication view with trx info
```

```
User -> MPA: 19. authenticate
User -> TER: 20. contactless 2nd tap
loop
TER -> HAS: 21. processCommandApdu(commandApdu, extras)
activate HAS
HAS -> SDK: 22. UCP::Pay#processHceApduCommand(apdu, extras)
group Select PPSE
SDK -> MPA: 23. onContactlessPaymentStarted()
MPA -> MPA: 24. checkIsUserAuthenticated
alt isAuthenticated=true
MPA -> SDK: 25. UCP::Pay#setUserAuthenticatedForPayment(paymentInstrumentId, pin?)
end
alt selectedCard == null
SDK -> SDK: 26. useDefaultPaymentInstrumentForContactless
else selectedCard != null
MPA -> SDK: 27. UCP::Pay#selectForPayment(selectedPaymentInstrumentId)
end
end
group Generate AC command
SDK -> MPA: 28. getFinalDecisionForTransaction(isUserAuthenticated=true,recommendedAdvice,
trxInfo)
MPA -> MPA: 29. checkTrxAndAuthentication
MPA --> SDK: 30. result(PROCEED)
end
SDK --> HAS: 31. responseApdu
HAS --> TER: 32. responseApdu
deactivate HAS
deactivate TER
end
end
SDK -> MPA: 33. onContactlessPaymentCompleted(paymentInstrument, trxInfo, trxResult)
deactivate SDK
MPA -> User: 34. show trx info view
deactivate MPA
...Transaction Processing ...
note over HAS #1C1E3F: See Transaction Processing diagram
...Transaction Credentials Automatic Replenishment ...
note over HAS #1C1E3F: See Transaction Credentials Automatic Replenishment diagram
@enduml
```

E-commerce transaction

E-commerce transaction can be processed using DSRP. Every DSRP transaction has to be authenticated. Depending on implementation e-commerce payment can be preceded with one time password authentication or just device level authentication.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
box "Consumer mobile android device" #white
participant "Merchant App" as MerchantApp
participant MPA
participant "Wallet SDK" as SDK
end box
participant Merchant
participant "Wallet Server" as WS
participant MDES
participant Issuer
participant "Payment Gateway" as PG
User -> MerchantApp: 1. pay
activate MerchantApp
MerchantApp -> Merchant: 2. startTransaction
activate Merchant
Merchant --> MerchantApp: 3. response(transactionId,\r unpredictableNumber)
deactivate Merchant
MerchantApp -> MPA: 4. getDsrpData(transactionId, trxInfo)
deactivate MerchantApp
activate MPA
alt One Time Password
MPA -> User: 5. select payment instrument
User -> MPA: 6. payment instrument
MPA -> SDK: 7. UCP::requestAuthCodeForPayment(transactionId \ras authenticationRequestId)
activate SDK
SDK -> WS: 8. requestAuthenticationCodeForPayment(authenticationRequestId)
```

activate WS
WS -> MDES: 9. requestAuthenticationCodeForPayment(authenticationRequestId)
activate MDES
MDES -> Issuer: 10. sendAuthenticationCode
activate Issuer
Issuer --> MDES: 11. response
MDES --> WS: 12. response
deactivate MDES
WS --> SDK: 13. response
deactivate WS
SDK --> MPA: 14. result
deactivate MPA
deactivate SDK
Issuer -> User: 15. sendAuthenticationCode(authenticationCode)
deactivate Issuer
User -> MPA: 16. provide authentication code
activate MPA
MPA -> SDK: 17. UCP::validateAuthenticationCodeForPayment(authenticationCode,\r transactionId)
activate SDK
SDK -> WS: 18. validateAuthenticationCodeForPayment(authenticationCode,\r
authenticationRequestId)
activate WS
WS -> MDES: 19. authenticate(authenticationCode,\r authenticationRequestId)
activate MDES
MDES --> WS: 20. response
deactivate MDES
WS -> WS: 21. createDigitalSignature(authenticationRequestId)
WS --> SDK: 22. response(digitlSignature)
deactivate WS
SDK --> MPA: 23. result(digitalSignature)
else Device Level Auth
MPA -> User: 24. show authentication view
User -> MPA: 25. authenticate
end
MPA -> SDK: 26. UCP::setUserAuthenticatedForPayment(id, pin?)
MPA -> SDK: 27. UCP::processDsrpTransaction(id, trxInfo)
SDK --> MPA: 28. result(dsrpData)
deactivate SDK
MPA --> MerchantApp: 29. result(dsrpData)
deactivate MPA
activate MerchantApp
MerchantApp -> MerchantApp: 30. encryptPaymentDataForTransit(dsrpData)
MerchantApp -> Merchant: 31. finishTransaction(encryptedPaymentData, \rdigitalSignature?,
transactionId)
activate Merchant
opt

```
Merchant -> Merchant: 32. validateSignature
note right: this check is needed to proof that given transactionId\n\r was preceded with OTP
end
Merchant -> PG: 33. processPayment(pan, exp, cryptogram)
activate PG
PG --> Merchant: 34. response
deactivate PG
Merchant --> MerchantApp: 35. response
deactivate Merchant
MerchantApp -> User: 36. show result
deactivate MerchantApp
@enduml
```

Transaction Processing

Transaction Processing starts after contactless communication between terminal and MPA in case of contactless payment or after payment gateway transaction authorization initiation in case of e-commerce payment. After authorization MDES notifies Wallet Server about the result of the authorization and sends transaction information. Transaction information is sent to MPA.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant "Terminal/PG" as TER
participant "Payment Network" as PN
```

```
participant MDES
participant Issuer
TER -> PN: 1. authorizeTransaction(tokenPAN, cryptogram)
activate PN
activate TER
PN -> MDES: 2. detokenize
activate MDES
MDES -> MDES: 3. lookup token mapping
MDES --> PN: 4. response(PAN)
deactivate MDES
PN -> Issuer: 5. authorize(PAN)
activate Issuer
Issuer --> PN: 6. response
deactivate Issuer
PN --> TER: 7. response
deactivate TER
PN -> MDES: 8. storeTransactionDetails
deactivate PN
activate MDES
MDES -> WS: 9. pushTransactionDetails
deactivate MDES
activate WS
alt store transaction enabled
WS -> WS: 10. storeTransaction
end
opt
WS ->>Issuer: 11. send transaction event
end
WS -> SDK: 12. deliverMessage(trxInfo)
NOTE LEFT: See: Handle Message From Server
deactivate WS
activate SDK
SDK -> MPA: 13. onNewTransaction(trxDetails)
deactivate SDK
activate MPA
MPA -> User: 14. showSystemNotification(trxDetails)
deactivate MPA
@enduml
```

Setting Defaults for Payment

SDK manages default Payment Instrument for contactless payments. After digitization, if there is no default Payment Instrument, SDK sets digitized Payment Instrument after activation as default. In case where there are more than one active Payment Instruments and current default Payment Instrument is deleted or suspended, the SDK will set first active Payment Instrument as default. Default Payment Instrument can be changed at any time. Only active Payment Instrument can be

set as default.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
MPA->User: 1. payment instrument list
activate MPA
User->MPA: 2. choose default for contactless payment)
MPA->SDK: 3. UCP::setDefaultForContactless(paymentInstrumentId)
activate SDK
SDK-> SDK: 4. storeDefault
SDK-->MPA: 5. result
deactivate MPA
deactivate SDK
@enduml
```

Login On Wallet Server

User data are protected by User session token which is issued by Wallet Server after providing authentication factor. Authentication factor is provided first in pairing device and then session is created. Since session has limited period of validity, it needs to be refreshed using login on Wallet Server methods.

Login on Wallet Server using Trusted Identity

In the Integrated implementation model User authentication doesn't occur directly on Wallet Server . Wallet Server will require User authentication when some user data will be requested. If User session token is no longer valid, SDK will return USER_UNATHORIZED error. In such case Trusted Identity needs to be prepared on Issuer server and sent via Wallet SDK in loginByTrustedIdentity method. MPA can decide whether ask User to provide authentication data or not. The latter case regards situation when user is already authenticated and Trusted Identity can be immediately returned from Issuer based on already valid session on Issuer side. During login process Wallet Server checks if given device still exists, if not then responds with CANT_FIND_DEVICE status which is interpreted on SDK side as given device is deleted and all local data stored on SDK side are cleared.

Since MDC 2.14.5 for devices which are already paired, Wallet SDK will try to asynchronously register device in new delivery message service. To make it possible *CloudMessagingRegistrationProvider* needs to be implemented and provided within setup.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant "Issuer" as Issuer
MPA -> SDK: 1. invoke API
activate MPA
activate SDK
SDK -> WS: 2. invoke API
activate WS
WS --> SDK: 3. response(USER_UNATHORIZED)
```

deactivate WS
SDK --> MPA: 4. result(USER_UNATHORIZED)
MPA->User: 5. show authenticate view
User->MPA: 6. put authentication data
MPA->Issuer: 7. authenticate
activate Issuer
Issuer -> Issuer: 8. generateTrustedIdentity - signed user id
Issuer --> MPA: 9. response(trustedIdentity)
deactivate Issuer
MPA-> SDK: 10. MDC::loginByTrustedIdentity(trustedIdentity)
SDK-> WS: 11. loginByTrustedIdentity(trustedIdentity)
alt Device not registered in new delivery message service
SDK -> MPA: 12. CloudMessagingRegistrationProvider::getRegistrationToken
SDK ->> WS: 13. registerDeviceForNewMessageDelivery(cloudMessageToken)
WS --> SDK: 14. response
end
activate WS
WS -> WS: 15. check if device exists
alt device exists
WS-> WS: 16. verify trusted identity
WS --> SDK: 17. response(userSessionToken)
SDK -> SDK: 18. store(userSessionToken)
SDK-->MPA: 19. result
MPA -> SDK: 20. invoke API
else device not exists
WS --> SDK: 21. response(CANT_FIND_DEVICE)
deactivate WS
SDK -> SDK: 22. clearAllLocalData
SDK --> MPA: 23. result(CANT_FIND_DEVICE)
deactivate MPA
deactivate SDK
... Pair device ...
note over MPA, WS #1C1E3F: See Pairing Device diagram
MPA -> SDK: 24. invoke API
end
@enduml

Getting Cards

When cards are already placed on Wallet Server, then they can be displayed to User in the application. Cards have identifiers which can be used e.g. for digitization.

@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF

```
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
MPA->SDK: 1. MDC::getAllCards
activate MPA
activate SDK
SDK -> WS: 2. getAllCards(userSessionToken)
WS --> SDK: 3. response(userCards)
deactivate WS
SDK --> MPA: 4. result(cardList)
deactivate SDK
deactivate MPA
@enduml
```

Getting Payement Intruments

After digitization process, payment instrument is stored in VCP SDK module of Wallet SDK. Payment instrument in context of VCP SDK is digitized card and contains information like:

- id (specified id which helps MPA to identify payment instrument which was digitized from MPA),
- status,
- transaction credentials count,
- paymentTokenId etc. .

MPA can get information about all Payment Instruments from the Wallet SDK at any time. Payment instruments will be retrieved only from local storage that is part of SDK. Payment Tokens for Payment Instruments can be refreshed/pulled from Wallet Server on demand. This scenario should be considered only when User e.g. makes swipe to refresh.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant "Issuer" as IS
MPA->SDK: 1. UCP::getAllPaymentInstruments(refresh)
activate MPA
alt refresh = true
activate SDK
SDK -> WS: 2. getAllPaymentTokens(userSessionToken)
activate WS
WS -> SDK: 3. response(devicePaymentTokens)
deactivate WS
SDK -> SDK: 4. updateLocalStorage
end
SDK --> MPA: 5. result(paymentInstrumentList)
deactivate SDK
deactivate MPA
@enduml
```

Getting Transaction History

It is possible that transaction history will be stored on Wallet Server for infinite time. This should be specified during onboarding. If this options is enabled, MPA can retrieve transaction history for given user and filtering. Transactions are returned in corresponding parts for better user experience. If next part is available then response from previous part contain information needed

for requesting next part. MPA should check if next part is not empty and then make another request.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
loop next != null
MPA->SDK: 1. MDC::getTransactionsHistory(filters, next?)
activate MPA
activate SDK
SDK -> WS: 2. getTransactionHistory(filters, next?, userSessionToken, ...)
activate WS
WS -> SDK: 3. response(transactionHistoryList, next?)
deactivate WS
SDK --> MPA: 4. result(transactionHistoryList, next?)
deactivate SDK
deactivate MPA
end
@enduml
```

Payment Token Lifecycle Management via SDK

Payment Token lifecycle management can be done via SDK.

Delete Payment Token via SDK

Payment Token can be deleted using SDK.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES
participant Issuer
User -> MPA: 1. Delete payment token
activate MPA
MPA -> SDK: 2. UCP::delete(paymentInstrumentId, reason)
deactivate MPA
activate SDK
SDK -> WS: 3. deletePaymentToken(userSessionToken, paymentTokenId, reason)
activate WS
WS -> MDES: 4. Delete token
activate MDES
MDES -> MDES: 5. Delete token mapping
MDES --> WS: 6. response
deactivate MDES
WS --> SDK: 7. response
opt
WS ->> Issuer: 8. send Payment Token event
end
```

```
deactivate WS
alt Request session if required
SDK -> MDES: 9. request session
activate MDES
MDES --> SDK: 10. response
MDES -> WS: 11. sendRemoteNotificationMessage
deactivate MDES
activate WS
WS -> SDK: 12. deliverMessage(mdesRemoteMessage)
NOTE LEFT: See: Handle Message From Server
deactivate WS
end
SDK -> MDES: 13. delete(tokenUniqueReference)
activate MDES
MDES --> SDK: 14. response
deactivate MDES
SDK -> SDK: 15. Delete transaction credentials, card profile
SDK -> MPA: 16. onPaymentInstrumentStatusChanged(id, status)
deactivate SDK
MPA --> User: 17. show update view
deactivate MPA
@enduml
```

Errors Reporting

Wallet SDK performs some security checks. When any issue is detected, Wallet SDK reports error to Wallet Server and clears own data. Also notification to MPA is called.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
```

```
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
SDK -> SDK: 1. detectSecurityIssue
activate SDK
SDK -> SDK: 2. clearSDKData
SDK -> WS: 3. reportSecurityIssue
activate WS
deactivate WS
deactivate SDK
SDK -> MPA: 4. onSecurityIssueAppeared(event)
activate MPA
deactivate MPA
MPA -> User: 5. show information
@enduml
```

Device Unpairing

Unpairing device clears all modules data and report that fact only if possible to server. If server receives this signal then removes all device data including provisioned Payment Tokens. If not then data are cleared locally only - similar like during app uninstallation. This can be used for scenario when MPA does not want to use SDK at all or for scenario when MPA supports switching between users accounts on the same installation. If MPA detects that new User is trying to log into application in case when previous had digitized cards, immediately should clear all data from previous, since SDK stores data in context of one User only.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
```

```
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
MPA -> SDK: 1. MDC::unpairDevice
activate MPA
activate SDK
opt
SDK -> WS: 2. unpairDevice
activate WS
WS --> SDK: 3. response
deactivate WS
end
SDK -> SDK: 4. clearAllData
SDK --> MPA: 5. result
deactivate SDK
deactivate MPA
@enduml
```

MDES Initiated

This section describes use cases which are initiated from MDES.

Re-digitization

Re-digitization process can be triggered by MDES for several use cases:

Token Expiry

One month before token expiry MDES will request for redigitization.

Attribute Change

Issuer may perform an attribute change at the BIN account-range level impacting theri MDES enabled ranges. Some device tokens may need to have their data refreshed to match the new attributes.

BIN Account-Range Split

Issuer may perform a BIN account-range split. Some existing tokens may need to be updated to ensure that they are linked to the correct funding BIN account ranges internally.

PAN Update in Different Account Range

Issuer may update existing PAN with new Pan in a different BIN account range.

For above cases:

- token unique reference remains the same
- token expiration date is extended by three years from the date of redigitization

After successful redigitization process transaction credentials replenishment is called in case where Payment Token is active.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES
MDES -> WS: 1. notifyTokenUpdated(redigitize=true)
activate MDES
activate WS
WS -> MDES: 2. redigitize
MDES --> WS: 3. response
WS --> MDES: 4. response
deactivate MDES
deactivate WS
MDES->WS: 5. sendRemoteNotificationMessage(mdesRemoteMessage)
activate MDES
deactivate MDES
```

activate WS
WS-> SDK: 6. deliverMessage(mdesRemoteMessage)
NOTE LEFT: See: Handle Message From Server
deactivate WS
activate SDK
SDK-> MDES: 7. provision(redigitize=true)
activate MDES
MDES --> SDK: 8. response(cardProfile)
SDK-> MDES: 9. notify provisioning result(redigitize=true)
MDES --> SDK: 10. response
SDK -> WS: 11. confirmReProvisioningStatus(SUCCESS/FAILURE)
alt FAILURE
WS -> WS: 12. markRedigitizationAsFailed
note left: redigitization process will be retried after some period of time
SDK -> MPA: 13. onReProvisioningFailure(paymentInstrument)
else SUCCESS
SDK -> SDK: 14. clearTransactionCredentials
SDK -> MPA: 15. onReProvisioningSuccess(paymentInstrument)
deactivate SDK
MDES -> WS: 16. notifyTokenUpdated(redigitized=false)
deactivate MDES
activate WS
opt
WS ->> Issuer: 17. send Payment Token event
end
WS -> SDK: 18. deliverMessage(PAYMENT_TOKEN_INFO_CHANGE(redigitized=true))
NOTE LEFT: See: Handle Message From Server
deactivate WS
activate SDK
SDK -> SDK: 19. replenish
... Automatic Replenishment ...
note over SDK, MDES #1C1E3F: Just after reprovisioning transaction credentials initial replenishment is performed by SDK\r. See Transaction Credentials Initial Replenishment diagram
end
@enduml

Wallet Server Initiated

Since important processes are asynchronous in MDES and there are many point of failures, wallet server provides additional functionalities to resolve some failure scenarios by running some operations on own side.

Removing Not Provisioned Tokens

Wallet Server checks periodically DEVICE Payment Tokens and verify if provisioning is completed. These Payment Tokens which have provisioning status in progress for long period of time are deleted automatically and from User perspective process needs to be started again. By default this period is set to 1 hour but can be modified.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES
WS -> WS: 1. find not provisioned payment tokens for a\r long period of time
activate WS
loop Not provisioned payment tokens for a long period of time
WS -> MDES: 2. Delete token
activate MDES
MDES -> MDES: 3. Delete token mapping
MDES --> WS: 4. response
deactivate MDES
opt
WS ->> Issuer: 5. send Payment Token event
end
WS -> SDK: 6. deliverMessage(paymentTokenDelete)
NOTE LEFT: See: Handle Message From Server
deactivate WS
activate SDK
alt Request session if required
```

SDK -> MDES: 7. request session
activate MDES
MDES --> SDK: 8. response
MDES -> WS: 9. sendRemoteNotificationMessage
deactivate MDES
activate WS
WS -> SDK: 10. deliverMessage(mdesRemoteMessage)
NOTE LEFT: See: Handle Message From Server
deactivate WS
end
SDK -> MDES: 11. delete(tokenUniqueReference)
activate MDES
MDES --> SDK: 12. response
deactivate MDES
SDK -> SDK: 13. Delete transaction credentials, card profile
SDK -> MPA: 14. onPaymentInstrumentStatusChanged(id, status)
deactivate SDK
end
deactivate MPA
@enduml

Wallet Server Admin API Initiated

This section describes use cases which are initiated from Wallet Server Admin Panel.

Admin Card Deletion

During this process all data related to given card are deleted. Payment Tokens are deleted asynchronously.

@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F

```
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES
participant "Admin Panel" as AP
participant Issuer
AP -> WS: 1. deleteCard(cardId)
activate WS
activate AP
WS -> WS: 2. deleteCard
WS --> AP: 3. response
deactivate AP
loop All Payment Tokens for card
WS -> MDES: 4. Delete token
activate MDES
MDES -> MDES: 5. Delete token mapping
MDES --> WS: 6. response
deactivate MDES
opt
WS ->>Issuer: 7. send Payment Token event
end
WS -> SDK: 8. deliverMessage(paymentTokenDelete)
NOTE LEFT: See: Handle Message From Server
deactivate WS
activate SDK
alt Request session if required
SDK -> MDES: 9. request session
activate MDES
MDES --> SDK: 10. response
MDES -> WS: 11. sendRemoteNotificationMessage
deactivate MDES
activate WS
WS -> SDK: 12. deliverMessage(mdesRemoteMessage)
NOTE LEFT: See: Handle Message From Server
deactivate WS
end
SDK -> MDES: 13. delete(tokenUniqueReference)
activate MDES
MDES --> SDK: 14. response
deactivate MDES
```

```
SDK -> SDK: 15. Delete transaction credentials, card profile
SDK -> MPA: 16. onPaymentInstrumentStatusChanged(id, status)
deactivate SDK
end
deactivate MPA
@enduml
```

Admin Device Deletion

During this process all data related to given device are deleted. Payment Tokens are deleted asynchronously.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES
participant "Admin Panel" as AP
AP -> WS: 1. deleteDevice(deviceInstallationId)
activate AP
activate WS
WS --> AP: 2. response
deactivate AP
loop All Payment Tokens for given device
WS -> MDES: 3. delete token
activate MDES
```

```
MDES --> WS: 4. response
deactivate WS
deactivate MDES
end
@enduml
```

Admin Token Deletion

Payment Token can be deleted via admin panel.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES
participant "Admin Panel" as AP
AP -> WS: 1. deletePaymentToken(paymentTokenId)
activate AP
activate WS
WS -> MDES: 2. Delete token
activate MDES
MDES -> MDES: 3. Delete token mapping
MDES --> WS: 4. response
deactivate MDES
WS --> AP: 5. response
```

deactivate AP
opt
WS ->> Issuer: 6. send Payment Token event
end
WS -> SDK: 7. deliverMessage(paymentTokenDeleted)
deactivate WS
activate SDK
NOTE LEFT: See: Handle Message From Server
deactivate MDES
alt Request session if required
SDK -> MDES: 8. request session
activate MDES
MDES --> SDK: 9. response
MDES -> WS: 10. sendRemoteNotificationMessage
deactivate MDES
activate WS
WS -> SDK: 11. deliverMessage(mdesRemoteMessage)
NOTE LEFT: See: Handle Message From Server
deactivate WS
end
SDK -> MDES: 12. delete(tokenUniqueReference)
activate MDES
MDES --> SDK: 13. response
deactivate MDES
SDK -> SDK: 14. Delete transaction credentials, card profile
SDK -> MPA: 15. onPaymentInstrumentStatusChanged(id, status)
deactivate SDK
MPA --> User: 16. show update view
deactivate MPA
@enduml

Admin Token Suspension

Payment Token can be suspended via admin panel.

@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F

```
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES
participant "Admin Panel" as AP
participant Issuer
AP -> WS: 1. suspendToken(paymentTokenId)
activate WS
activate AP
WS -> MDES: 2. suspend token
activate MDES
MDES -> MDES: 3. Suspend token
MDES --> WS: 4. response
deactivate MDES
WS --> AP: 5. response
deactivate AP
opt
WS ->> Issuer: 6. send Payment Token event
end
WS -> SDK: 7. deliverMessage(paymentTokenSuspend)
NOTE LEFT: See: Handle Message From Server
deactivate WS
activate SDK
SDK -> SDK: 8. suspend
SDK -> MPA: 9. onPaymentInstrumentStatusChanged(id, status)
deactivate SDK
deactivate MPA
@enduml
```

Admin Token Unsuspension

Payment Token can be unsuspended via admin panel.

```
@startuml
skinparam ParticipantPadding 30
```

```
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES
participant "Admin Panel" as AP
AP -> WS: 1. unsuspendToken(paymentTokenId)
activate WS
activate AP
WS -> MDES: 2. unsuspend token
activate MDES
MDES -> MDES: 3. Unsuspend token
MDES --> WS: 4. response
deactivate MDES
WS --> AP: 5. response
deactivate AP
opt
WS ->>Issuer: 6. send Payment Token event
end
WS -> SDK: 7. deliverMessage(paymentTokenUnsuspend)
deactivate WS
activate SDK
SDK -> SDK: 8. activate
SDK -> MPA: 9. onPaymentInstrumentStatusChanged(id, status)
deactivate SDK
deactivate MPA
... Replenishment ...
note over SDK, MDES #1C1E3F: Just after token activation transaction credentials replenishment is
```

performed by SDK\r. See Transaction Credentials Automatic Replenishment diagram
@enduml

Admin User Deletion

During this process all data related to given User are deleted. Payment Tokens are deleted asynchronously.

```
@startuml
skinparam ParticipantPadding 30
skinparam BoxPadding 30
skinparam noteFontColor #FFFFFF
skinparam noteBackgroundColor #1C1E3F
skinparam noteBorderColor #1C1E3F
skinparam noteBorderThickness 1
skinparam sequence {
ArrowColor #1C1E3F
ArrowFontColor #1C1E3F
ActorBorderColor #1C1E3F
ActorBackgroundColor #FFFFFF
ActorFontStyle bold
ParticipantBorderColor #1C1E3F
ParticipantBackgroundColor #1C1E3F
ParticipantFontColor #FFFFFF
ParticipantFontStyle bold
LifeLineBackgroundColor #1C1E3F
LifeLineBorderColor #1C1E3F
}
actor User
participant MPA
participant "Wallet SDK" as SDK
participant "Wallet Server" as WS
participant MDES
participant "Admin Panel" as AP
AP -> WS: 1. deleteUser(userId)
activate AP
activate WS
WS -> WS: 2. delete cards, devices
WS --> AP: 3. response
deactivate AP
loop All Payment Tokens for given User
WS -> MDES: 4. delete token
activate MDES
MDES --> WS: 5. response
deactivate WS
deactivate MDES
```

end
@enduml

Summary of Changes

This section describes changes introduced in next version based on time

Version 2.0

- Base version of the document describing VCP Solution for cards

Version 2.1

- Introduced *onContactlessPaymentStarted* in Contactless Transaction use case
- Added new use case: Handle Message From Server
- Introduced provider for *cloudMessageRegistrationToken* in Login On Wallet Server use case
- Updated all use cases with new way of message delivery from server
- Added sending Payment Token event when token is created or updated

---

Revision #132
Created 4 June 2022 18:46:07 by Wojciech Nowakowski
Updated 24 August 2023 05:59:43 by Jagoda Mazurek