# Specifiche tecniche di integrazione

# CSR-D Lombardia

# Modulo SDK

Revisione del Documento: **01**

Data revisione: **03/10/2025**

| | Struttura | Data |
|---|---|---|
| **Redatto da:** | *Divisione - Servizi per le Entrate Regionali, Territorio, Ambiente, Mobilità – ARIA S.p.A.* | 03/10/2025 |

## Cronologia delle Revisioni

| Revisione | Data | Sintesi delle Modifiche |
|---|---|---|
| 01 | 03/10/2025 | Prima versione del documento che descrive il modulo SDK |

**Tabella 1 - Cronologia delle revisioni**

| Limiti di utilizzo del documento |
|---|
| In base alla classificazione del documento. |

**Tabella 2 - Link di utilizzo del documento**

# Sommario

# Indice delle Tabelle

## Glossario e Acronimi

| Nome | Descrizione |
|---|---|
| Account | Rappresenta l'identificativo di uno specifico utente registrato su un portale web |
| ABT | Account Based Ticketing |
| Alias_R | Alias Regionale, identificativo della carta nel sistema regionale |
| Anagrafica utente | Rappresenta i dati anagrafici di un utente |
| ARIA | Azienda Regionale per l'Innovazione e gli Acquisti S.p.A. |
| Autenticazione | Significa che l'utente è correttamente loggato nel sistema |
| BE | Back End |
| BELL | Bigliettazione Elettronica Lombardia |
| CCA | Centro controllo aziendale, possiamo considerarlo come il BE dell'OT |
| CCB | Centro di Controllo Bacino |
| CSR | Centro Servizi Regionale |
| CSR-D | CSR Digitale |
| Device_ID | Identifico univoco del Device |
| Device_ID_Utilizzo | Identifico univoco del Device_ID dove è stato utilizzato un Tdv |
| EMV | Europay Mastercard Visa |
| FE | Front End |
| IAM | Identity and Access Management |
| ID_Anagrafica | Identifico dell'anagrafica della tessera nei vari sistemi (OT o CSR) |
| ID_Regionale | Identificativo univoco dell'utente nel sistema regionale |
| OT | Operatore di Trasporto |
| R_PPxx | ID Requisito funzionale post-pagato |
| R_PREPxx | ID Requisito funzionale prepagato |
| RL | Regione Lombardia |
| SBE | Sistema di Bigliettazione Elettronica |
| STIBM | Sistema Tariffario Integrato del Bacino di Mobilità |
| STIL | Sistema Tariffario Integrato Lineare |
| STIR | Sistema Tariffario Integrato Regionale |
| SW | Software |
| TdV | Titoli di Viaggio |
| Tessera | Identificativo di un certo utente. Un utente può avere più tessere |
| TIR | Tariffa Integrata Regionale |
| TouchPoint | Si intende il FE, cioè dove l'utente visualizza effettivamente le informazioni propagate dal BE |
| TPL | Trasporto Pubblico Locale |
| Vettore | Azienda di trasporto che offre servizi agli utenti |
| TSP | Token Service Provider |

**Tabella 3 - Glossario e Acronimi**

# 1. Introduzione

Nel presente documento viene mostrato come poter installare, gestire ed integrare il **plus-wallet-sdk**. Tale SDK, sviluppata in modalità nativa per IOS e per Android, racchiude le funzionalità di generazione del deviceId / deviceToken e la generazione di un qrCode compatibile con il CSR-D. I sistemi operativi supportati sono iOS e Android. Per poter integrare l'SDK all'interno delle app degli Ot sarà possibile, in funzione della tecnologia di riferimento dell'app digitale dell'OT, utilizzare direttamente l'SDK in modalità nativa qualora l'app sia sviluppata in nativo oppure sviluppare un wrapper che disintermedi le logiche specifiche del SO dal framework di sviluppo (es. React Native).

Per la gestione del repository si sono utilizzate due soluzioni:

- per **iOS** si utilizza il gestore di dipendenze **CocoaPods**,
- per **Android** si utilizza invece il gestore **Maven**.

# 2. Importazione progetto iOS con Cocoapods

## 2.1 Autenticazione con il repository remoto

Per l'importazione dell'sdk in un progetto iOS, è necessario autenticarsi con il repository remoto. Per fare questo lanciare da terminale il seguente comando:

pod repo add pluservice-ticketing https://devops.superdriver.it/plus-wallet-sdk-cocoapods

Una volta lanciato questo comando, il terminale chiederà di inserire le credenziali fornite con l'sdk, per autenticarsi con il repository remoto. Le credenziali saranno disponibili una volta configurato l'ambiente di testing.

## 2.2 Aggiunta della dipendenza nel Pod file

Ora per importare l'sdk nel progetto iOS, è sufficiente aggiungere la seguente riga nel Pod file:

pod 'PlusTicketingSdk'

Poi lanciare da terminale il seguente comando per installare l'sdk:

pod install

# 3. Importazione progetto Android con Maven

## 3.1 Autenticazione con il repository remoto

Per potersi autenticare, e quindi importare l'sdk in un progetto Android, vanno dichiarate il repository e le credenziali, fornite insieme all'sdk, nel **settings.gradle.kts**:

```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
        maven {
            url = uri("https://maven.pkg.github.com/pluservice/plus.wallet.sdk")
            credentials {
                username = "yourusername"
                password = "your_github_token"
            }
        }
    }
}
```

## 3.2 Autenticazione con il repository remoto

Una volta dichiarato il repository e le credenziali nel settings.gradle.kts, va aggiunta la dipendenza nel build.gradle.kts:

```
dependencies {
    implementation("net.pluservice:plus-wallet-sdk:1.0.0")

}
```

Ora sarà sufficiente sincronizzare il progetto con il gradle: File-> Sync Project With Gradle Files

# 4. Metodi e proprietà SDK

Nel seguente capitolo verranno mostrati i metodi e le proprietà esposte dall'sdk. L'app attraverso l'invocazione di questi metodi riuscirà a sincronizzarsi con il Server e a generare il contenuto firmato di un ticket attivo.

## 4.1 Metodo initialize

Inizializza l'Sdk e consente di interagire con le api successive. Senza inizializzazione le funzionalità dell'Sdk non saranno disponibili. Si richiede di inizializzare l'Sdk subito prima dei flussi che prevedono l'utilizzo del deviceToken/deviceId o per la generazione del qrCode.

**Firma del metodo iOS:**

```
func initialize(clientId: String, applicationKey: String) async throws
```

**Parametri:**

- **clientId:** stringa che rappresenta l'identificativo del cliente che sta utilizzando l'sdk.
- **applicationKey:** stringa che rappresenta la chiave richiesta per poter utilizzare l'sdk.

**Ritorna:**

Questo metodo asincrono ritorna void se l'esecuzione è andata a buon fine, altrimenti genera un'eccezione.

**Esempio di utilizzo iOS:**

```
Task {
    do {
        try await sdk.initialize(clientId: "client-id", applicationKey: "application-key")
    } catch {
        handleException(error)
    }
}
```

**Firma del metodo Android:**

```
@Throws(SDKException::class)
suspend fun initialize(context: Context, clientId: String, applicationKey: String)
```

**Parametro aggiuntivo:**

- **context:** interfaccia che rappresenta le informazioni globali del sistema Android. Serve all'sdk per completare l'inizializzazione.

**Esempio di utilizzo Android:**

```
lifecycleScope.launch {
    try {
        sdk.initialize(context, clientId = "client-id", applicationKey = "application-id")
    } catch (e: SDKException) {
        Log.e("Initialize Error", e.message)
    }
}
```

## 4.2 Metodo isInitialized

Questo metodo restituisce un booleano, che permette di verificare se l'Sdk è stato inizializzato correttamente.

**Firma del metodo iOS:**

```
func isInitialized() -> Bool
```

**Ritorna:**

Questo metodo ritorna un booleano che rappresenta la corretta inizializzazione dell'sdk con **true** o in caso contrario con **false**.

**Esempio di utilizzo iOS:**

```
let isInitialized = sdk.isInitalized()
if isInitialized {
    // SDK INITIALIZED CORRECTLY
}
```

**Firma del metodo Android:**

```
fun isInitialized(): Boolean
```

**Esempio utilizzo Android:**

```
val isInitialized = sdk.isInitialized()
if (isInitialized) {
    Log.i("SDK IS INITIALIZED")
}
```

## 4.3 Metodo synchronize

Questo metodo asincrono ha il compito di sincronizzare i dati con il server, se non riesce ad effettuare la sincronizzazione allora genera un Errore. Questo metodo va richiamato nei seguenti casi:

- A seguito dell'inizializzazione;
- Quando si passa dallo stato di background a quello di foreground e l'SDK è correttamente inizializzata (verificabile tramite il metodo isInitialized)

**Firma del metodo iOS:**

```
func synchronize() async throws
```

**Ritorna:**

Questo metodo asincrono ritorna void se l'esecuzione è andata a buon fine, altrimenti genera un'eccezione.

**Esempio di utilizzo iOS:**

```
Task {
    do {
        try await sdk.synchronize()
    } catch {
        handleException(error)
    }
}
```

**Firma del metodo Android:**

```
@Throws(SDKException::class)
suspend fun synchronize()
```

**Esempio di utilizzo Android:**

```
lifecycleScope.launch {
    try {
        sdk.synchronize()
    } catch (e: SDKException) {
        Log.e("Synchronize Error", e.message)
    }
}
```

## 4.4 Metodo getTicketToken

Questo metodo asincrono fornisce un token di autenticazione per invocare le api di:

- POST CSRTicketing/tickets/{pnr}/guest/redeem
- POST CSRTicketing/tickets/{pnr}/activate
- POST CSRTicketing/tickets/{pnr}/change-device
- POST CSRTicketing/tickets/{pnr}/validate

Il metodo potrebbe richiedere che il dispositivo sia online.

**Firma del metodo iOS:**

```
func getTicketActivationToken(ticketId: String) async throws -> String
```

**Parametri:**

- **ticketId**: stringa che rappresenta l'identificativo del ticket, ovvero il PNR

**Ritorna:**

Questo metodo asincrono ritorna una stringa che rappresenta il token firmato necessario per invocare i webservice descritti precedentemente.

**Esempio di utilizzo iOS:**

```
Task {
    do {
        let token = try await sdk.getTicketActivationToken(ticketId: "ticket-id", userId: "user-id")
```

```
        } catch {
            handleException(error)
        }
    }
```

**Firma del metodo Android:**

```
@Throws(SDKException::class)
suspend fun getActivationToken(ticketId: String): String
```

**Esempio di utilizzo Android:**

```
lifecycleScope.launch {
    try {
        val token = sdk.getActivationToken(ticketId = "ticket-id")
    } catch (e: SDKException) {
        Log.e("Get Activation Token Error", e.message)
    }
}
```

# 4.5 Metodo generateSignedTicketData

Questo metodo asincrono genera una stringa firmata in base64 con cui è possibile generare il qrcode da mostrare in fase di validazione. A seconda dello scenario potrebbe essere richiesta la connettività ad internet. Si rimanda ad una versione successiva la definizione formale delle specifiche operative su come generare un'immagine contenente il qrCode/Barcode (es. dimensioni qrCode, possibilità di aprirlo, dimensione da bordo screen, ecc.

**Firma del metodo iOS:**

```
func generateSignedTicketData(ticketId: String, userId: String) async throws -> String
```

**Parametri:**

- **ticketId:** stringa che rappresenta l'identificativo del ticket, ovvero il PNR
- **userId:** stringa che rappresenta l'identificativo dell'utente (da non passare in caso di Tdv anonimo)

**Esempio di utilizzo iOS:**

```
    Task {
        do {
            let signedQrCode = try await sdk.generateSignedTicketData(ticketId: "ticket-id", userId: "user-id")
        } catch {
            handleException(error)
        }
    }
```

**Firma del metodo Android:**

```
@Throws(SDKException::class)
suspend fun generateSignedTicketData(): String
```

**Esempio di utilizzo Android:**

```
lifecycleScope.launch {
    try {
        val signedData = sdk.generateSignedTicketData()
    } catch (e: SDKException) {
        Log.e("Generate Signed Ticket Data Error", e.message)
    }
}
```

## 4.6 Metodo getSubscriptionActivationToken

Questo metodo asincrono fornisce un token di autenticazione per invocare le api di attivazione di una tessera dell'utente. Il metodo potrebbe richiedere che il dispositivo sia online.

Questo metodo asincrono fornisce un token di autenticazione per invocare le api di:

- POST CSRTicketingSubscription/Account/ActivateCardDeviceAccount

**Firma del metodo iOS:**

```
func getSubscriptionActivationToken(userDataCode: String) async throws -> String
```

**Parametri:**

- **userDataCode**: stringa che rappresenta l'identificativo della tessera dell'utente, ovvero il cardCode. Tale campo è ottenuto dalle API del CSR Ticketing Subscription

**Ritorna:**

Questo metodo asincrono ritorna una stringa che rappresenta il token firmato necessario per invocare l'api di attivazione della tessera dell'utente sul dispositivo.

**Esempio di utilizzo iOS:**

```
    do {
        let token = try await sdk.getSubscriptionActivationToken(userDataCode: "user-data-code", userId: "user-id")
    } catch {
```

```
        handleException(error)
    }
```

**Firma del metodo Android:**

```
@Throws(SDKException::class)
suspend fun getSubscriptionActivationToken(userDataCode: String)
```

**Esempio di utilizzo Android:**

```
lifecycleScope.launch {
    try {
        val token = sdk.getSubscriptionActivationToken(userDataCode = "user-data-code")
    } catch (e: SDKException) {
        Log.e("Get Activation Token Error", e.message)
    }
}
```

# 4.7 Metodo getGuestOrdersToken

Questo metodo asincrono ritorna una stringa che rappresenta il token firmato necessario per invocare l'api di acquisto di un titolo di viaggio impersonale anonimo.

Questo metodo asincrono fornisce un token di autenticazione per invocare le api di:

- POST CSRTicketing/guest/order/issue

**Firma del metodo iOS:**

```
func getGuestOrdersToken() async throws -> String
```

**Esempio di utilizzo iOS:**

```
    Task {
        do {
            let token = try await sdk.getGuestOrdersToken()
        } catch {
            handleError(error)
        }
    }
```

**Firma del metodo Android:**

```
@Throws(SDKException::class)
suspend fun getGuestOrdersToken(): String
```

**Esempio di utilizzo Android:**

```
lifecycleScope.launch {
    try {
        val token = sdk.getGuestOrdersToken()
    } catch (e: SDKException) {
        Log.e("Get Guest Token Error", e.message)
    }
}
```

## 4.8 Metodo getDeviceId

Questo metodo ritorna una stringa contenente un identificativo univoco per il dispositivo in uso.

**Firma del metodo iOS:**

```
func getDeviceId() throws -> String
```

**Esempio di utilizzo iOS:**

```
    do {

        let deviceId = sdk.getDeviceId()

    } catch {

        handleError(error)

    }
```

**Firma del metodo Android:**

```
@Throws(SDKException::class)
fun getDeviceId(): String
```

**Esempio di utilizzo Android:**

```
lifecycleScope.launch {
    try {
        val deviceId = sdk.getDeviceId()
    } catch (e: SDKException) {
        Log.e("Get DeviceId Error", e.message)
    }
}
```

## 4.9 Proprietà version

Questa proprietà è una stringa, in sola lettura, che rappresenta la versione dell'sdk attualmente integrata.

**Firma della proprietà iOS:**

```
var version: String { get }
```

**Firma della proprietà Android:**

```
val version: String
```

# 5. Classi e metodi per la gestione delle eccezioni

Nel seguente capitolo verranno mostrate le classi di Errore e le sue proprietà, che permettono di interpretare le possibili eccezioni generate durante l'esecuzione di uno dei metodi descritti nei capitoli precedenti.

## 5.1 Classi di Errore

L'sdk mette a disposizione una serie di classi di errore che rappresentano l'errore generato durante l'invocazione di uno dei metodi dell'sdk.

Tutte le classi di errore, estendono la classe padre **SdkError** per iOS, **e** SdkException **per Android.**

**Proprietà iOS:**

```
var message: String

var code: Int
```

**Proprietà Android:**

override val message: String
val code: Int

La proprietà **message** è una stringa che riporta una descrizione testuale dell'errore che è stato generato, mentre **code** è un intero che rappresenta un codice specifico dell'errore.

### 5.1.1. NotInitializedError iOS – NotInitializedException Android

Questo errore/eccezione viene generato nel caso in cui viene invocato un metodo (tranne isInitialized) prima di aver chiamato il metodo **initialize**.

### 5.1.2. OfflineError iOS – OfflineException Android

Questo errore/eccezione viene generato nel caso in cui il dispositivo utente è offline e viene invocato un metodo che richiede invece la connessione a internet.

### 5.1.3. GenerateSignedDataError iOS – GenerateSignedDataException Android

Questo errore viene generato nel caso la generazione dei dati del ticket attivo non va a termine.

### 5.1.4. RetrieveTicketActivationTokenError iOS – RetrieveTicketActivationTokenException Android

Questo errore viene generato nel caso il recupero del token per un ticket attivo non va a termine.

### 5.1.5. RetrieveTicketSubscriptionTokenError iOS – RetrieveTicketSubscriptionTokenException Android

Questo errore viene generato nel caso il recupero del token per un abbonamento non va a termine.

### 5.1.6. RetrieveGuestOrdersTokenError iOS – RetrieveGuestOrdersTokenException Android

Questo errore viene generato nel caso il recupero del token per l'acquisto di un titolo impersonale non va a termine.

### 5.1.7. GenericError iOS – GenericException Android

Questo errore viene generato nel caso in cui si presenti un'eccezione che non rientra nei casi sopra descritti.

# 6. Clausola riservatezza

Le informazioni, i dati e le notizie contenute nella presente documentazione sono da ritenersi di natura strettamente confidenziale, privata e come tali sono riservate e comunque inviate esclusivamente ai destinatari indicati in epigrafe. La lettura, la diffusione, la distribuzione, la copiatura non autorizzata del documento - in toto e/o in parte - o qualsiasi altra azione derivante dalla conoscenza di queste informazioni sono rigorosamente vietate sia ai sensi dell'art. 616 c.p., sia ai sensi del D.Lgs. n. 196/2003 ed al Regolamento UE 2016/679 GDPR. Se si ritiene di non essere il destinatario di questa comunicazione o se la si è ricevuta per errore si prega di darne immediata comunicazione al mittente e di provvedere immediatamente alla sua distruzione.