Advanced StoreKit
Receipt validation and subscriptions

Pete Hare, App Store Engineer
Agenda

Receipt validation
Maintaining subscription state
Developing with the Sandbox
Agenda

Receipt validation

Maintaining subscription state

Developing with the Sandbox
Agenda

Receipt validation

Maintaining subscription state

Developing with the Sandbox
In-App Purchase Process
Processing transactions

1. Load In-App Identifiers
2. Fetch Product Info
3. Show In-App UI
4. Request Payment
5. Process Transaction
6. Unlock Content
7. Finish Transaction
In-App Purchase Process

Processing transactions

1. Load In-App Identifiers
2. Fetch Product Info
3. Show In-App UI
4. Request Payment
5. Process Transaction
6. Unlock Content
7. Finish Transaction
In-App Purchase Process
Processing transactions

- Process Transaction
- Unlock Content
- Finish Transaction
In-App Purchase Process
Processing transactions

Device

Receive Transaction → Validate Receipt → Update State/Unlock Content → Finish Transaction

Your Server

Receive Transaction → Validate Receipt → Update State/Unlock Content
In-App Purchase Process

Processing transactions

1. **Receive Transaction**
   - Device

2. **Validate Receipt**
   - Device

3. **Update State/Unlock Content**
   - Device

4. **Finish Transaction**
   - Device

5. **Update State/Unlock Content**
   - Your Server

6. **Validate Receipt**
   - Your Server

7. **Receive Transaction**
   - Your Server
In-App Purchase Process
Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

Validate Receipt

Update State/Unlock Content

Update State/Unlock Content

Finish Transaction
In-App Purchase Process

Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

Validate Receipt

Update State/Unlock Content

Update State/Unlock Content

Finish Transaction
In-App Purchase Process

Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

Validate Receipt

Update State/Unlock Content

Update State/Unlock Content

Finish Transaction
In-App Purchase Process
Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

Update State/Unlock Content

Finish Transaction

Validate Receipt

Update State/Unlock Content
In-App Purchase Process
Processing transactions

Your Server

Device

Receive Transaction → Validate Receipt → Update State/Unlock Content → Finish Transaction
In-App Purchase Process
Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

Validate Receipt

Update State/Unlock Content

Update State/Unlock Content

Finish Transaction
In-App Purchase Process

Processing transactions

Your Server

Receive Transaction → Validate Receipt → Update State/Unlock Content → Finish Transaction

Device

Validate Receipt → Update State/Unlock Content
In-App Purchase Process

Processing transactions

Your Server

Device

Receive Transaction → Validate Receipt → Update State/Unlock Content → Finish Transaction
import UIKit
import StoreKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, SKPaymentTransactionObserver {

    func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {
        SKPaymentQueue.default().add(self)
        return true
    }
}
import UIKit
import StoreKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, SKPaymentTransactionObserver {

    func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {
        SKPaymentQueue.default().add(self)
        return true
    }
}
// Start Observing the Payment Queue

import UIKit
import StoreKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate, SKPaymentTransactionObserver {

    func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {
        SKPaymentQueue.default().add(self)
        return true
    }
}
class AppDelegate: UIResponder, UIApplicationDelegate, SKPaymentTransactionObserver {

    // ...

    // MARK: – SKPaymentTransactionObserver

    func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
        for transaction in transactions {
            switch transaction.transactionState {
            case .purchased:
                // Validate the purchase
            }
        }
    }
}
class AppDelegate: UIResponder, UIApplicationDelegate, SKPaymentTransactionObserver {

    // ...

    // MARK: - SKPaymentTransactionObserver

    func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
        for transaction in transactions {
            switch transaction.transactionState {
            case .purchased:
                // Validate the purchase
            }
        }
    }
}
class AppDelegate: UIResponder, UIApplicationDelegate, SKPaymentTransactionObserver {

    // ...

    // MARK: - SKPaymentTransactionObserver

    func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
        for transaction in transactions {
            switch transaction.transactionState {
            case .purchased:
                // Validate the purchase
            }
        }
    }
}
class AppDelegate: UIResponder, UIApplicationDelegate, SKPaymentTransactionObserver {

    // ...

    // MARK: - SKPaymentTransactionObserver

    func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
        for transaction in transactions {
            switch transaction.transactionState {
            case .purchased:
                // Validate the purchase
            }
        }
    }
}
class AppDelegate: UIResponder, UIApplicationDelegate, SKPaymentTransactionObserver {

    // ...

    // MARK: - SKPaymentTransactionObserver

    func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
        for transaction in transactions {
            switch transaction.transactionState {
            case .purchased:
                // Validate the purchase
            }
        }
    }
}
In-App Purchase Process

Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

Validate Receipt

Update State/Unlock Content

Update State/Unlock Content

Finish Transaction

updatedTransactions()
In-App Purchase Process

Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

Validate Receipt

Update State/Unlock Content

Update State/Unlock Content

Finish Transaction

updatedTransactions()
The Receipt
The Receipt

Trusted record of app and in-app purchases
The Receipt

Trusted record of app and in-app purchases

Stored on device
The Receipt

Trusted record of app and in-app purchases

Stored on device

Issued by the App Store
The Receipt

Trusted record of app and in-app purchases

Stored on device

Issued by the App Store

Signed and verifiable
The Receipt

Trusted record of app and in-app purchases

Stored on device

Issued by the App Store

Signed and verifiable

For your app, on that device only
Receipt Validation

Your Server

Device

Validate Receipt

Validate Receipt
Receipt Validation

On-device validation

- Unlock features and content within the app
Receipt Validation

On-device validation
• Unlock features and content within the app

Server-to-server validation
• Online validation through a request to the App Store
• Unlock features/subscription state on your server
Receipt Validation

On-device validation
• Unlock features and content within the app

Server-to-server validation
• Online validation through a request to the App Store
• Unlock features/subscription state on your server

Do not use online validation directly from the device!
On-Device Receipt Validation
The basics

- Receipt
- Purchase Information
- Certificates
- Signature
On-Device Receipt Validation

The basics

Stored in the app bundle

- Receipt
- Purchase Information
- Certificates
- Signature
On-Device Receipt Validation
The basics

Stored in the app bundle
• API to get the path
On-Device Receipt Validation
The basics

Stored in the app bundle
• API to get the path

Single file
On-Device Receipt Validation

The basics

Stored in the app bundle
  • API to get the path

Single file
  • Purchase data

- Receipt
  - Purchase Information
  - Certificates
  - Signature
On-Device Receipt Validation

The basics

Stored in the app bundle
• API to get the path

Single file
• Purchase data
• Signature to check authenticity
On-Device Receipt Validation

Standards

Receipt

Purchase Information

Certificates

Signature
On-Device Receipt Validation

Standards

Signing
• PKCS#7 Cryptographic Container
On-Device Receipt Validation
Standards

Signing
• PKCS#7 Cryptographic Container

Data Encoding
• ASN.1
On-Device Receipt Validation

Standards

Signing
• PKCS#7 Cryptographic Container

Data Encoding
• ASN.1

Options for verifying and reading
• OpenSSL, asn1c, and more
• Create your own

Receipt
- Purchase Information
- Certificates
- Signature
On-Device Receipt Validation
The basics

Locate the receipt using Bundle API

```swift
// Locate the file
guard let url = Bundle.main.appStoreReceiptURL else {
    // Handle failure
    return
}

// Read the contents
let receipt = Data(contentsOf: url)
```
On-Device Receipt Validation
The basics

Locate the receipt using Bundle API

```swift
// Locate the file
guard let url = Bundle.main.appStoreReceiptURL else {
    // Handle failure
    return
}

// Read the contents
let receipt = Data(contentsOf: url)
```
On-Device Receipt Validation

The basics

Locate the receipt using Bundle API

```swift
// Locate the file
guard let url = Bundle.main.appStoreReceiptURL else {
    // Handle failure
    return
}

// Read the contents
let receipt = Data(contentsOf: url)
```
On-Device Receipt Validation
The basics

Locate the receipt using Bundle API

// Locate the file
guard let url = Bundle.main.appStoreReceiptURL else {
    // Handle failure
    return
}
// Read the contents
let receipt = Data(contentsOf: url)
On-Device Receipt Validation
Tips for using OpenSSL
On-Device Receipt Validation
Tips for using OpenSSL

Build your own static library (.a file)
• Not a dynamic library
On-Device Receipt Validation
Tips for using OpenSSL

Build your own static library (.a file)
• Not a dynamic library

Include Apple Root CA Certificate
• Available online
• If bundled in app, watch out for expiry
On-Device Receipt Validation

Tips for using OpenSSL

Build your own static library (.a file)
- Not a dynamic library

Include Apple Root CA Certificate
- Available online
- If bundled in app, watch out for expiry

Documentation online
On-Device Receipt Validation

Downloading pre-built solutions
On-Device Receipt Validation
Downloading pre-built solutions

Convenience comes at a price

• Reusing code brings with it bugs and vulnerabilities
• Single exploit affects many
On-Device Receipt Validation

Downloading pre-built solutions

Convenience comes at a price

• Reusing code brings with it bugs and vulnerabilities
• Single exploit affects many

It’s your revenue stream

• Make decisions that suit your product
• Know and own the risks
On-Device Receipt Validation
Certificate verification
On-Device Receipt Validation

Certificate verification

Do not check the expiry date of the certificate relative to current date
On-Device Receipt Validation
Certificate verification

- Do not check the expiry date of the certificate relative to current date

- Compare expiry date to purchase date of the transaction
On-Device Receipt Validation

Receipt payload

Purchase Information

Attribute
Type 2 Bundle Identifier
Type # Value
Type # Value

Certificates
Signature
On-Device Receipt Validation

Receipt payload

Series of attributes

• Type
• Value
• (Version)
On-Device Receipt Validation
Verify application

Receipt

Purchase Information

Attribute
Type 2 Bundle Identifier

Attribute
Type 3 Bundle Version
On-Device Receipt Validation
Verify application

Check the bundle identifier
On-Device Receipt Validation

Verify application

Check the bundle identifier

Check the bundle version
On-Device Receipt Validation

Verify application

Check the bundle identifier

Check the bundle version

Use hardcoded values
  • Not Info.plist values
On-Device Receipt Validation
Verify device

Receipt

Purchase Information

Attribute

Type 2 | Bundle Identifier

Type 3 | Bundle Version

Type 4 | Opaque Value

Type 5 | SHA-1 Hash
On-Device Receipt Validation
Verify device

Attribute 5 is a SHA-1 hash of three key values
• Bundle identifier
• Device identifier
• Opaque value (Attribute 4)
On-Device Receipt Validation
Verify device

Attribute 5 is a SHA-1 hash of three key values
• Bundle identifier
• Device identifier
• Opaque value (Attribute 4)

Unique to your app on this device
On-Device Receipt Validation

Verify device

Attribute 5 is a SHA-1 hash of three key values
• Bundle identifier
• Device identifier
• Opaque value (Attribute 4)

Unique to your app on this device

Create hash using hardcoded values, compare
In-App Purchase Process

Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

PKCS#7
ASN.1

Update State/
Unlock Content

Finish Transaction

updatedTransactions()
In-App Purchase Process
Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

Update State/Unlock Content

Finish Transaction

updatedTransactions()

PKCS#7
ASN.1
On-Device In-App Purchase State

Receipt

Purchase Information

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bundle Identifier</td>
<td>Type 2</td>
</tr>
<tr>
<td>In-App Purchase</td>
<td>Type 17</td>
</tr>
<tr>
<td>In-App Purchase</td>
<td>Type 17</td>
</tr>
<tr>
<td>In-App Purchase</td>
<td>Type 17</td>
</tr>
</tbody>
</table>
On-Device In-App Purchase State
In-app purchase attributes

In-App Purchase Receipt

Receipt
Purchase Information

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type 2</th>
<th>Bundle Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>Type 17</td>
<td>In-App Purchase</td>
</tr>
<tr>
<td>Attribute</td>
<td>Type 17</td>
<td>In-App Purchase</td>
</tr>
<tr>
<td>Attribute</td>
<td>Type 17</td>
<td>In-App Purchase</td>
</tr>
</tbody>
</table>
On-Device In-App Purchase State

In-app purchase attributes

In-App Purchase Receipt
Type 1701
Quantity

Receipt

Purchase Information

Attribute
Type 2
Bundle Identifier

Type 17
In-App Purchase

Type 17
In-App Purchase

Type 17
In-App Purchase

Type 17
In-App Purchase
On-Device In-App Purchase State

In-app purchase attributes

In-App Purchase Receipt

- Type 1701: Quantity
- Type 1702: Product Identifier

Receipt

Purchase Information

- Attribute
  - Type 2: Bundle Identifier
- Attribute
  - Type 17: In-App Purchase
- Attribute
  - Type 17: In-App Purchase
- Attribute
  - Type 17: In-App Purchase
On-Device In-App Purchase State

In-app purchase attributes

In-App Purchase Receipt

| Type 1701 | Quantity |
| Type 1702 | Product Identifier |
| Type 1703 | Transaction Identifier |

Receipt

Purchase Information

<table>
<thead>
<tr>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 2</td>
</tr>
<tr>
<td>Bundle Identifier</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 17</td>
</tr>
<tr>
<td>In-App Purchase</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 17</td>
</tr>
<tr>
<td>In-App Purchase</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 17</td>
</tr>
<tr>
<td>In-App Purchase</td>
</tr>
</tbody>
</table>
On-Device In-App Purchase State

In-app purchase attributes

In-App Purchase Receipt
- Type 1701: Quantity
- Type 1702: Product Identifier
- Type 1703: Transaction Identifier
- Type 1704: Purchase Date

Receipt

Purchase Information
- Attribute
  - Type 2: Bundle Identifier
  - Type 17: In-App Purchase
  - Type 17: In-App Purchase
On-Device In-App Purchase State

In-app purchase attributes

### In-App Purchase Receipt
- **Type 1701**: Quantity
- **Type 1702**: Product Identifier
- **Type 1703**: Transaction Identifier
- **Type 1704**: Purchase Date
- **Type 1708**: Subscription Expiry Date

### Purchase Information
- **Attribute**
  - **Type 2**: Bundle Identifier
- **Attribute**
  - **Type 17**: In-App Purchase
- **Attribute**
  - **Type 17**: In-App Purchase
- **Attribute**
  - **Type 17**: In-App Purchase
On-Device In-App Purchase State
In-app purchase attributes

In-App Purchase Receipt

<table>
<thead>
<tr>
<th>Type</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>1701</td>
<td>Quantity</td>
</tr>
<tr>
<td>1702</td>
<td>Product Identifier</td>
</tr>
<tr>
<td>1703</td>
<td>Transaction Identifier</td>
</tr>
<tr>
<td>1704</td>
<td>Purchase Date</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>1708</td>
<td>Subscription Expiry Date</td>
</tr>
</tbody>
</table>

Purchase Information

<table>
<thead>
<tr>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bundle Identifier</td>
</tr>
<tr>
<td>In-App Purchase</td>
</tr>
<tr>
<td>In-App Purchase</td>
</tr>
<tr>
<td>In-App Purchase</td>
</tr>
<tr>
<td>In-App Purchase</td>
</tr>
</tbody>
</table>

For more types:
Receipt Validation Programming Guide
On-Device In-App Purchase State
Unlocking content on-device
On-Device In-App Purchase State
Unlocking content on-device

Transaction will appear in `updatedTransactions` callback

• Transaction observer must be registered early in lifecycle!
On-Device In-App Purchase State
Unlocking content on-device

Transaction will appear in `updatedTransactions` callback

• Transaction observer must be registered early in lifecycle!

Receipt payload contains in-app purchase transactions

• Verify transaction in `updatedTransactions` callback is present in a receipt transaction
On-Device Subscription State
Does the user have an active subscription?
On-Device Subscription State
Does the user have an active subscription?

Valid receipt ≠ subscribed
On-Device Subscription State
Does the user have an active subscription?

Valid receipt ≠ subscribed

Filter transactions by originalTransactionId
• Matches the first in-app purchase for that subscription
On-Device Subscription State

Does the user have an active subscription?

Valid receipt ≠ subscribed

Filter transactions by `originalTransactionId`
  • Matches the first in-app purchase for that subscription

Check matching transactions for latest expiry date
  • Type 1708
On-Device Subscription State
Does the user have an active subscription?

Valid receipt ≠ subscribed

Filter transactions by `originalTransactionId`
• Matches the first in-app purchase for that subscription

Check matching transactions for latest expiry date
• Type 1708

If there’s no valid transaction, can refresh receipt
• Repeat above steps
On-Device Subscription State

Caveat
On-Device Subscription State

Caveat

For determining if subscription is valid, inspect

• Purchase date
• Expiry date
• System date
On-Device Subscription State

Caveat

For determining if subscription is valid, inspect

• Purchase date
• Expiry date
• System date

Caveat for on-device subscription state

• Device clock could be wound back!
On-Device Receipt Validation
Receipt Refresh on iOS
On-Device Receipt Validation

Receipt Refresh on iOS

If the receipt doesn’t exist or is invalid, refresh the receipt using StoreKit
On-Device Receipt Validation
Receipt Refresh on iOS

If the receipt doesn’t exist or is invalid, refresh the receipt using StoreKit

Receipt refresh will require network
On-Device Receipt Validation
Receipt Refresh on iOS

If the receipt doesn’t exist or is invalid, refresh the receipt using StoreKit

Receipt refresh will require network

Store sign-in will be required
On-Device Receipt Validation
Receipt Refresh on iOS

If the receipt doesn’t exist or is invalid, refresh the receipt using StoreKit.

Receipt refresh will require network.

Store sign-in will be required.

Avoid continuous loop of validate-and-refresh.
On-Device Receipt Validation
Receipt Refresh on iOS

If the receipt doesn’t exist or is invalid, refresh the receipt using StoreKit.

Receipt refresh will require network.

Store sign-in will be required.

Avoid continuous loop of validate-and-refresh.

```swift
let request = SKReceiptRefreshRequest()
request.delegate = self
request.start()
```
On-Device Receipt Validation
Receipt Refresh on macOS

If the receipt is invalid
On-Device Receipt Validation
Receipt Refresh on macOS

If the receipt is invalid
Receipt refresh will require network
On-Device Receipt Validation
Receipt Refresh on macOS

If the receipt is invalid
Receipt refresh will require network
Store sign-in will be required
On-Device Receipt Validation
Receipt Refresh on macOS

If the receipt is invalid
Receipt refresh will require network
Store sign-in will be required
Exit with code 173 to refresh receipt
On-Device Receipt Validation
Receipt Refresh on macOS

If the receipt is invalid
Receipt refresh will require network
Store sign-in will be required
Exit with code 173 to refresh receipt

// Receipt is invalid
exit(173)
Receipt Tips

Restoring transactions vs. refreshing receipt
Receipt Tips
Restoring transactions vs. refreshing receipt

restoreCompletedTransactions()  SKReceiptRefreshRequest()
Receipt Tips
Restoring transactions vs. refreshing receipt

```swift
restoreCompletedTransactions()
```

SKReceiptRefreshRequest()

API on SKPaymentQueue
Receipt Tips
Restoring transactions vs. refreshing receipt

```
restoreCompletedTransactions()

API on SKPaymentQueue

SKReceiptRefreshRequest()

Create request instance, call start()
```
Receipt Tips

Restoring transactions vs. refreshing receipt

```
restoreCompletedTransactions()
```

API on `SKPaymentQueue`

Completed transactions appear in `updatedTransactions()`

```
SKReceiptRefreshRequest()
```

Create request instance, call `start()`

New receipt document is downloaded onto device
Receipt Tips
Restoring transactions vs. refreshing receipt

**API on `SKPaymentQueue`**

Completed transactions appear in `updatedTransactions()`.

Restores:
- Non-consumable
- Auto-renewable subscriptions

**`restoreCompletedTransactions()`**

**`SKReceiptRefreshRequest()`**

Create request instance, call `start()`.

New receipt document is downloaded onto device.
Receipt Tips
Restoring transactions vs. refreshing receipt

**restoreCompletedTransactions()**

API on `SKPaymentQueue`

Completed transactions appear in `updatedTransactions()`

Restores:
- Non-consumable
- Auto-renewable subscriptions

**SKReceiptRefreshRequest()**

Create request instance, call `start()`

New receipt document is downloaded onto device

Restores:
- Non-consumable
- Non-renewing subscriptions
- Auto-renewable subscriptions
Receipt Tips
Switching to subscriptions

Receipt
Purchase Information
Attribute
Type 19
Original Application Version
Receipt Tips
Switching to subscriptions

Original application version in receipt
Receipt Tips

Switching to subscriptions

Original application version in receipt

Know whether the app was purchased as paid version, or is a subscription version
In-App Purchase Process

Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

Validate Receipt

PKCS#7
ASN.1

Updated Transactions()

Update State/Unlock Content

Update State/Unlock Content

Type 17: ...

Type 17: ...

Finish Transaction
In-App Purchase Process

Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

Update State/Unlock Content

Finish Transaction

updatedTransactions()

PKCS#7
ASN.1

Type 17: ...

Type 17: ...
In-App Purchase Process
Finish the transaction
In-App Purchase Process

Finish the transaction

Finish all transactions once content is unlocked

- If downloading hosted content, wait until after the download completes
In-App Purchase Process

Finish the transaction

Finish all transactions once content is unlocked

• If downloading hosted content, wait until after the download completes

Includes all auto-renewable subscription transactions
In-App Purchase Process
Finish the transaction

Finish all transactions once content is unlocked
• If downloading hosted content, wait until after the download completes

Includes all auto-renewable subscription transactions

Otherwise, the payment will stay in the queue
In-App Purchase Process
Finish the transaction

Finish all transactions once content is unlocked
• If downloading hosted content, wait until after the download completes

Includes all auto-renewable subscription transactions

Otherwise, the payment will stay in the queue

Subscription billing retry depends on up-to-date information about transaction
In-App Purchase Process
Finish the transaction

Finish all transactions once content is unlocked
• If downloading hosted content, wait until after the download completes

Includes all auto-renewable subscription transactions

Otherwise, the payment will stay in the queue

Subscription billing retry depends on up-to-date information about transaction
In-App Purchase Process
Processing transactions

Your Server

Device

updatedTransactions()

PKCS#7
ASN.1

Validate Receipt

Update State/
Unlock Content

Finish Transaction

Type 17: …
In-App Purchase Process

Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

Validate Receipt

Update State/Unlock Content

Update State/Unlock Content

Finish Transaction

updatedTransactions()

PKCS#7 ASN.1

Type 17: ...

Type 17: ...

finishTransaction()
Server-Side Receipt Validation

Server-to-server validation

App Store

Your Server

User
Server-Side Receipt Validation
Server-to-server validation

updatedTransactions()
Server-Side Receipt Validation
Server-to-server validation

App Store → User
updatedTransactions() → Your Server
Server-Side Receipt Validation

Server-to-server validation

App Store

/verifyReceipt

Your Server

updatedTransactions()
Server-Side Receipt Validation

Server-to-server validation

Diagram:
- App Store
- /verifyReceipt
- Your Server
- User
- updatedTransactions()
Server-Side Receipt Validation
Server-to-server validation

Allows your servers to validate the receipt

Your app sends the receipt to your servers
• Your server sends the receipt to the App Store
• Endpoint is `verifyReceipt`

Response is in JSON
• Returns status on whether receipt is valid or not
Server-Side Receipt Validation

Server-to-server validation

Allows your servers to validate the receipt

Your app sends the receipt to your servers

• Your server sends the receipt to the App Store
• Endpoint is `verifyReceipt`

Response is in JSON

• Returns status on whether receipt is valid or not

Never send the receipt directly from your app to the App Store server
In-App Purchase Process
Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

finishTransaction()
In-App Purchase Process

Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

/verifyReceipt

Update State/Unlock Content

Finish Transaction

updatedTransactions()

PKCS#7
ASN.1

Type 17: ...

PKCS#7
ASN.1

Type 17: ...

finishTransaction()
Server-Side In-App Purchase State

Verifying a purchase on your server

App Store

Your Server

updatedTransactions()
Server-Side In-App Purchase State

Verifying a purchase on your server

App Store → VerifyReceipt

Your Server → updatedTransactions()
Server-Side In-App Purchase State

Verifying a purchase on your server

App Store /verifyReceipt

updatedTransactions()
Server-Side In-App Purchase State

Verifying a purchase on your server

`/verifyReceipt`

`updatedTransactions()`
Server-Side In-App Purchase State

Verifying a purchase on your server

```
updatedTransactions()
```

```
/verifyReceipt
```

Flow diagram showing interaction between App Store, Your Server, and the client device.
Server-Side In-App Purchase State

Verifying a purchase on your server

- App Store
- Your Server
- Website

/verifyReceipt
updatedTransactions()
Server-Side In-App Purchase State
Unlocking content on your server

In addition to receipt validity, `verifyReceipt` returns:
- Latest decoded application receipt
- Contains array of in-app purchase transactions
  - Verify product in the `updatedTransactions` callback on device is present in a transaction

Tell the device to `finishTransaction()`
Server-Side In-App Purchase State

Does the user have an active subscription?
Server-Side In-App Purchase State

Does the user have an active subscription?

Filter transactions by `originalTransactionId`
- Matches the first in-app purchase for that subscription
Server-Side In-App Purchase State

Does the user have an active subscription?

Filter transactions by `originalTransactionId`

- Matches the first in-app purchase for that subscription

Check matching transactions for latest expiry date
In-App Purchase Process
Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

PKCS#7
ASN.1

Update State/
Unlock Content

transactions:[]

Finish Transaction

updatedTransactions()

finishTransaction()
**In-App Purchase Process**

Processing transactions

- **Device**:
  - Receive Transaction
    - `updatedTransactions()`
    - Validate Receipt
      - `/verifyReceipt`
      - `transactions:[]`
      - PKCS#7
      - ASN.1
    - Validate Receipt
    - Update State/Unlock Content
      - Type 17: ...
    - Update State/Unlock Content
  - Finish Transaction
    - `finishTransaction()`

- **Your Server**:
  - Validate Receipt
  - Update State/Unlock Content
Server-Side Subscription State

App Store

updatedTransactions()

Your Server
Server-Side Subscription State

App Store

Your Server

updatedTransactions()
Server-Side Subscription State

App Store

/verifyReceipt

updatedTransactions()

Your Server
Server-Side Subscription State

- App Store
- Your Server

/verifyReceipt

updatedTransactions()
Server-Side Subscription State

- App Store
- Your Server

/verifyReceipt

updatedTransactions()
Server-Side Subscription State

- **App Store**
- **Your Server**
- **Website**

Flow:
- **App Store** sends a receipt to **Your Server** via /verifyReceipt.
- **Your Server** responds with a green checkmark.
- **Website** updates transactions through `updatedTransactions()` function.
Server-Side Subscription State

Polling for subscription state

- App Store
- Your Server

updatedTransactions()
Server-Side Subscription State
Polling for subscription state

- App Store
- Your Server
- Website

/verifyReceipt
updatedTransactions()
Server-Side Subscription State
Polling for subscription state

App Store
/verifyReceipt
updatedTransactions()

Your Server

Website
Server-Side Subscription State
Polling for subscription state

App Store

Your Server

Website

updatedTransactions()
Server-Side Subscription State

Polling for subscription state

App Store

/verifyReceipt

Your Server

updatedTransactions()

Website
Server-Side Subscription State
Polling for subscription state

- App Store
- Your Server
- Website

`updatedTransactions()`
Server-Side In-App Purchase State
For auto-renewable subscriptions
Server-Side In-App Purchase State
For auto-renewable subscriptions

Treat the receipt data like a “token”
Server-Side In-App Purchase State
For auto-renewable subscriptions

Treat the receipt data like a “token”
• Store on your own server
Server-Side In-App Purchase State
For auto-renewable subscriptions

Treat the receipt data like a "token"
• Store on your own server
• Same binary data, multiple times
Server-Side In-App Purchase State
For auto-renewable subscriptions

Treat the receipt data like a "token"
• Store on your own server
• Same binary data, multiple times

Propagate subscription state across devices
Server-Side In-App Purchase State
For auto-renewable subscriptions

Treat the receipt data like a “token”
• Store on your own server
• Same binary data, multiple times

Propagate subscription state across devices

Still need to process all `updateTransactions()`
Server-Side In-App Purchase State
For auto-renewable subscriptions

Treat the receipt data like a “token”
• Store on your own server
• Same binary data, multiple times

Propagate subscription state across devices

Still need to process all `updateTransactions()`
• Including all renewal transactions
Server-Side In-App Purchase State
For auto-renewable subscriptions

Treat the receipt data like a “token”
• Store on your own server
• Same binary data, multiple times

Propagate subscription state across devices
Still need to process all `updateTransactions()`
• Including all renewal transactions
• All the way to `finishTransaction()`
Server-Side In-App Purchase State
Optimizing app receipts
Server-Side In-App Purchase State
Optimizing app receipts

App receipts include all subscription renewals
Server-Side In-App Purchase State

Optimizing app receipts

App receipts include all subscription renewals

Many developers only care about latest
Server-Side In-App Purchase State
Optimizing app receipts

App receipts include all subscription renewals

Many developers only care about latest

Optional query parameter `exclude-old-transactions`
  • true: only return latest transaction
  • false: include all transactions (default)
In-App Purchase Process

Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

/verifyReceipt

transactions:[]

Update State/Unlock Content

Unlock Content

Finish Transaction

updatedTransactions()

PKCS#7 ASN.1

Type 17: ...

Type 17: ...

finishTransaction()
In-App Purchase Process

Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

/verifyReceipt

transactions:[]

Update State/Unlock Content

Finish Transaction

PKCS#7
ASN.1

updatedTransactions()

finishTransaction()
Server-Side Subscriptions

Why did a user’s subscription expire?
Server-Side Subscriptions

Why did a user’s subscription expire?

Will this user’s subscription be renewed?
Server-Side Subscriptions

- Why did a user’s subscription expire?
- Will this user’s subscription be renewed?
- Will the user be downgraded after this billing cycle?
Server-Side Subscriptions

Why did a user’s subscription expire?

Why did AppleCare give the user a refund?

Will this user’s subscription be renewed?

Will the user be downgraded after this billing cycle?
Server-Side Subscriptions

- Why did a user’s subscription expire?
- Will this user’s subscription be renewed?
- Why did AppleCare give the user a refund?
- Has the user agreed to a price increase?
- Will the user be downgraded after this billing cycle?
Server-Side Subscriptions

- Why did a user’s subscription expire?
- What does the user need to know about their subscription?
- Will this user’s subscription be renewed?
- Why did AppleCare give the user a refund?
- Has the user agreed to a price increase?
- Will the user be downgraded after this billing cycle?
Why?
Subscription Churn
Losing subscribers
Subscription Churn
Losing subscribers

Every subscription that lapses is *lost revenue*
Subscription Churn
Losing subscribers

Every subscription that lapses is lost revenue

Involuntary
Subscription Churn
Losing subscribers

Every subscription that lapses is lost revenue

Involuntary
• Credit card expires
Subscription Churn

Losing subscribers

Every subscription that lapses is **lost revenue**

Involuntary
• Credit card expires

Voluntary
Subscription Churn
Losing subscribers

Every subscription that lapses is lost revenue

Involuntary
• Credit card expires

Voluntary
• Turns off auto-renew
Subscription Churn
Losing subscribers

Every subscription that lapses is lost revenue

Involuntary
• Credit card expires

Voluntary
• Turns off auto-renew
• Receives a refund from AppleCare
Subscription Churn
Losing subscribers

Every subscription that lapses is **lost revenue**

Involuntary
- Credit card expires

Voluntary
- Turns off auto-renew
- Receives a refund from AppleCare

**New tools to help reduce churn**
Example

App Store

Your Server

Example
Example

App Store → Your Server
Example

App Store

Your Server

✔️
Example

App Store

Your Server

Credit Card

Checkmark
Example

- App Store
- Your Server
Example

App Store

Your Server

Website
Example

Sorry! Your subscription has expired

Close
Example

Sorry! Your subscription has expired.

Close
Example

Application Store

Your Server

Website

Sorry! Your subscription has expired
Close
Example

Sorry! Your subscription has expired

Close
Server Notifications

- App Store
- Your Server

Sorry! Your subscription has expired

Close
Server Notifications

App Store

Your Server

Website

Sorry! Your subscription has expired

Close

NEW
Server Notifications

App Store → Your Server

Sorry! Your subscription has expired

Close
Server Notifications

App Store

Your Server

Website

NEW
Server Notifications

App Store

Your Server

Website

😃
Server Notifications

Status URL in iTunes Connect
Server Notifications

Status URL in iTunes Connect

App Transport Security requirements apply
Server Notifications

Status URL in iTunes Connect

App Transport Security requirements apply

HTTP POST to your server for status changes
Server Notifications

Status URL in iTunes Connect

App Transport Security requirements apply

HTTP POST to your server for status changes

• Initial purchase of a subscription
Server Notifications

Status URL in iTunes Connect

App Transport Security requirements apply

HTTP POST to your server for status changes
  • Initial purchase of a subscription
  • Subscription cancellation by AppleCare
Server Notifications

Status URL in iTunes Connect

App Transport Security requirements apply

HTTP POST to your server for status changes

• Initial purchase of a subscription
• Subscription cancellation by AppleCare
• Subscription downgrades
Server Notifications

Status URL in iTunes Connect

App Transport Security requirements apply

HTTP POST to your server for status changes

- Initial purchase of a subscription
- Subscription cancellation by AppleCare
- Subscription downgrades
- Successful renewal or re-purchase for an expired subscription
Server Notifications

Status URL in iTunes Connect

App Transport Security requirements apply

HTTP POST to your server for status changes

• Initial purchase of a subscription
• Subscription cancellation by AppleCare
• Subscription downgrades
• Successful renewal or re-purchase for an expired subscription

Includes latest transaction for the transaction in question
Server Notifications

No longer need to poll verifyReceipt every day
Server Notifications

No longer need to poll verifyReceipt every day

Still call verifyReceipt to update your state
Server Notifications

No longer need to poll `verifyReceipt` every day

Still call `verifyReceipt` to update your state

- App Store notification may not reach your server
Server Notifications

No longer need to poll verifyReceipt every day
Still call verifyReceipt to update your state
• App Store notification may not reach your server
• Be smart about when to poll
Server Notifications

No longer need to poll verifyReceipt every day

Still call verifyReceipt to update your state

- App Store notification may not reach your server
- Be smart about when to poll
  - Day before, day of expiry date
Server Notifications

No longer need to poll verifyReceipt every day
Still call verifyReceipt to update your state
• App Store notification may not reach your server
• Be smart about when to poll
  - Day before, day of expiry date

Notifications coming later this year
Server-Side Subscriptions
Server-Side Subscriptions

Why did a user’s subscription expire?

What does the user need to know about their subscription?

Will this user’s subscription be renewed?

Why did AppleCare give the user a refund?

Has the user agreed to a price increase?

Will the user be downgraded after this billing cycle?
Server-Side Subscriptions
New fields in verifyReceipt endpoint
Server-Side Subscriptions
New fields in verifyReceipt endpoint

- Initial Subscription
- Turns Off Auto-Renew
- Enters Subscription Retry Window
- Updates Invalid Payment Info
- Upgrades Subscription
- Agrees to Price Increase
- Cancels and Receives Refund
Server-Side Subscriptions

New fields in verifyReceipt endpoint
Server-Side Subscriptions

New fields in verifyReceipt endpoint

Now included in JSON from verifyReceipt
Server-Side Subscriptions
New fields in verifyReceipt endpoint

Now included in JSON from verifyReceipt

• Auto-Renew Status
Server-Side Subscriptions

New fields in verifyReceipt endpoint

Now included in JSON from verifyReceipt

• Auto-Renew Status
• Auto-Renew Preference
Server-Side Subscriptions
New fields in verifyReceipt endpoint

Now included in JSON from verifyReceipt
• Auto-Renew Status
• Auto-Renew Preference
• Price Consent Status
Server-Side Subscriptions
New fields in verifyReceipt endpoint

Now included in JSON from verifyReceipt

- Auto-Renew Status
- Auto-Renew Preference
- Price Consent Status
- Subscription Retry Flag
Server-Side Subscriptions
New fields in verifyReceipt endpoint

Now included in JSON from verifyReceipt
• Auto-Renew Status
• Auto-Renew Preference
• Price Consent Status
• Subscription Retry Flag
• Expiration Intent
Server-Side Subscriptions
New fields in verifyReceipt endpoint

Now included in JSON from verifyReceipt

- Auto-Renew Status
- Auto-Renew Preference
- Price Consent Status
- Subscription Retry Flag
- Expiration Intent
- Cancellation Reason
Keeping Your Subscribers

Involuntary expiration
Keeping Your Subscribers

Involuntary expiration

Expanded retry window up to 60 days
Keeping Your Subscribers

Involuntary expiration

Expanded retry window up to 60 days

Use the Expiration Intent and Subscription Retry Flag to
Keeping Your Subscribers
Involuntary expiration

Expanded retry window up to 60 days

Use the Expiration Intent and Subscription Retry Flag to

1. Provide messaging to user
Keeping Your Subscribers
Involuntary expiration

Expanded retry window up to 60 days

Use the Expiration Intent and Subscription Retry Flag to

1. Provide messaging to user
2. Offer a downgraded or temporary experience
Keeping Your Subscribers
Involuntary expiration

Expanded retry window up to 60 days

Use the Expiration Intent and Subscription Retry Flag to

1. Provide messaging to user
2. Offer a downgraded or temporary experience
3. Unblock your user immediately if subscription renews
Keeping Your Subscribers
Voluntary expiration
Keeping Your Subscribers
Voluntary expiration

Use the Expiration Intent field to
Keeping Your Subscribers

Voluntary expiration

Use the Expiration Intent field to

• Offer messaging to user
Keeping Your Subscribers

Voluntary expiration

Use the Expiration Intent field to
  • Offer messaging to user

User canceled?
Keeping Your Subscribers
Voluntary expiration

Use the **Expiration Intent** field to

- Offer messaging to user

User canceled?

- Apply win-back
Keeping Your Subscribers
Voluntary expiration

Use the Expiration Intent field to
• Offer messaging to user

User canceled?
• Apply win-back

Did not consent to higher price?
Keeping Your Subscribers
Voluntary expiration

Use the Expiration Intent field to
• Offer messaging to user

User canceled?
• Apply win-back

Did not consent to higher price?
• Offer attractive downgrade option
Server-Side Subscriptions

New fields in verifyReceipt endpoint
Server-Side Subscriptions

New fields in verifyReceipt endpoint

• Auto-Renew Status
• Auto-Renew Preference
• Price Consent Status
• Subscription Retry Flag
• Expiration Intent
• Cancellation Reason
In-App Purchase Process

Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

/verifyReceipt

Validate Receipt

Update State/Unlock Content

transactions:[]

Update State/Unlock Content

Finish Transaction

PKCS#7
ASN.1

Type 17: ...

Type 17: ...

updatedTransactions()
In-App Purchase Process

Processing transactions

Your Server

Device

Receive Transaction

Validate Receipt

/verifyReceipt

transactions:[]

Validate Receipt

Update State/
Unlock Content

Update State/
Unlock Content

Finish Transaction

Server Notifications
Detailed Subscription Info
Status Polling

updatedTransactions()

PKCS#7
ASN.1

Type 17: ...

Type 17: ...

finishTransaction()
Auto-Renewable Subscriptions
Free trials
Auto-Renewable Subscriptions

Free trials

Subscription begins immediately
Auto-Renewable Subscriptions

Free trials

Subscription begins immediately

Not billed until free trial period is over
## Free Trials

### Previously

<table>
<thead>
<tr>
<th>Subscription Length</th>
<th>Available Free Trial Duration</th>
<th>Available Bonus Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 days</td>
<td>7 days</td>
<td>7 days</td>
</tr>
<tr>
<td>1 month</td>
<td>7 days, 1 month</td>
<td>7 days, 1 month</td>
</tr>
<tr>
<td>2 months</td>
<td>7 days, 1 month</td>
<td>7 days, 1 month</td>
</tr>
<tr>
<td>3 months</td>
<td>1 month</td>
<td>1 month</td>
</tr>
<tr>
<td>6 months</td>
<td>1 month, 2 months</td>
<td>1 month, 2 months</td>
</tr>
<tr>
<td>1 year</td>
<td>1 month, 2 months, 3 months</td>
<td>1 month, 2 months, 3 months</td>
</tr>
</tbody>
</table>
# Free Trials

Recent changes to trial durations

<table>
<thead>
<tr>
<th>Subscription Length</th>
<th>Available Free Trial Duration</th>
<th>Available Bonus Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>3 days</td>
<td>3 days</td>
</tr>
<tr>
<td></td>
<td>1 week</td>
<td>1 week</td>
</tr>
<tr>
<td></td>
<td>2 weeks</td>
<td>2 weeks</td>
</tr>
<tr>
<td></td>
<td>1 month</td>
<td>1 month</td>
</tr>
<tr>
<td></td>
<td>2 months</td>
<td>2 months</td>
</tr>
<tr>
<td></td>
<td>3 months</td>
<td>3 months</td>
</tr>
<tr>
<td></td>
<td>6 months</td>
<td>6 months</td>
</tr>
<tr>
<td></td>
<td>1 year</td>
<td>1 year</td>
</tr>
</tbody>
</table>
# Free Trials

Recent changes to trial durations

<table>
<thead>
<tr>
<th>Subscription Length</th>
<th>Available Free Trial Duration</th>
<th>Available Bonus Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>3 days</td>
<td>3 days</td>
</tr>
<tr>
<td></td>
<td>1 week</td>
<td>1 week</td>
</tr>
<tr>
<td></td>
<td>2 weeks</td>
<td>2 weeks</td>
</tr>
<tr>
<td></td>
<td>1 month</td>
<td>1 month</td>
</tr>
<tr>
<td></td>
<td>2 months</td>
<td>2 months</td>
</tr>
<tr>
<td></td>
<td>3 months</td>
<td>3 months</td>
</tr>
<tr>
<td></td>
<td>6 months</td>
<td>6 months</td>
</tr>
<tr>
<td></td>
<td>1 year</td>
<td>1 year</td>
</tr>
</tbody>
</table>
Developing with the Sandbox
The Sandbox
The Sandbox

Environment for testing purchases during development
The Sandbox

Environment for testing purchases during development

Based on certificate used to sign your application
The Sandbox

Environment for testing purchases during development

Based on certificate used to sign your application

Developer Certificate
The Sandbox

Environment for testing purchases during development

Based on certificate used to sign your application
The Sandbox

Environment for testing purchases during development

Based on certificate used to sign your application
The Sandbox

Environment for testing purchases during development

Based on certificate used to sign your application

Sandbox

Production

Certificate

Developer Certificate

App Store / Production Certificate
The Sandbox
Differences
The Sandbox
Differences

No charge
The Sandbox

Differences

No charge

Different endpoint for server-to-server validation for App Store Sandbox servers
The Sandbox

Differences

No charge

Different endpoint for server-to-server validation for App Store Sandbox servers

Can request expired and/or revoked receipts
The Sandbox
Differences

No charge

Different endpoint for server-to-server validation for App Store Sandbox servers

Can request expired and/or revoked receipts

Time contraction for subscriptions
# Subscription Timing

## Condensed intervals

<table>
<thead>
<tr>
<th>Face Value</th>
<th>Actual Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Days</td>
<td>3 minutes</td>
</tr>
<tr>
<td>1 Month</td>
<td>5 minutes</td>
</tr>
<tr>
<td>2 Months</td>
<td>10 minutes</td>
</tr>
<tr>
<td>3 Months</td>
<td>15 minutes</td>
</tr>
<tr>
<td>6 Months</td>
<td>30 minutes</td>
</tr>
<tr>
<td>1 Year</td>
<td>60 minutes</td>
</tr>
</tbody>
</table>
## Subscription Timing

Condensed intervals

<table>
<thead>
<tr>
<th>Face Value</th>
<th>Actual Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Days</td>
<td>3 minutes</td>
</tr>
<tr>
<td>1 Month</td>
<td>5 minutes</td>
</tr>
<tr>
<td>2 Months</td>
<td>10 minutes</td>
</tr>
<tr>
<td>3 Months</td>
<td>15 minutes</td>
</tr>
<tr>
<td>6 Months</td>
<td>30 minutes</td>
</tr>
<tr>
<td>1 Year</td>
<td>60 minutes</td>
</tr>
</tbody>
</table>

Maximum 6 renewals per 8-hour window
The Sandbox
Setting up the test environment
The Sandbox
Setting up the test environment

Setup in iTunes Connect
• Create test user
• Enter products for sale
The Sandbox
Setting up the test environment

Setup in iTunes Connect
• Create test user
• Enter products for sale

Build and sign your app
The Sandbox
Setting up the test environment

Setup in iTunes Connect
• Create test user
• Enter products for sale

Build and sign your app

Buy products
• Sign in with test user when prompted
Setup in iTunes Connect
• Create test user
• Enter products for sale

Build and sign your app

Buy products
• Sign in with test user when prompted

macOS only: Launch app from Finder once to fetch receipt
The Sandbox
Using the Sandbox from server
The Sandbox
Using the Sandbox from server

Development

App Store Sandbox → Your Test Server

Dev Signed App

Production

App Store → Your Prod Server

Prod Signed App
The Sandbox
Using the Sandbox from server

Development
- App Store Sandbox
  - Your Test Server
    - Dev Signed App

Production
- App Store
  - Your Prod Server
    - Prod Signed App

App Review
- App Store Sandbox
  - Your Prod Server
    - Prod Signed App
The Sandbox
App review considerations

App Review

- App Store Sandbox
- Your Prod Server
- Prod Signed App
The Sandbox

App review considerations

Try the production environment
The Sandbox
App review considerations

Try the production environment

If receipt is from Sandbox
  • Receive error 21007
The Sandbox
App review considerations

Try the production environment

If receipt is from Sandbox
• Receive error 21007

Then try against Sandbox
The Sandbox
Server notifications
The Sandbox
Server notifications

Not separate production and Sandbox servers
The Sandbox
Server notifications

Not separate production and Sandbox servers

Sandboxing is handled by a parameter in the payload
The Sandbox
Server notifications

Not separate production and Sandbox servers

Sandboxing is handled by a parameter in the payload

- Use the `environment` key in payload
Summary
Summary

Receipt Validation
Summary

Receipt Validation

Maintaining Subscription State
Summary

Receipt Validation

Maintaining Subscription State

Server-to-Server Notifications
Summary

Receipt Validation
Maintaining Subscription State
Server-to-Server Notifications
New Receipt Fields
Summary

Receipt Validation
Maintaining Subscription State
Server-to-Server Notifications
New Receipt Fields
Easy Steps to Retain Subscribers
Summary

Receipt Validation
Maintaining Subscription State
Server-to-Server Notifications
New Receipt Fields
Easy Steps to Retain Subscribers
Developing with the Sandbox
More Information

https://developer.apple.com/wwdc17/305
## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>WWDC 2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s New in iTunes Connect</td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>Introducing the New App Store</td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>What's New in StoreKit</td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>Lab</td>
<td>Location</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>StoreKit Lab</td>
<td>Technology Lab E</td>
</tr>
<tr>
<td>StoreKit Lab</td>
<td>Technology Lab E</td>
</tr>
</tbody>
</table>