What’s New in StoreKit

Session 303

Pete Hare, App Store Engineer
Ross LeBeau, App Store Engineer
What’s New in StoreKit
What’s New in StoreKit

Promoting in-app purchases
What’s New in StoreKit

Promoting in-app purchases

Server-to-server subscription notifications
What’s New in StoreKit

Promoting in-app purchases

Server-to-server subscription notifications

Detailed subscription status information
What’s New in StoreKit

Promoting in-app purchases
Server-to-server subscription notifications
Detailed subscription status information
Responding to reviews
What’s New in StoreKit

Promoting in-app purchases

Server-to-server subscription notifications

Detailed subscription status information

Responding to reviews

Asking for ratings and reviews
Review of In-App Purchases
In-App Purchase Overview
In-App Purchase Overview

Digital content or service bought in-app
In-App Purchase Overview

Digital content or service bought in-app
Not for physical goods
Types of In-App Purchases
Types of In-App Purchases

Consumable products
Types of In-App Purchases

Consumable products
Non-consumable products
Types of In-App Purchases

Consumable products
Non-consumable products
Non-renewing subscriptions
Types of In-App Purchases

- Consumable products
- Non-consumable products
- Non-renewing subscriptions
- Auto-renewable subscriptions
Types of In-App Purchases

Consumable products
Non-consumable products
Non-renewing subscriptions
Auto-renewable subscriptions
Types of In-App Purchases

- Consumable products
- Non-consumable products
- Non-renewing subscriptions
- Auto-renewable subscriptions

Advanced StoreKit

Grand Ballroom A
Thursday 1:50PM
In-App Purchase Process

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

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

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

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

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

Load In-App Identifiers → Fetch Product Info → Show In-App UI → Request Payment → Process Transaction → Unlock Content → Finish Transaction
In-App Purchase Process

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

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
Load In-App Identifiers
Options for storing the list of product identifiers
Load In-App Identifiers
Options for storing the list of product identifiers

After setting up product identifiers in iTunes Connect
Load In-App Identifiers
Options for storing the list of product identifiers

After setting up product identifiers in iTunes Connect

Baked into your app

```swift
let identifiers = 
"com.myCompany.myApp.product1",
"com.myCompany.myApp.product2"
```
Load In-App Identifiers
Options for storing the list of product identifiers

After setting up product identifiers in iTunes Connect

Baked into your app

```swift
let identifiers = [
"com.myCompany.myApp.product1",
"com.myCompany.myApp.product2"
]
```

Or fetch from your server

```swift
let identifiers = remoteIdentifiers()
```
In-App Purchase Process

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
// Fetch Product Info

let request = SKProductsRequest(productIdentifiers: identifierSet)
request.delegate = self
request.start()
// Fetch Product Info

let request = SKProductsRequest(productIdentifiers: identifierSet)
request.delegate = self
request.start()
// Fetch Product Info

let request = SKProductsRequest(productIdentifiers: identifierSet)
request.delegate = self
request.start()
// Fetch Product Info

let request = SKProductsRequest(productIdentifiers: identifierSet)
request.delegate = self
request.start()
// Fetch Product Info

let request = SKProductsRequest(productIdentifiers: identifierSet)
request.delegate = self
request.start()
// Fetch Product Info

func productsRequest(_ request: SKProductsRequest, didReceive response: SKProductsResponse) {
    for product in response.products {
        // Localized title and description
        product.localizedDescription
        product.localizedTitle
        product.localizedDescription
        // Price and locale
        product.price
        product.priceLocale
        // Content size and version (hosted)
        product.downloadContentLengths
        product.downloadContentSizeVersion
    }
}

func productsRequest(_ request: SKProductsRequest, didReceive response: SKProductsResponse) {
    for product in response.products {
        // Localized title and description
        product.localizedDescription
        product.localizedTitle
        product.localizedDescription
        // Price and locale
        product.price
        product.priceLocale
        // Content size and version (hosted)
        product.downloadContentSize
        product.downloadContentVersion
    }
}
// Fetch Product Info

func productsRequest(_ request: SKProductsRequest, didReceive response: SKProductsResponse) {
    for product in response.products {
        // Localized title and description
        product.localizedTitle
        product.localizedDescription
        // Price and locale
        product.price
        product.priceLocale
        // Content size and version (hosted)
        product.downloadContentLengths
        product.downloadContentVersion
    }
}

// Fetch Product Info

func productsRequest(_ request: SKProductsRequest, didReceive response: SKProductsResponse) {
    for product in response.products {
        // Localized title and description
        product.localizedTitle
        product.localizedDescription
        // Price and locale
        product.price
        product.priceLocale
        // Content size and version (hosted)
        product.downloadContentLengths
        product.downloadContentVersion
    }
}
// Fetch Product Info

func productsRequest(_ request: SKProductsRequest, didReceive response: SKProductsResponse) {
    for product in response.products {
        // Localized title and description
        product.localizedTitle
        product.localizedDescription
        // Price and locale
        product.price
        product.priceLocale
        // Content size and version (hosted)
        product.downloadContentLengths
        product.downloadContentVersion
    }
}

In-App Purchase Process

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 UI

Up to the application
In-App Purchase UI

Up to the application can have a large effect on sales.
In-App Purchase UI

Up to the application can have a large effect on sales.

In-App Purchase UI
Formatting the product price

```swift
let formatter = NumberFormatter()
formatter.numberStyle = .currency
formatter.locale = product.priceLocale // Not the device locale!
let formattedString = formatter.string(from: product.price)
```
let formatter = NumberFormatter()
formatter.numberStyle = .currency
formatter.locale = product.priceLocale // Not the device locale!
let formattedString = formatter.string(from: product.price)
In-App Purchase UI
Formatting the product price

```swift
let formatter = NumberFormatter()
formatter.numberStyle = .currency
formatter.locale = product.priceLocale // Not the device locale!
let formattedString = formatter.string(from: product.price)
```
In-App Purchase UI
Formatting the product price

```swift
let formatter = NumberFormatter()
formatter.numberStyle = .currency
formatter.locale = product.priceLocale // Not the device locale!
let formattedString = formatter.string(from: product.price)
```
In-App Purchase UI
Formatting the product price

```swift
let formatter = NumberFormatter()
formatter.numberStyle = .currency
formatter.locale = product.priceLocale // Not the device locale!
let formattedString = formatter.string(from: product.price)
```
In-App Purchase UI
Formatting the product price

```swift
let formatter = NumberFormatter()
formatter.numberStyle = .currency
formatter.locale = product.priceLocale // Not the device locale!
let formattedString = formatter.string(from: product.price)
```
In-App Purchase UI
Formatting the product price

```swift
let formatter = NumberFormatter()
formatter.numberStyle = .currency
formatter.locale = product.priceLocale // Not the device locale!
let formattedString = formatter.string(from: product.price)
```

Do not perform currency conversion
In-App Purchase Process

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
// Requesting a Payment

let payment = SKPayment(product: product)
SKPaymentQueue.default().add(payment)
// Requesting a Payment

let payment = SKPayment(product: product)

SKPaymentQueue.default().add(payment)
// Requesting a Payment

let payment = SKPayment(product: product)

SKPaymentQueue.default().add(payment)
// Requesting a Payment

let payment = SKPayment(product: product)
SKPaymentQueue.default().add(payment)
App Store

REVERB
PACEMAKER
IN-APP PURCHASE

RATING  4+

ACCOUNT  J.APPLESEED@ICLOUD.COM

PAY APP STORE  $1.99

Buy with Touch ID
Detecting Irregular Activity
Suspicious activity during payment process
Detecting Irregular Activity
Suspicious activity during payment process

App Store

johnnyappleseed@icloud.com
d-higgins@icloud.com
hank-zakroff@icloud.com

Server

Phone

Phone

Laptop
Detecting Irregular Activity
Suspicious activity during payment process

johnnyappleseed@icloud.com
d-higgins@icloud.com
hank-zakroff@icloud.com
neverfoldsJim35
Detecting Irregular Activity
Provide an account identifier
Detecting Irregular Activity
Provide an account identifier

For applications with their own account management
Detecting Irregular Activity
Provide an account identifier

For applications with their own account management

Provide an opaque identifier for your user’s account
Detecting Irregular Activity
Provide an account identifier

For applications with their own account management
Provide an opaque identifier for your user’s account
• Don’t send us the user’s Apple ID
Detecting Irregular Activity
Provide an account identifier

For applications with their own account management

Provide an opaque identifier for your user’s account
• Don’t send us the user’s Apple ID
• Don’t provide the actual account name
Detecting Irregular Activity
Provide an account identifier

For applications with their own account management

Provide an opaque identifier for your user’s account
• Don’t send us the user’s Apple ID
• Don’t provide the actual account name
• Don’t provide the password
Detecting Irregular Activity
Provide an account identifier

For applications with their own account management
Provide an opaque identifier for your user’s account
• Don’t send us the user’s Apple ID
• Don’t provide the actual account name
• Don’t provide the password
• We suggest using a hash of the account name
Detecting Irregular Activity
Provide an account identifier

For applications with their own account management

Provide an opaque identifier for your user’s account
• Don’t send us the user’s Apple ID
• Don’t provide the actual account name
• Don’t provide the password
• We suggest using a hash of the account name

```swift
let payment = SKPayment(product: product)
payment.applicationUsername = hash(yourCustomerAccountName)
SKPaymentQueue.default().add(payment)
```
Detecting Irregular Activity
Provide an account identifier

For applications with their own account management

Provide an **opaque identifier for your user’s account**

- Don’t send us the user’s Apple ID
- Don’t provide the actual account name
- Don’t provide the password
- We suggest using a hash of the account name

```swift
let payment = SKPayment(product: product)
payment.applicationUsername = hash(yourCustomerAccountName)
SKPaymentQueue.default().add(payment)
```
In-App Purchase Process

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
// 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
    }
}
// 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
    }
}
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
    }
}
// Handle SKPaymentQueueObserver Events

// MARK: - SKPaymentTransactionObserver

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
        case .purchased:
            // Validate the purchase
        }
    }
}
// Handle SKPaymentQueueObserver Events

// MARK: - SKPaymentTransactionObserver

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
        case .purchased:
            // Validate the purchase
        }
    }
}
// Handle SKPaymentQueueObserver Events

// MARK: - SKPaymentTransactionObserver

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
        case .purchased:
            // Validate the purchase
        }
    }
}
func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
        case .purchased:
            // Validate the purchase
        }
    }
}
// Handle SKPaymentQueueObserver Events

// MARK: - SKPaymentTransactionObserver

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

        }
    }
}
// Handle SKPaymentQueueObserver Events

// MARK: - SKPaymentTransactionObserver

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
        case .purchased:
            // Validate the purchase
        case .deferred:
            // Allow the user to continue to use the app
            // It may be some time before the transaction is updated
            // Do not get stuck in a modal "Purchasing..." state!
        }
    }
}
// Handle SKPaymentQueueObserver Events

// MARK: - SKPaymentTransactionObserver

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
        case .purchased:
            // Validate the purchase
        case .deferred:
            // Allow the user to continue to use the app
            // It may be some time before the transaction is updated
            // Do not get stuck in a modal "Purchasing..." state!
    }
    }
}
}
// Handle SKPaymentQueueObserver Events

// MARK: - SKPaymentTransactionObserver

func paymentQueue(_ queue: SKPaymentQueue, updatedTransactions transactions: [SKPaymentTransaction]) {
    for transaction in transactions {
        switch transaction.transactionState {
        case .purchased:
            // Validate the purchase
        case .deferred:
            // Allow the user to continue to use the app
            // It may be some time before the transaction is updated
            // Do not get stuck in a modal "Purchasing..." state!
        }
    }
}
Testing Deferred Transactions
Testing Deferred Transactions

Create a mutable payment
Testing Deferred Transactions

Create a mutable payment

Set the `simulatesAskToBuyInSandbox` flag
Testing Deferred Transactions

Create a mutable payment

Set the `simulatesAskToBuyInSandbox` flag

```swift
let payment = SKMutablePayment(product: product)
payment.simulatesAskToBuyInSandbox = true
SKPaymentQueue.default().add(payment)
```
Handling Errors
Handling Errors

Not all errors are equal
Handling Errors

Not all errors are equal

Check the error code
Handling Errors

Not all errors are equal

Check the error code

• Don’t show an error alert unless necessary
Handling Errors

Not all errors are equal

Check the error code

• Don’t show an error alert unless necessary
• User canceling a payment will result in an error
Handling Errors

Not all errors are equal

Check the error code

• Don’t show an error alert unless necessary
• User canceling a payment will result in an error

Let StoreKit handle the transaction flow as much as possible
Handling Errors

Not all errors are equal

Check the error code
• Don’t show an error alert unless necessary
• User canceling a payment will result in an error

Let StoreKit handle the transaction flow as much as possible
• Including asking for confirmation for purchase
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
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
• Restrict access to downloadable content
• Used often for subscriptions
Receipt Validation

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

Server-to-server validation
• Restrict access to downloadable content
• Used often for subscriptions
In-App Purchase Process

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
Unlock Content

Unlock functionality in your app
Unlock Content

Unlock functionality in your app

Download additional content
Downloading Content
Downloading Content

Apple-hosted content
Downloading Content

Apple-hosted content
• On-demand resources
Downloading Content

Apple-hosted content
• On-demand resources
• Hosted in-app purchase content
Downloading Content

Apple-hosted content
• On-demand resources
• Hosted in-app purchase content

Self-hosted content
Downloading Content

Apple-hosted content
• On-demand resources
• Hosted in-app purchase content

Self-hosted content
• Use background downloads with NSURLSession
In-App Purchase Process

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
Finish the Transaction
Finish the Transaction

Finish all transactions once content is unlocked

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

SKPaymentQueue.default().finishTransaction(transaction)
App Review

You must have a Restore button
App Review

You must have a Restore button

Restore and Purchase must be separate buttons
App Review

You must have a Restore button

Restore and Purchase must be separate buttons

Not just as a “backup” tool
App Review

You must have a Restore button

Restore and Purchase must be separate buttons

Not just as a “backup” tool

• Users with multiple devices
 Restore Completed Transactions

Only restores transactions for
• Non-consumables
• Auto-renewable subscriptions

For consumables and non-renewing subscriptions
• You must persist the state!
Restore Completed Transactions
Restore Completed Transactions

SKPaymentQueue.default().restoreCompletedTransactions()
Restore Completed Transactions

SKPaymentQueue.default().restoreCompletedTransactions()

Observe the queue

// Additional callbacks in SKPaymentTransactionObserver
func paymentQueueRestoreCompletedTransactionsFinished(_ queue: SKPaymentQueue) {}
func paymentQueue(_ queue: SKPaymentQueue,
    restoreCompletedTransactionsFailedWithError error: NSError) {}

Inspect the receipt and unlock content and features accordingly
Implementing In-App Purchases
Implementing In-App Purchases

Always observe the Payment Queue
Implementing In-App Purchases

Always observe the Payment Queue

Fetch localized product information from the App Store
Implementing In-App Purchases

Always observe the Payment Queue

Fetch localized product information from the App Store

Display pricing using the product’s price locale
Implementing In-App Purchases

Always observe the Payment Queue

Fetch localized product information from the App Store

Display pricing using the product’s price locale

Use the receipt to validate your purchases
Implementing In-App Purchases

Always observe the Payment Queue

Fetch localized product information from the App Store

Display pricing using the product’s price locale

Use the receipt to validate your purchases

Make the content available
Implementing In-App Purchases

Always observe the Payment Queue

Fetch localized product information from the App Store

Display pricing using the product’s price locale

Use the receipt to validate your purchases

Make the content available

Finish the transaction
Implementing In-App Purchases

Always observe the Payment Queue
Fetch localized product information from the App Store
Display pricing using the product’s price locale
Use the receipt to validate your purchases
Make the content available
Finish the transaction
Allow the user to restore completed transactions
Promoting In-App Purchases

Ross LeBeau, App Store Engineer
Promoting In-App Purchases
Promoting In-App Purchases

Discoverable
Promoting In-App Purchases

Discoverable
• App page
Promoting In-App Purchases

Discoverable
• App page
• Editorial features
Promoting In-App Purchases

Discoverable
• App page
• Editorial features
• Search results
Promoting In-App Purchases

Discoverable
• App page
• Editorial features
• Search results

Start purchase on the App Store
Implementation Overview
Implementation Overview

Required

• Set up in iTunes Connect
• Handle info from App Store
Implementation Overview

Required
- Set up in iTunes Connect
- Handle info from App Store

Optional
- Order and visibility
Implementation Overview

Required
• Set up in iTunes Connect
• Handle info from App Store

Optional
• Order and visibility
Today

IN-APP PURCHASE

Reverb
for Pacemaker
Size matters—control the size of the room

$1.99
Reverb for Pacemaker
Size matters—control the size of the room
$1.99
// Continuing a Transaction from the App Store

// MARK: - SKPaymentTransactionObserver

func paymentQueue(_ queue: SKPaymentQueue, shouldAddStorePayment payment: SKPayment, forProduct product: SKProduct) -> Bool {
    return true
}

// Continuing a Transaction from the App Store

// MARK: - SKPaymentTransactionObserver

func paymentQueue(_ queue: SKPaymentQueue, shouldAddStorePayment payment: SKPayment, forProduct product: SKProduct) -> Bool {
    return true
}
}
// Deferring or Stopping a Transaction

// MARK: - SKPaymentTransactionObserver

func paymentQueue(_ queue: SKPaymentQueue, shouldAddStorePayment payment: SKPayment, forProduct product: SKProduct) -> Bool {
    // Hold on to the payment
    return false
}

SKPaymentQueue.default().add(savedPayment)
// Deferring or Stopping a Transaction

// MARK: - SKPaymentTransactionObserver

func paymentQueue(_ queue: SKPaymentQueue, shouldAddStorePayment payment: SKPayment, forProduct product: SKProduct) -> Bool {
    // Hold on to the payment
    return false
}

SKPaymentQueue.default().add(savedPayment)
// Deferring or Stopping a Transaction

// MARK: - SKPaymentTransactionObserver

func paymentQueue(_ queue: SKPaymentQueue, shouldAddStorePayment payment: SKPayment, forProduct product: SKProduct) -> Bool {
    // Hold on to the payment
    return false
}

SKPaymentQueue.default().add(savedPayment)
# Testing Purchases

<table>
<thead>
<tr>
<th>Protocol</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol</td>
<td><code>itms-services://</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;action&quot;</td>
<td>&quot;purchaseIntent&quot;</td>
</tr>
<tr>
<td>&quot;bundleId&quot;</td>
<td><code>com.example.app</code></td>
</tr>
<tr>
<td>&quot;productIdentifier&quot;</td>
<td><code>product_name</code></td>
</tr>
</tbody>
</table>
## Testing Purchases

<table>
<thead>
<tr>
<th>Protocol</th>
<th>itms-services://</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>&quot;action&quot;</td>
<td>&quot;purchaseIntent&quot;</td>
</tr>
<tr>
<td>&quot;bundleId&quot;</td>
<td>com.example.app</td>
</tr>
<tr>
<td>&quot;productName&quot;</td>
<td>product_name</td>
</tr>
</tbody>
</table>

itms-services://?action=purchaseIntent&bundleId=com.example.app&productName=product_name
Order and Visibility

Defaults in iTunes Connect
Order and Visibility

Defaults in iTunes Connect

Override on device
Order and Visibility

Defaults in iTunes Connect

Override on device

Not synced
Visibility

Pro Subscription
Detailed topography and satellite imagery.
$35.99

Fishing Hot Spots
Find hidden streams and lakes for hundreds of top fishing spots.
$1.99

Hidden Beaches
Get to the most isolated beaches away from all the tourists.
$1.99
Visibility

Fishing Hot Spots
Find hidden streams and lakes for hundreds of top fishing spots.
$1.99

Hidden Beaches
Get to the most isolated beaches away from all the tourists.
$1.99
// Updating Visibility Override of a Promoted In-App Purchase

// Fetch Product Info for Pro Subscription

let storePromotionController = SKProductStorePromotionController.default()
storePromotionController.update(storePromotionVisibility: .hide, forProduct: proSubscription,
    completionHandler: { (error: Error?) in
        // Complete
    })
let storePromotionController = SKProductStorePromotionController.default() 
storePromotionController.update(storePromotionVisibility: .hide, forProduct: proSubscription, completionHandler: { (error: Error?) in 
    // Complete 
})
let storePromotionController = SKProductStorePromotionController.default()
storePromotionController.update(storePromotionVisibility: .hide, forProduct: proSubscription, completionHandler: { (error: Error?) in
    // Complete
})
// Reading Visibility Override of a Promoted In-App Purchase

// Fetch Product Info for Hidden Beaches pack

let storePromotionController = SKProductStorePromotionController.default()
storePromotionController.fetchStorePromotionVisibility(forProduct: hiddenBeaches,
    completionHandler: { (visibility: SKProductStorePromotionVisibility, error: Error?) in
        // visibility == .default
    })
// Reading Visibility Override of a Promoted In-App Purchase

// Fetch Product Info for Hidden Beaches pack

let storePromotionController = SKProductStorePromotionController.default()

storePromotionController.fetchStorePromotionVisibility(forProduct: hiddenBeaches,
completionHandler: { (visibility: SKProductStorePromotionVisibility, error: Error?) in
    // visibility == .default
})
Order

Pro Subscription
Detailed topography and satellite imagery.
$35.99

Fishing Hot Spots
Find hidden streams and lakes for hundreds of top fishing spots.
$1.99

Hidden Beaches
Get to the most isolated beaches away from all the tourists.
$1.99
Order

- **Hidden Beaches**: Get to the most isolated beaches away from all the tourists. $1.99
- **Pro Subscription**: Detailed topography and satellite imagery. $35.99
- **Fishing Hot Spots**: Find hidden streams and lakes for hundreds of top fishing spots. $1.99
// Updating Order Override of Promoted In-App Purchases

// Fetch Product Info for Pro Subscription, Fishing Hot Spots, and Hidden Beaches

let storePromotionController = SKProductStorePromotionController.default()
let newProductsOrder = [hiddenBeaches, proSubscription, fishingHotSpots]
storePromotionController.updateStorePromotionOrder(newProductsOrder,
    completionHandler: { (error: Error?) in
        // Complete
    })}
// Updating Order Override of Promoted In-App Purchases

// Fetch Product Info for Pro Subscription, Fishing Hot Spots, and Hidden Beaches

let storePromotionController = SKProductStorePromotionController.default()
let newProductsOrder = [hiddenBeaches, proSubscription, fishingHotSpots]
storePromotionController.updateStorePromotionOrder(newProductsOrder,
    completionHandler: { (error: Error?) in
        // Complete
        }
    )
// Updating Order Override of Promoted In-App Purchases

// Fetch Product Info for Pro Subscription, Fishing Hot Spots, and Hidden Beaches

let storePromotionController = SKProductStorePromotionController.default()
let newProductsOrder = [hiddenBeaches, proSubscription, fishingHotSpots]

storePromotionController.updateStorePromotionOrder(newProductsOrder,
    completionHandler: { (error: Error?) in
        // Complete
    })}
let storePromotionController = SKProductStorePromotionController.default()

storePromotionController.fetchStorePromotionOrder(completionHandler: {
    (products: [SKProduct], error: Error?) in
        // products == [hiddenBeaches, proSubscription, fishingHotSpots]
})
// Reading Order Override of Promoted In-App Purchases

let storePromotionController = SKProductStorePromotionController.default()
storePromotionController.fetchStorePromotionOrder(completionHandler: {
    (products: [SKProduct], error: Error?) in
    // products == [hiddenBeaches, proSubscription, fishingHotSpots]
})
Promoting In-App Purchases
Promoting In-App Purchases

Discoverable in App Store
Promoting In-App Purchases

Discoverable in App Store

Set up in iTunes Connect
Promoting In-App Purchases

Discoverable in App Store

Set up in iTunes Connect

Start purchase in App Store
Promoting In-App Purchases

Discoverable in App Store

Set up in iTunes Connect

Start purchase in App Store

Handle in app via SKPaymentTransactionObserver
Promoting In-App Purchases

Discoverable in App Store
Set up in iTunes Connect
Start purchase in App Store
Handle in app via SKPaymentTransactionObserver
Optional—order and visibility
Ratings, Reviews, and Responses

Pete Hare, App Store Engineer
Alto's Adventure
A serene snowboarding odyssey

$0.99

4.7, 7.8K Ratings
Game
Action
Age Rating

Offers iPad and Apple TV Apps

Above the placid ivory snow lies a sleepy mountain village, brimming with the promise of adventure. Join Alto and his friends as they...
Ratings & Reviews

4.7 out of 5
7,867 Ratings

Truly Amazing Game
1d ago
CuriousFox25

This game is wonderful and highly enjoyable for the entire family! I love how easy it is to pick up and play wherever you are. If you need an infinite runner to play with the kids, this is the one to get. After a long day, the colorful visuals and cheery music always put me in a good mood.

Editors’ Choice

In this gorgeous twist on the infinite runner, you guide an agile snowboarder down a never ending mountain, pull...
Ratings & Reviews

4.7 out of 5
7,867 Ratings

Impossible to Stop Playing

2d ago
PhilSchiller

This game rocks! Ever since I unlocked the wingsuit I haven’t been able to put it down... even during our ET meetings.

Developer Response

1h ago

Just make sure to keep the volume off!

Editors’ Choice

In this gorgeous twist on the infinite runner, you guide an agile snowboarder down a never ending mountain, pulli...
Ratings & Reviews

4.7
out of 5
7,867 Ratings

A Fun Challenge

4d ago
F1racer09

I can’t even count how many hours I’ve spent playing this awesome game. The visuals are stunning and doing backflips is so addictive. But it’s not easy—I had to replay one level multiple times before moving on. I’m getting better each time I play, but I still have a lot to learn.

Editors’ Choice

In this gorgeous twist on the infinite runner, you guide an agile snowboarder down a never ending mountain, pulli...
Ratings, Reviews, and Responses

What’s new

Reset your rating
Ratings, Reviews, and Responses

What’s new

Reset your rating

Respond to reviews
Ratings, Reviews, and Responses

What’s new

Reset your rating

Respond to reviews

Ask for ratings and reviews via SKStoreReviewController
Ratings, Reviews, and Responses

What’s new

Reset your rating

Respond to reviews

Ask for ratings and reviews via SKStoreReviewController

Deep link to write review in the App Store
Ratings, Reviews, and Responses
What’s new

Reset your rating

Respond to reviews

Ask for ratings and reviews via SKStoreReviewController

Deep link to write review in the App Store

Helpfulness and Report a Concern on iOS
This game is wonderful and highly enjoyable for the entire family! I love how easy it is to pick up and play wherever you are. If you need an infinite runner to play with the kids, this is the one to get. After a long day, the colorful visuals and cheery music always puts me in a good mood.
This game is wonderful and highly enjoyable for the entire family! I love how easy it is to pick up and play wherever you are. If you need an infinite runner to play with the kids, this is the one to get. After a long day, the colorful visuals and cheery music always puts me in a good mood.
Ratings, Reviews, and Responses

Responding to reviews

I loved this game

⭐⭐⭐⭐⭐

Jan 30
Comcup

I have never written a review before, but this game is so good. I just want more levels. I played through this game at

Developer Response

Feb 14
Thanks for making us your first review! Keep checking for updates, there’s more!

Awesome game

⭐⭐⭐⭐⭐

Feb 2
De Maximus

If you like unusual challenge of ripping a paper up to save the cute little guys, this game is fun to be stumped, save the

Developer Response

Mar 16
Glad you had fun getting stumped. We’re big fans of unusual challenges, too!
You’ve received a developer response to your review of TripGuides

Today at 9:41 AM

Dear Johnny,

App Co. responded to your review of

TripGuides
App Co.

“Glad you’re enjoying the game though I’m sorry you’re having problems on that level, we’ll be issuing an update soon to fix it. In the meantime we suggest you go level 3 manually via Load Game in the Main Menu. Sorry again!”

Do you want to update your review?

You can also email the developer.
Dear Johnny,

App Co. responded to your review of TripGuides

“Glad you’re enjoying the game though I’m sorry you’re having problems on that level, we’ll be issuing an update soon to fix it. In the meantime we suggest you go level 3 manually via Load Game in the Main Menu. Sorry again.”

Do you want to update your review?

You can also email the developer.
This game has improved a LOT. There were some problems with the earlier version, but since updating I've had no issues. Thanks for listening!
Average increase of 1.5 stars per review
Ratings, Reviews, and Responses
Responding to reviews

More information
Ratings, Reviews, and Responses
Responding to reviews

More information
Ratings, Reviews, and Responses
Receiving reviews
Ratings, Reviews, and Responses
Receiving reviews

Prompt for review with SKStoreReviewController
Ratings, Reviews, and Responses
Receiving reviews

Prompt for review with SKStoreReviewController

Deep link to review in App Store
Ratings, Reviews, and Responses
Asking for ratings and reviews with SKStoreReviewController
Ratings, Reviews, and Responses
Asking for ratings and reviews with SKStoreReviewController

Introduced in iOS 10.3
Ratings, Reviews, and Responses
Asking for ratings and reviews with SKStoreReviewController

Introduced in iOS 10.3
Quick way to request a rating/review
Ratings, Reviews, and Responses
Asking for ratings and reviews with SKStoreReviewController

Introduced in iOS 10.3
Quick way to request a rating/review
Will be required for all modal rating/review prompts
Ratings, Reviews, and Responses
Asking for ratings and reviews with SKStoreReviewController

Introduced in iOS 10.3
Quick way to request a rating/review
Will be required for all modal rating/review prompts
Restrictions in place
Ratings, Reviews, and Responses
Asking for ratings and reviews with SKStoreReviewController

Introduced in iOS 10.3

Quick way to request a rating/review

Will be required for all modal rating/review prompts

Restrictions in place
  • Limited requests per device
Ratings, Reviews, and Responses
Asking for ratings and reviews with SKStoreReviewController

Introduced in iOS 10.3

Quick way to request a rating/review

Will be required for all modal rating/review prompts

Restrictions in place
  • Limited requests per device
  • Can be disabled by user in Settings
Ratings, Reviews, and Responses
Asking for ratings and reviews with SKStoreReviewController
Ratings, Reviews, and Responses
Asking for ratings and reviews with SKStoreReviewController

☑ Present as a prompt after a user action
Ratings, Reviews, and Responses
Asking for ratings and reviews with SKStoreReviewController

Present as a prompt after a user action

Do not present from a button
May not present any UI due to restrictions
// Asking for Ratings and Reviews with SKStoreReviewController

if shouldPromptUser() {
    SKStoreReviewController.requestReview()
}

func shouldPromptUser() -> Bool {
    // Local business rules
}
// Asking for Ratings and Reviews with SKStoreReviewController

if shouldPromptUser() {
    SKStoreReviewController.requestReview()
}

func shouldPromptUser() -> Bool {
    // Local business rules
}
// Asking for Ratings and Reviews with SKStoreReviewController

if shouldPromptUser() {
    SKStoreReviewController.requestReview()
}

func shouldPromptUser() -> Bool {
    // Local business rules
}
// Asking for Ratings and Reviews with SKStoreReviewController

if shouldPromptUser() {
    SKStoreReviewController.requestReview()
}

func shouldPromptUser() -> Bool {
    // Local business rules
}
Ratings, Reviews, and Responses
Deep link to write a review in App Store

Great improvements!
This game has improved a LOT. There were some problems with the earlier version, but since updating I've had no issues. Thanks for listening!
Ratings, Reviews, and Responses
Deep link to write a review in App Store

Introduced in iOS 10.3
Ratings, Reviews, and Responses
Deep link to write a review in App Store

Introduced in iOS 10.3
Link to open your app in the App Store
• Presents compose review from app page
Ratings, Reviews, and Responses
Deep link to write a review in App Store

Introduced in iOS 10.3

Link to open your app in the App Store
  • Presents compose review from app page

User initiated actions
  • Button in settings
Ratings, Reviews, and Responses
Deep link to write a review in App Store
Ratings, Reviews, and Responses
Deep link to write a review in App Store

Use from an embedded button in your app
Such as a settings screen
Ratings, Reviews, and Responses
Deep link to write a review in App Store

Use from an embedded button in your app
Such as a settings screen

Do not use from an alert
Use SKStoreReviewController instead
Deep Link to Write Review

URL is formed using regular product URL with an anchor tag

For creating product URLs visit
https://linkmaker.itunes.apple.com/
Ratings, Reviews, and Responses
Asking users for reviews

More information
Summary
Summary

How to implement in-app purchases
Summary

How to implement in-app purchases
Promote in-app purchases in the App Store
Summary

How to implement in-app purchases
Promote in-app purchases in the App Store
New App Store design
Summary

How to implement in-app purchases
Promote in-app purchases in the App Store
New App Store design
New opportunities to improve your ratings and reviews
More Information

https://developer.apple.com/wwdc17/303
<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>What’s New in iTunes Connect</td>
<td></td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>Introducing the New App Store</td>
<td></td>
<td>WWDC 2017</td>
</tr>
<tr>
<td>Advanced StoreKit</td>
<td>Grand Ballroom A</td>
<td>Thursday 1:50PM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lab Name</th>
<th>Lab Location</th>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>App Store and iTunes Connect Lab</td>
<td>Technology Lab H</td>
<td>Thu</td>
<td>12:00PM-1:50PM</td>
</tr>
<tr>
<td>StoreKit Lab</td>
<td>Technology Lab E</td>
<td>Thu</td>
<td>3:10PM-6:00PM</td>
</tr>
<tr>
<td>StoreKit Lab</td>
<td>Technology Lab E</td>
<td>Fri</td>
<td>1:50PM-4:00PM</td>
</tr>
</tbody>
</table>