StoreKit 2 for iOS

Control StoreKit 2 features in your Unity app on **iOS** platform using native Swift API.

This plugin is based on official Apple documentation for **StoreKit 2**.

All logic and naming conventions are as close as possible to the behavior of native Swift APIs.

All asynchronous workflows are preserved using `Task<T>` in C#, reflecting Swift's `async` / `await`.

> ⚠️ **Platform Supports: ✅ iOS 15.0+ ✅ iPadOS 15.0+**

> ⚠️ In-App Purchase capability is enabled automatically upon build.

🛠️ How it Works

StoreKit 2 for iOS bridges native iOS StoreKit2 APIs written in Swift with Unity using P/Invoke and a C# wrapper.

> 🗣️ 🧠 All data is passed between Swift and Unity in **JSON** format and deserialized into strongly typed C# models.

Architecture Overview

1. Unity calls methods from the static `StoreKit2` class.
2. Under the hood, native iOS methods are invoked using `[DllImport("__Internal")]`.
3. Swift handles native StoreKit2 logic (subscriptions, purchases, refunds, etc.).
4. Results are returned as serialized JSON strings and parsed into C# objects (e.g. `Transaction`, `Product`, `SubscriptionStatus`, `Offer`).

API
StoreKit2

Main static class for interaction with StoreKit 2 system.

Member	Description
static event **OnTransactionUpdated**	Raised when a new StoreKit transaction is received or updated. Returns a **Transaction** object.
static event **OnTransactionUpdateError**	Raised when an error occurs while listening for transactions. Returns an error message.
static void **SubscribeTransactionsUpdate**()	Starts listening to StoreKit transaction updates.

static void **UnSubscribeTransactionsUpdate**()	Stops listening to transaction updates.

Product Fetching

Member	Description
static Task<List<Product>> **FetchProductsAsync**(List<string> productIds)	Fetches metadata for the specified product IDs.
static void **FetchProducts**(List<string> ids, Action<List<Product>> success, Action<string> error)	Same as above, but with callback-based interface.

Purchasing

Member	Description
static Task<Transaction> **PurchaseProductAsync**(string productId)	Begins a purchase flow for a product.
static void **PurchaseProduct**(string productId, Action<Transaction> success, Action<string> error)	Same as above, with callbacks.
static bool **CanMakePayments**()	static bool **CanMakePayments**()Returns `true` if device is allowed to purchase.

Transaction handling

Member	Description
static Task<List<Transaction>> **GetAllTransactionsAsync**()	Returns all verified transactions.
static void **GetAllTransactions**(Action<List<Transaction>> success, Action<string> error)	Same as above, with callbacks.
static Task<List<Transaction>> **GetActiveTransactionsAsync**()	Returns current active entitlements.
static Task<List<Transaction>> **GetUnfinishedTransactionsAsync**()	Returns unfinished transactions.
static Task<Transaction> **GetLatestTransactionAsync**(string productId)	Gets latest transaction for a given product.
static void **GetLatestTransaction**(string productId, Action<Transaction> success, Action<string> error)	Same as above, with callbacks.
static Task<bool> **FinishTransactionAsync**(ulong transactionId)	Marks a transaction as finished.
static void **FinishTransaction**(ulong transactionId, Action<bool> complete)	Same as above, with callback.

Subscriptions handling

Member	Description
static Task<SubscriptionStatus> **GetSubscriptionStatusAsync**(Transaction transaction)	Returns subscription status for a transaction.

static void **GetSubscriptionStatus**(Transaction transaction, Action<SubscriptionStatus> success, Action<string> error)	Same as above, with callbacks.
static Task<bool> **IsEligibleForIntroOfferAsync**(Product product)	Checks if user is eligible for intro offer.
static void **IsEligibleForIntroOffer**(Product product, Action<bool> complete)	Same as above, with callback.

Sync

Member	Description
static Task<bool> **SyncTransactionsAsync**()	Syncs transactions with App Store.
static void **SyncTransactions**(Action<bool> complete)	Same as above, with callback.

App Account Token

Member	Description
static string **GetOrCreateAppAccountToken**()	Returns or creates a persistent App Account Token — a UUID uniquely identifying the user for App Store interactions. This token is automatically generated on first use and securely stored in iOS Keychain. It is always attached to every purchase made through the plugin to support advanced App Store features such as fraud detection, subscription linking, and backend correlation.
static string **GenerateNewAppAccountToken**()	Generates a new UUID token and overwrites the existing one in Keychain. Use only if you intentionally want to rotate the token.

appAccountToken is a value you assign to correlate users in your backend.

Learn more in Apple docs.

Refunds

Member	Description
static Task<RefundResultType> **BeginRefundRequestAsync**(ulong transactionId)	Opens refund request sheet for a transaction.
static void **BeginRefundRequest**(ulong transactionId, Action<RefundResultType> complete)	Same as above, but uses a callback.

Utilities

Member	Description
static bool **OpenSubscriptionManagement**()	Opens native iOS subscription settings.
static void **PresentCodeRedemptionSheet**()	Opens native sheet for entering promo codes.

Legacy from Store Kit 1

Member	Description

static Task<string> **RefreshAppStoreReceiptAsync**()	Refreshes StoreKit 1 receipt.
static void **RefreshAppStoreReceipt**(Action<string> complete)	Same as above, with callback.
static string **GetAppStoreReceipt**()	Returns current StoreKit 1 receipt as Base64.

🚫 Not Implemented (Yet)

Feature	Status
Promotional offers (signed offers with key + signature)	❌ Not implemented
Offer Codes / Intro Eligibility based on appStoreServer APIs	❌ Not implemented
Native APIStatusNotes `Product.subscription?.status`	❌ Not supported — there is no method to get subscription status directly from a `Product`. Use `Transaction.subscriptionStatus` instead.

API
Transaction

Represents a completed or in-progress purchase made by the user.

Property	Description
`Id`	Unique StoreKit transaction identifier
`OriginalID`	ID of the original transaction (for renewals/restores)
`Storefront`	App Store storefront (e.g., "US")
`OriginalPurchaseDate`	Date of the original purchase
`Environment`	Environment of the transaction (`Sandbox`, `Production`, etc.)
`WebOrderLineItemID`	Group-level transaction ID for subscriptions
`AppBundleID`	Your app's bundle identifier
`ProductID`	Purchased product ID
`ProductType`	Type of the product (e.g., AutoRenewable)
`SubscriptionGroupID`	Subscriptions group ID, if applicable
`PurchaseDate`	Date of this transaction
`ExpirationDate`	Expiration date (if subscription)
`Price`	Recorded price for this transaction
`Currency`	Currency code (e.g., "USD")
`IsUpgraded`	Whether the user upgraded a subscription
`OwnershipType`	Whether the user owns this or got it via Family Sharing
`PurchasedQuantity`	Quantity of product purchased (usually 1)
`Reason`	Reason for the transaction (`Purchase`, `Renewal`, etc.)
`Offer`	Applied `Offer`, if available (iOS 17.2+)
`AppAccountToken`	App-defined user token (UUID)

Apple Reference: `Transaction`

API
Product

Metadata for a purchasable item in the App Store.

Property	Description
`Id`	Product identifier
`Type`	Product type (`Consumable` , `AutoRenewable` , etc.)
`IsFamilyShareable`	Can this be shared via Family Sharing
`DisplayName`	Localized display name
`Description`	Localized description
`Price`	Decimal price (e.g., 4.99)
`DisplayPrice`	Localized price string (e.g., "₴129.00")
`CurrencyCode`	ISO currency code (e.g., "USD")
`CurrencySymbol`	Localized currency symbol
`SubscriptionPeriod`	Subscription billing period (if subscription)
`IntroductoryOffer`	Intro offer (if configured)
`WinBackOffers`	Win-back offers (iOS 18+)
`IsSubscription`	Whether this product is a subscription
`GroupLevel`	Relative priority in subscription group (iOS 16.4+)
`GroupDisplayName`	Localized group name
`SubscriptionGroupID`	ID of the subscription group

Apple Reference: `Product`

API
SubscriptionStatus

Represents the current status and renewal metadata for a subscription.

Property	Description
`State`	Current subscription state: `Subscribed` , `Expired` , `InGracePeriod` , etc.
`RenewalInfo`	Metadata about next renewal (see below)

Apple Reference: **subscriptionStatus**

API
RenewalInfo

Details about the upcoming renewal for an auto-renewable subscription.

Property	Description
`Environment`	Environment where info was signed (`Sandbox` , etc.)

`OriginalTransactionID`	ID of the original purchase
`ExpirationReason`	Reason for subscription end
`WillAutoRenew`	Will renew automatically
`RenewalPrice`	Price for next renewal
`Currency`	Currency of renewal price
`AutoRenewPreference`	Product ID that will renew next
`CurrentProductID`	Currently active product ID
`GracePeriodExpiresDate`	When grace period ends, if active
`IsInBillingRetry`	In retry billing state
`PriceIncreaseStatus`	Has user accepted price increase
`RecentSubscriptionStartDate`	When the current period started
`Offer`	Subscription offer applied at renewal (iOS 18+)
`RenewalDate`	Scheduled renewal/expiration date

Apple Reference: `Product.SubscriptionInfo.RenewalInfo`

API
Offer

Represents a promotional offer or introductory deal.

Property	Description
`Id`	Offer identifier
`Price`	Price of the offer
`PaymentMode`	Mode (free trial, pay up front, etc.)
`Type`	Type of offer: `Intro`, `Promotional`, `WinBack`, etc.
`Period`	Duration of offer (e.g., 7 days)

Apple Reference: `Product.SubscriptionOffer`

API
SubscriptionPeriod

Defines the time unit and value for recurring billing periods.

Property	Description
`Unit`	Unit of time: `Day`, `Week`, `Month`, or `Year`
`Value`	How many of the above units

Example: `Unit = Month`, `Value = 3` means "every 3 months".

Apple Reference: `Product.SubscriptionPeriod`

❗ Error Handling

The plugin uses a combination of **native SKError codes** and custom error mappings to provide consistent and readable error messages across Unity.

Error Code	Source	Description
`SKErrorUnknown`	`SKError.Code.unknown` / `PurchaseError.unknown`	Unknown error occurred
`SKErrorClientInvalid`	`SKError.Code.clientInvalid`	Client is not allowed to issue the request
`SKErrorPaymentCancelled`	`SKError.Code.paymentCancelled`	User canceled the payment request
`SKErrorPaymentInvalid`	`SKError.Code.paymentInvalid`	Purchase identifier was invalid
`SKErrorPaymentNotAllowed`	`SKError.Code.paymentNotAllowed`	Device is not allowed to make payments
`SKErrorStoreProductNotAvailable`	`SKError.Code.storeProductNotAvailable`	Product not available in the current storefront
`SKErrorCloudServicePermissionDenied`	`SKError.Code.cloudServicePermissionDenied`	User has not allowed access to cloud service
`SKErrorCloudServiceNetworkConnectionFailed`	`SKError.Code.cloudServiceNetworkConnectionFailed`	Could not connect to the network
`SKErrorCloudServiceRevoked`	`SKError.Code.cloudServiceRevoked`	Cloud service was revoked
`SKErrorPrivacyAcknowledgementRequired`	`SKError.Code.privacyAcknowledgementRequired`	Privacy acknowledgement is required
`SKErrorUnauthorizedRequestData`	`SKError.Code.unauthorizedRequestData`	App is not allowed to use the request data
`SKErrorInvalidOfferIdentifier`	`SKError.Code.invalidOfferIdentifier`	Promotional offer identifier is invalid
`SKErrorInvalidOfferPrice`	`SKError.Code.invalidOfferPrice`	Price for promotional offer is invalid
`SKErrorInvalidSignature`	`SKError.Code.invalidSignature`	Signature for promotional offer is invalid

`SKErrorMissingOfferParams`	`SKError.Code.missingOfferParams`	Required parameters for promotional offer are missing
`SKErrorIneligibleForOffer`	`SKError.Code.ineligibleForOffer`	User is not eligible for the promotional offer
`SKErrorOverlayCancelled`	`SKError.Code.overlayCancelled`	Overlay (e.g., code redemption sheet) was canceled
`SKErrorOverlayInvalidConfiguration`	`SKError.Code.overlayInvalidConfiguration`	Overlay was not configured properly
`SKErrorOverlayPresentedInBackgroundScene`	`SKError.Code.overlayPresentedInBackgroundScene`	Tried to present overlay while app was in background
`SKErrorOverlayTimeout`	`SKError.Code.overlayTimeout`	Overlay took too long to complete
`SKErrorUnsupportedPlatform`	`SKError.Code.unsupportedPlatform`	Feature is not supported on this device/platform
`SKCustomUnhandledError`	`@unknown default` fallback	An unhandled or unknown error from Apple
`SKErrorProductNotFound`	`PurchaseError.productNotFound`	No product found with the given ID
`SKErrorTransactionNotFound`	`PurchaseError.transactionNotFound`	No transaction found for the given ID
`SKErrorPurchasePending`	`PurchaseError.purchasePending`	Purchase is still pending (not finalized)
`SKErrorPurchaseUnverified`	`PurchaseError.puchaseUnverified`	Transaction failed cryptographic verification

Example

Fetching Products

```
var productIds = new List<string> { "com.myapp.subscription.monthly" };
var products = await StoreKit2.FetchProductsAsync(productIds);
```

Purchasing a Product

```
try
{
```

```
    var transaction = await StoreKit2.PurchaseProductAsync("com.myapp.subscription.monthl
y");
    Debug.Log("Purchase successful: " + transaction.ProductID);
}
catch (Exception e)
{
    Debug.LogError("Purchase failed, code: " + e.Message);
}
```

Listening for Transaction Updates

```
StoreKit2.OnTransactionUpdated += transaction ⇒
{
    Debug.Log("Transaction updated: " + transaction.ProductID);
};

StoreKit2.SubscribeTransactionsUpdate();
```

Checking Subscription Status

```
var transaction = await StoreKit2.GetLatestTransactionAsync("com.myapp.subscription.monthl
y");
var status = await transaction.GetSubscriptionStatusAsync();
Debug.Log("Subscription state: " + status.State;
```

Presenting Refund Sheet

```
var result = await StoreKit2.BeginRefundRequestAsync(transactionId);
Debug.Log("Refund result: " + result);
```