

# Crypto Game Store

---

## Unity In-App Purchasing with Direct Cryptocurrency Payments

### Table of Contents

---

1. [Overview](#)
2. [System Requirements](#)
3. [Getting Started](#)
4. [Installation and Setup](#)
5. [Wallet Configuration](#)
6. [Game Store Server Setup](#)
7. [Server Deployment](#)
8. [Client Integration](#)
9. [Product Configuration](#)
10. [Testing Your Store](#)
11. [Troubleshooting](#)
12. [Glossary](#)

### Overview

---

Crypto Game Store is a Unity asset that integrates with Unity's In-App Purchasing 5.0 system to enable direct cryptocurrency payments from players to developers. This solution eliminates third-party app store and platform fees by allowing customers to pay directly from their crypto wallets to your store's wallet.

### Key Features

- **Direct payments:** Receive payments directly from customers without intermediaries
- **Zero platform fees:** Eliminate traditional app store commissions and fees
- **Blockchain security:** All transactions are secured and verified on the Bitcoin blockchain
- **Unity IAP compatibility:** Seamlessly integrates with existing Unity In-App Purchasing workflows
- **Flexible pricing:** Support for both fiat currency (USD, EUR, JPY, ...) and cryptocurrency pricing
- **Real-time exchange rates:** Automatic conversion between fiat and crypto currencies
- **Self-hosted solution:** Complete control over your payment infrastructure
- **Comprehensive example:** Takes guess work out of getting your store working

### How It Works

The system consists of two main components:

1. **Client-side Unity package:** Integrates with your game and handles the payment interface
2. **Game Store Server:** A self-hosted application that manages wallet interactions

When a player makes a purchase, the Server generates a unique crypto address for the transaction, the client displays payment details to the player, and monitors the blockchain for payment confirmation.

## System Requirements

---

### Unity Environment

- Unity 6000.0 or higher
- Unity In-App Purchasing 5.0
- Unity Services (Authentication and Cloud Save)

### Supported Platforms

- Android clients
- PC Windows clients

### Server Environment

- Windows x64 or Linux x64 or Linux amd64
- Stable internet connection with 24/7 operation

### Wallet Requirements

Your store can use one of two wallet types:

- **Watch-Only Wallet:** Uses an X PUB from existing wallets (hardware or software)
- **Knots Wallet:** Full Bitcoin node implementation for complete control

## Getting Started

---

### Implementation Steps

1. Install the Unity package from the Asset Store
2. Configure Unity Services for your project
3. Choose your wallet type (Watch-Only or Knots)
4. Set up the Game Store Server with your configuration
5. Deploy and run the server on your hosting environment
6. Integrate the client code into your game
7. Define your store products and pricing
8. Test the complete system before production deployment

### Prerequisites

Before beginning, ensure you have:

- A Unity project with Services enabled
- A Bitcoin wallet or plan to set up a Knots node

- A server environment for hosting the Game Store Server
- Basic understanding of Unity's In-App Purchasing system

## Installation and Setup

---

### 1. Install Crypto Game Store Package

The Crypto Game Store package is distributed in two parts:

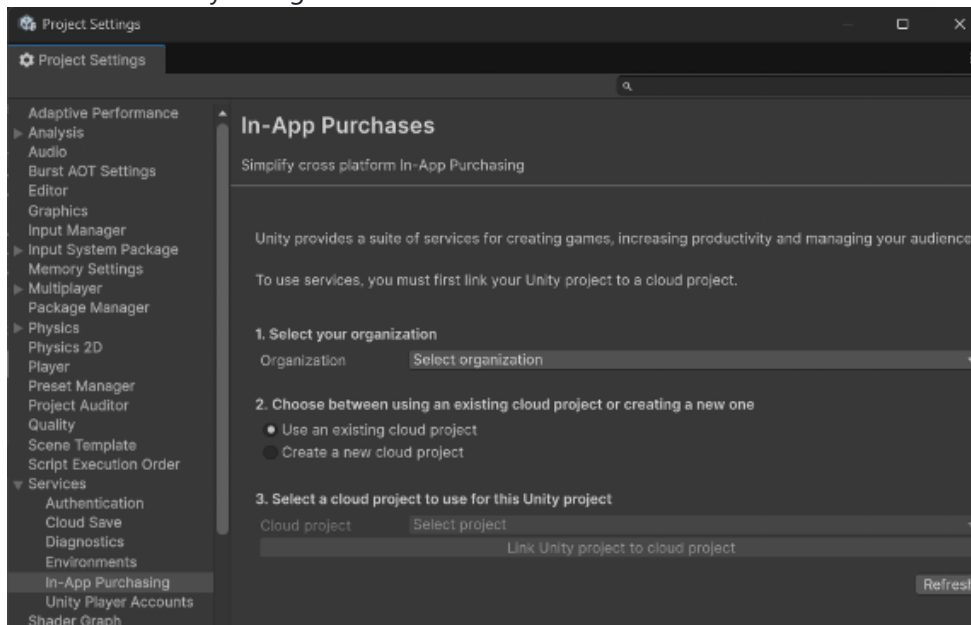
1. **Unity Package:** Available from the Unity Asset Store ( `CryptoGameStore.unitypackage` )
  - Import using Unity's Package Manager
  - Automatically installs required dependencies
2. **Game Store Server:** Downloaded separately due to Unity Asset Store restrictions (Download links below)

### 2. Configure Unity Services

Crypto Game Store requires two Unity Services:

#### In-App Purchasing Setup

1. Navigate to **Services** → **In-App Purchasing** → **Configure...**
2. Follow the setup wizard to enable In-App Purchasing
3. This automatically configures Authentication and Cloud Save services



## Wallet Configuration

---

### Understanding Wallet Types

Crypto Game Store supports two wallet configurations, each with distinct advantages:

## Watch-Only Wallet

A Watch-Only wallet uses an Extended Public Key (XPUB) from an existing wallet to generate deposit only addresses without exposing your wallet's private keys.

### Advantages:

- **Security:** Private keys remain with your primary wallet
- **Simplicity:** Quick setup with existing wallet infrastructure
- **Flexibility:** Compatible with many wallet types and platforms

## Supported Wallets:

Category	Wallets	Platforms	Testnet	XPUB Export Method
Custodial	Coinbase	Web	No	Settings → Export Account → Extended Public Key
	Kraken	Web	No	API → Deposit Addresses → Export XPUB (business accounts)
	Bitstamp	Web	No	Account → API Keys → enable Read-only → XPUB via support
	Gemini	Web	No	Account Details → Export XPUB
Non-Custodial	Knots	Windows	Yes	Window → Console → run command: listdescriptors
	Sparrow	Windows	Yes	Wallet → panel Settings → Keystores → xpub:
	Wasabi	Windows	Yes	3-dot menu → Wallet Info → Extended Account Public Key
	Electrum	Windows / Android	Yes / No	Wallet → Information → Show xpub
	Blockstream	Windows / Android	Yes / Yes	3-dot menu → Settings → Watch-only
	Mycelium	Android	Yes	3-dot menu → Export account keys
	Edge	Android	Yes	3-dot menu → View XPub Address
	Bitpay	Android	Yes	Not supported
	BlueWallet	Android	No	3-dot menu → Show Wallet XPUB
	Blockstream Jade / Jade+	Blockstream / Sparrow / Electrum / Knots / ...	Yes	Options → Wallet → Export Xpub
	Coldcard Mk4 / Q	Wasabi / Sparrow / Electrum / Knots / ...	Yes	Advanced → Export Wallet → Export XPUB (creates xpub.txt)
	Trezor One / Model T	Trezor Suite	Yes	Accounts → three-dot menu → Export XPUB

*Note: This list is not exhaustive. Hardware wallets can also be used with certain non-custodial apps. Consult your wallet's current documentation for specific XPUB export procedures.*

## Knots Wallet

Bitcoin Knots is a full node implementation that provides complete Bitcoin protocol validation and wallet functionality.

### Advantages:

- **Full control:** Complete sovereignty over your Bitcoin operations
- **Network participation:** Contribute to Bitcoin network security
- **Direct validation:** Verify all transactions against the blockchain
- **Hot wallet capability:** Can both receive and spend funds

### Setup Requirements:

1. Download Bitcoin Knots from [bitcoinknots.org](https://bitcoinknots.org)
2. Install using default directories for easier integration
3. Use default configuration settings
4. Enable RPC Server: **Settings** → **Options** → **RPC Server**
5. Create a wallet through the Knots interface

### Installation Resources:

- [Bitcoin Knots Setup Guide](#)
- [Advanced Configuration](#)

*Note: Installing Knots in default locations simplifies Game Store Server integration.*

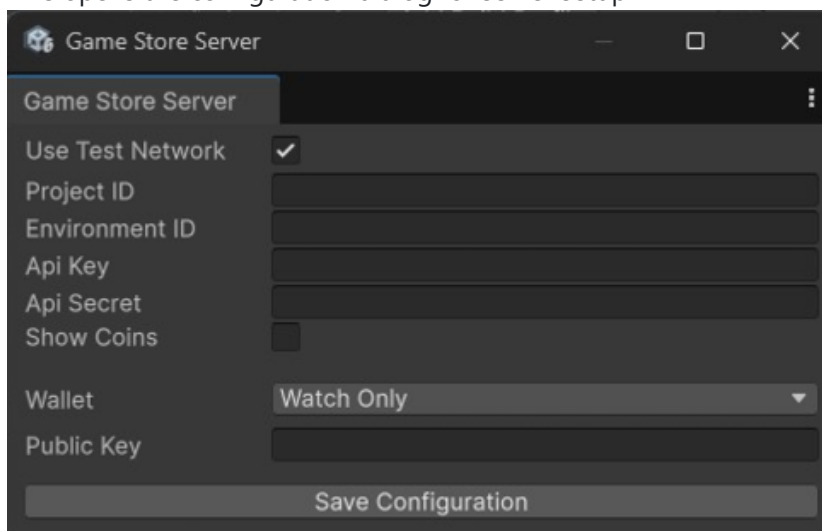
## Game Store Server Setup

---

The Game Store Server configuration is managed through Unity's built-in interface.

### Accessing Configuration

1. Navigate to **Windows** → **Crypto Game Store Server** in Unity
2. This opens the configuration dialog for server setup



# Configuration Parameters

## Network Selection

- **Use Test Network:** Enable for testing with testnet4 (no real Bitcoin required)
- **Use Main Network:** Disable for production with real Bitcoin

*Default: Test Network enabled for safe initial testing*

## Project Identification

### Project ID

- Location: **Project Settings** → **Services** or **Unity Cloud** → **Projects** → **Settings**
- Purpose: Ensures only your game can communicate with your server
- Format: Unique identifier for your Unity project

### Environment ID

- Location: **Project Settings** → **Services** → **Environments** or **Unity Cloud** → **Projects** → **Environments**
- Purpose: Isolates different deployment environments (development, staging, production)
- Default: "production" environment
- Note: Use actual Environment ID from **Unity Cloud**, not display name shown in **Unity Editor**

## Unity Services Authentication

The Game Store Server requires Unity Cloud Save API access through Service Account keys.

### Creating Service Account Keys:

1. Navigate to **Unity Cloud** → **Administration** → **Service Accounts**
2. Create a new service account (if none exist)
3. Select the account and click **+Add Key**
4. Copy the **Key ID** to **Api Key** field
5. Copy the **Secret Key** to **Api Secret** field
6. Save the Secret Key securely (it won't be shown again)

### Setting Permissions:

1. Click **+Manage Project Roles**
2. Select your game's project
3. Under "Live Ops", choose **Cloud Save Editor**
4. Click **Save**

## Optional Settings

### Bitcoin Character Output

- Displays a Bitcoin symbol (฿) when dispensing payment addresses
- Purely informational, does not affect functionality

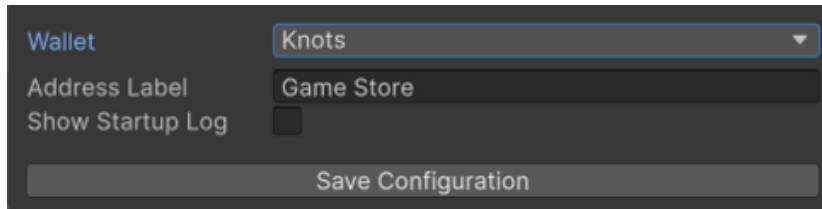
## Wallet-Specific Configuration

### Watch-Only Wallet Setup

#### Public Key (XPUB)

- Enter the XPUB exported from your chosen wallet
- Ensure network type (Main/Test) matches your "Use Test Network" setting
- The system will warn about network mismatches when saving

### Knots Wallet Setup

A screenshot of a configuration window for the Knots wallet. It has a dark theme. At the top, there's a label 'Wallet' and a dropdown menu showing 'Knots'. Below that, there's a label 'Address Label' and a text input field containing 'Game Store'. Underneath is a checkbox labeled 'Show Startup Log' which is currently unchecked. At the bottom of the window is a button labeled 'Save Configuration'.

The Game Store Server auto-configures for Knots installations in default locations.

#### Optional Settings:

##### Address Label

- Custom label added to Knots wallet addresses
- Helps identify Game Store transactions in Knots wallet's transaction log
- Distinguishes from other wallet uses

##### Show Startup Logs

- Enables diagnostic output for Knots startup issues
- Useful for troubleshooting configuration problems

## Saving Configuration

1. Click **Save Configuration** to create a configuration file
2. The file `GameStoreServer.config` is saved to `Assets/GameStore/`
3. Windows Explorer opens and highlights the saved file
4. Move this file to your Game Store Server application directory

## Security Considerations

#### Data Protection:

- **Api Secret** and **Public Key** values are encrypted in the configuration file
- These values are erased from the file after first server startup
- Store these values securely outside the configuration system
- Avoid committing `GameStoreServer.config` to source control



# Server Deployment

---

## Server Application

The Game Store Server is a self-contained executable with no external dependencies except the configuration file.

### Download Links

- Windows x64: [Download win-x64 version](#)
- Linux x64: [Download linux-x64 version](#)
- Linux arm64: [Download linux-arm64 version](#)

Linux: GameStoreServer (requires execute permission: `chmod +x GameStoreServer` )

## Deployment Steps

1. **Download** the appropriate server application for your platform
2. **Place** the server application in your chosen directory
3. **Copy** `GameStoreServer.config` from `Assets/GameStore/` to the same directory
4. Run the server application

## Server Operation

### Console Output

#### Windows Example:

```
C:\Dev\CryptoGameStore\GameStoreServer\win-x64\GameStoreServer.exe

Game Store Server - 0.7.8.0
Environment ID - 70a41656-6417-4ff7-a71f-aa245994439b
Starting GameStoreServer using 'WatchBTC' wallet
Wallet loaded successfully on TestNet4 from C:\Dev\CryptoGameStore\GameStoreServer\win-x64\watch-only-btc-wallet-test
Game Store Server started 2025-09-04 11:29:19Z
```

#### Linux Example:

```
tairo@MSI-14: ~/GameStoreServer

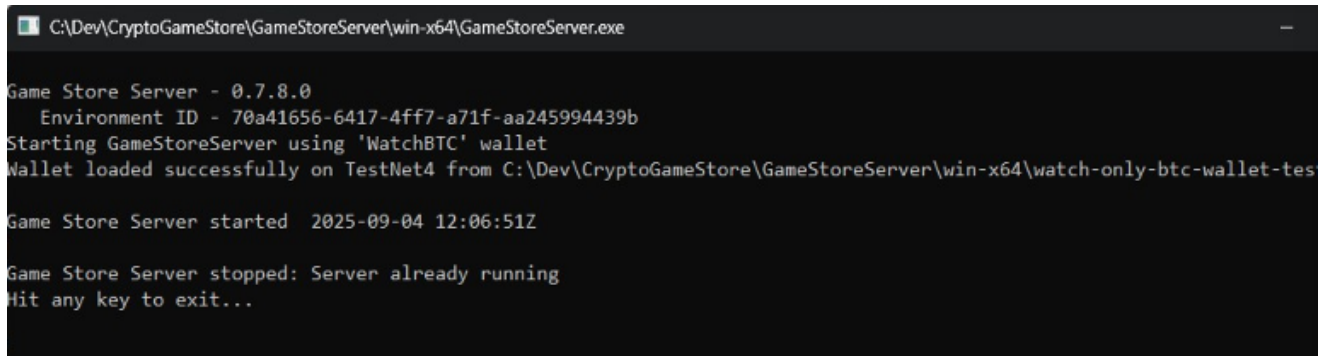
tairo@MSI-14:~$ cd GameStoreServer
tairo@MSI-14:~/GameStoreServer$ ./GameStoreServer

Game Store Server - 0.7.2.0
Environment ID - 70a41656-6417-4ff7-a71f-aa245994439b
Starting GameStoreServer using 'WatchBTC' wallet
Wallet loaded successfully on TestNet4 from /home/tairo/GameStoreServer/watch-only-btc-wallet-testnet4 122
Game Store Server started 2025-09-04 11:43:36Z
```

### Server Locking

The Game Store Server implements an exclusive locking mechanism:

- Only one server instance per Project ID + Environment ID combination
- Prevents conflicting server instances
- Failed startup displays lock status
- Automatic recovery after 2 minutes if original instance becomes unreachable



```
C:\Dev\CryptoGameStore\GameStoreServer\win-x64\GameStoreServer.exe

Game Store Server - 0.7.8.0
Environment ID - 70a41656-6417-4ff7-a71f-aa245994439b
Starting GameStoreServer using 'WatchBTC' wallet
Wallet loaded successfully on TestNet4 from C:\Dev\CryptoGameStore\GameStoreServer\win-x64\watch-only-btc-wallet-testnet4.dat

Game Store Server started 2025-09-04 12:06:51Z

Game Store Server stopped: Server already running
Hit any key to exit...
```

## Production Deployment

For production environments:

### Requirements

- **24 / 7 uptime:** Continuous internet connectivity
- **Automatic startup:** Configure server to start with system boot
- **Monitoring:** Implement logging and health checks
- **Security:** Secure server environment and network access

### Automation Setup

Configure automatic server startup using platform-specific methods:

- **Windows:** Windows Service, Task Scheduler, or Startup folder
- **Linux:** systemd service, init scripts, or cron jobs

## Client Integration

---

### Unity In-App Purchasing Integration

Crypto Game Store integrates seamlessly with Unity's In-App Purchasing 5.0 API, requiring minimal code changes from standard IAP implementations.

### Purchase Flow Overview

1. **Product Definition:** Game defines products for sale
2. **Player Selection:** Player chooses item to purchase
3. **Purchase Initiation:** Game calls Unity IAP purchase method
4. **Payment Dialog:** Crypto Game Store displays payment interface
5. **Player Confirmation:** Player confirms and completes payment
6. **Pending Status:** Game receives pending payment notification

7. **Blockchain Monitoring:** System monitors for payment confirmation
8. **Confirmation:** Game receives successful payment notification
9. **Receipt Verification:** Game verifies transaction receipt

## Crypto Game Store API

The integration requires only one Crypto Game Store-specific method:

```
public static string GameStore.CreateGameStore(ILogger logger = null, bool
logProductDetails = false)
```

Usage:

```
string storeName = GameStore.CreateGameStore();
UnityIAPService.StoreController(storeName);
```

Parameters:

- `logger` : Optional logging interface for debugging
- `logProductDetails` : Enable detailed product information logging

## Payment Dialog Integration

Required Prefab

Include `Assets/GameStore/Prefabs/GameStorePaymentDialog.prefab` in every scene that processes purchases. Selecting this prefab within the Unity Inspector window will allow some **Optional Elements** to be customized for the UI

Dialog Customization

- **Style Location:** `Assets/GameStore/Resources/GameStorePaymentDialog.uxml`
- **Customization:** Modify inline styles or add USS class references
- **Restrictions:** Do not modify VisualElement IDs or non-style properties

Dialog Features

Information Display:

- Cryptocurrency type (currently Bitcoin only)
- Crypto amount to be charged
- Item price in store specified currency
- Real-time exchange rate
- Currency conversion details
- Unique payment address

User Actions:

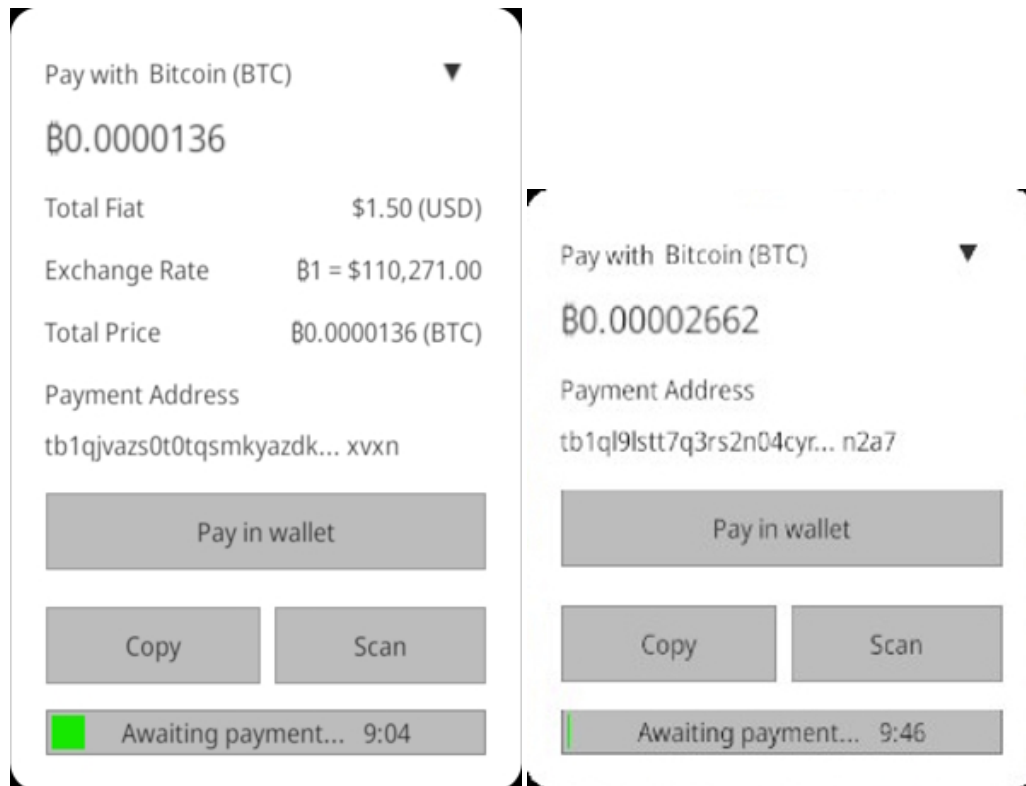
- **Open Wallet:** Launches compatible wallet applications for platform

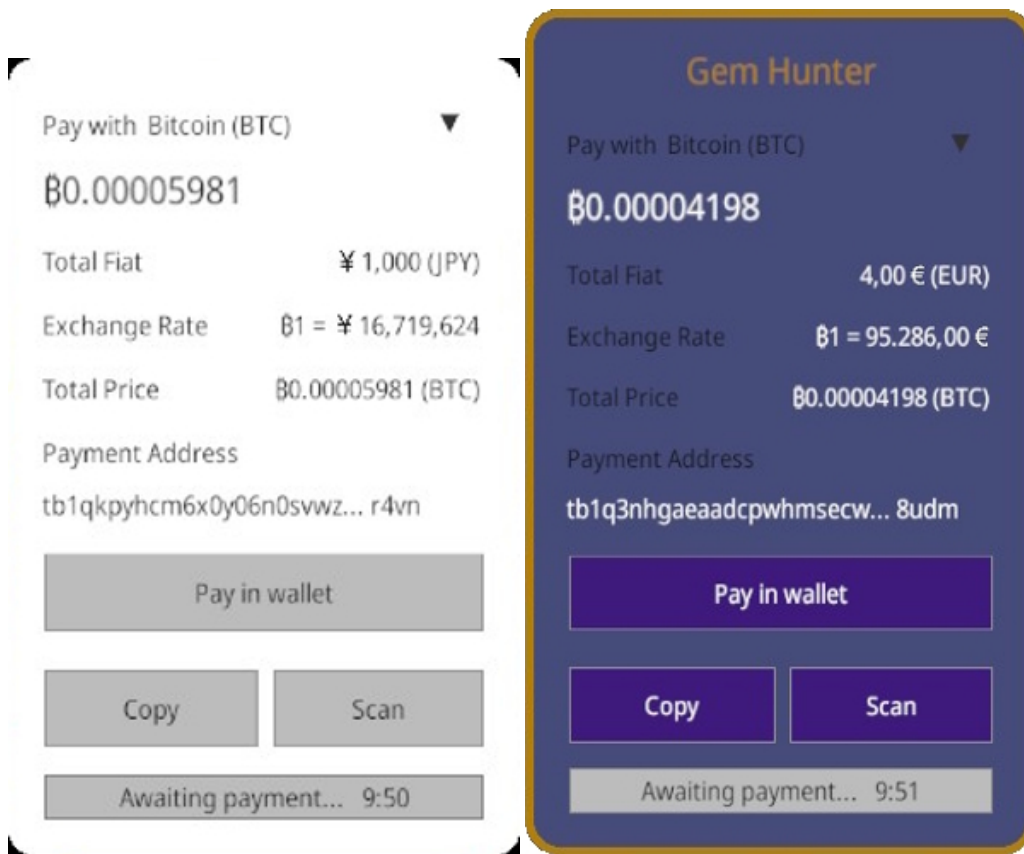
- **Copy Address:** Copies payment address to clipboard
- **QR Code:** Displays QR code to scan from wallets
- **Timer:** Shows validity period for payment quote

#### Optional Elements:

- **Title:** Customizable dialog title
- **Payment Label:** Transaction note for player's wallet

These example show 1) priced in U.S. dollars, 2) priced in bitcoin, 3) priced in Japan Yen, 4) Price in Euro with a custom title and UXML inline-style applied.





## Unity Services Integration

### Prerequisites

Ensure Unity Services initialization and player authentication before using Crypto Game Store:

```
// Initialize Unity Services
await UnityServices.InitializeAsync();

// Authenticate player
await AuthenticationService.Instance.SignInAnonymouslyAsync();
```

### Helper Class

The included `UsgHelp.cs` class provides utilities for Unity Services setup and authentication.

## Example Implementation

A complete example is provided at `Assets/GameStore/Examples/ExampleCryptoGameStoreClient.cs` :

- Demonstrates full integration workflow
- Includes extensive documentation
- Shows best practices for error handling
- Provides debugging and logging examples

# Product Configuration

---

## Unity IAP Catalog Integration

Crypto Game Store uses Unity's standard product definition system with specific adaptations for cryptocurrency pricing.

### Required Product Information

For each store item, define:

1. **ID**: Unique string identifier
2. **Type**: Consumable, Non-consumable, or Subscription (not currently support)
3. **Price**: Numerical price value
4. **Currency ISO**: Price's 3 letter currency code (USD, EUR, JPY, BTC, etc.)

### Configuration Methods

#### Method 1: IAP Catalog (Recommended)

1. Navigate to **Services** → **In-App Purchasing** → **IAP Catalog...**
2. Add products with the following mappings:
  - **ID** → Catalog item *ID*
  - **Title** → Catalog item *Title*
  - **Description** → Catalog item *Description*
  - **Price** → Google Configuration: *Price*
  - **Currency ISO** → Google Configuration: *Pricing Template*

#### Method 2: Runtime PayoutDefinition

For dynamic product configuration use `PayoutDefinition` :

```
var productDefinition = new ProductDefinition(  
    id: "item_001",  
    type: ProductType.Consumable,  
    payouts: new[]  
    {  
        new PayoutDefinition(  
            type: PayoutType.Currency,  
            subtype: "USD",  
            quantity: 9.99m  
        )  
    }  
);
```

`PayoutDefinition` field mappings:

- **Price** → `quantity` field

- **Currency ISO** → subtype field
- `Insure` type is set to `PayoutType.Currency`

## Pricing Best Practices

1. **Consistency:** Use a single currency across all your store items when possible
2. **Clarity:** Clearly indicate currency to players
3. **Exchange Rates:** Account for real-time rate fluctuations
4. **Testing:** Verify pricing accuracy in test environment

## Catalog File Location

Unity saves the IAP Catalog to `Assets/Resources/IAPProductCatalog.json`. Keep this file in the default location for proper IAP Catalog window functionality.

## Bitcoin Configuration

### Confirmations

The bitcoin network secures your store's purchases by adding them to its blockchain. The longer you wait after adding a purchase to the blockchain, the more secure that purchase will be.

By default, Crypto Game Store wait for one security cycle, a single *confirmation*, to occur before reporting to your store that a purchase is complete. This should be sufficient for purchases under \$100 or thereabouts. If you store sells items for amounts larger than this, you may want to consider increasing the number of confirmations needed to report a purchase complete.

You can use change the confirmations waited in the Unity Inspector of the [BTC Network Provider](#) using the **Confirmation For Paid** setting. On average each confirmation will take approximately 10 minutes.

## Testing Your Store

---

### Test Network Setup

Use Bitcoin's testnet4 for safe testing without real Bitcoin:

1. **Configure Server:** Enable "Use Test Network" in Game Store Server configuration
2. **Wallet Configuration:**
  - **Knots:** Automatically uses *testnet4* when enabled in **Game Store Server** configuration
  - **Watch-Only:** Export *testnet4* X PUB (starts with "tpub..." or "vpub...")
3. **Save and Restart:** Apply test configuration to server

### Obtaining Test Bitcoin

#### Testnet4 Faucets

Request free *testnet4* Bitcoin from these faucets:

- [mempool.space testnet4 faucet](#)

- [coinfaucet.eu testnet4](https://coinfaucet.eu/testnet4)

## Testnet4 Address Format

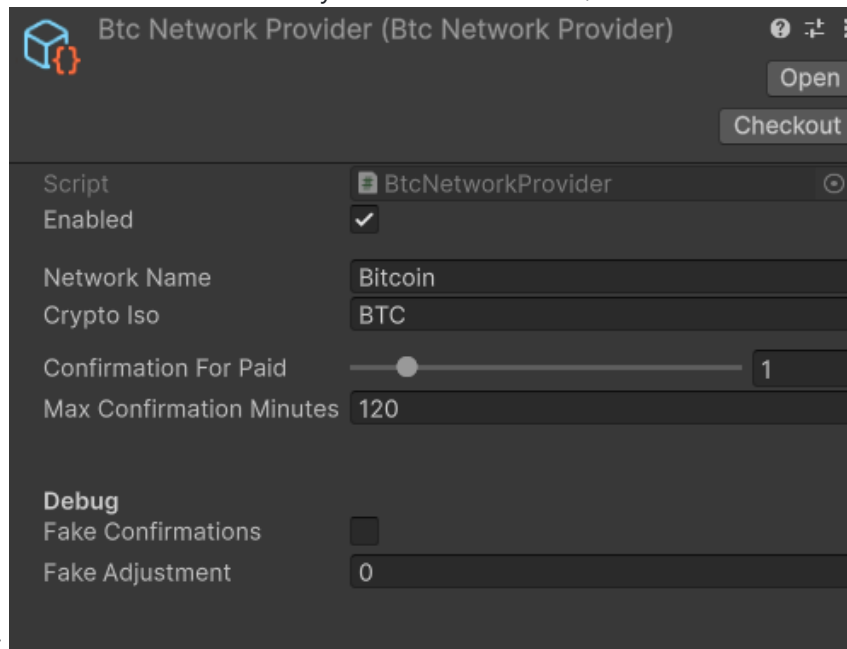
*Testnet4* addresses begin with "tb1..." prefix.

## Testing Considerations

- **Transaction Time:** Testnet4 transactions may take 1+ hours to confirm (vs 10 minutes on mainnet)
- **Network Reliability:** Test network may experience longer delays

## Development Testing Features

Since *testnet4* is often considerably slower than *mainnet*, fake confirmation can be sent for rapid development



testing:

1. Navigate to `Assets/GameStore/Resources/PaymentNetworks/BtcNetworkProvider`
2. Enable **Fake Confirmations** option
3. **Behavior:** Generates fake confirmation 30 seconds after a transaction starts
4. **Limitations:**
  - Only works with DEBUG builds
  - Only works on *testnet4* network
  - Bypasses actual blockchain monitoring
5. **Features:**
  - **Fake Adjustment:** Test over/under payment scenarios. 0 = pay exact invoice amount; -1 = under pay invoice by 10%; 1 = over pay invoice by 10%
  - Status: Simulate success/failure conditions

## Production Testing Checklist

Before deploying to mainnet:

1. **Test Network Validation:** Complete testing on *testnet4*



2. **Server Configuration:** Verify *mainnet* wallet configuration
3. **Exchange Rates:** Confirm real-time price feeds
4. **Security Review:** Validate server and wallet security
5. **Backup Procedures:** Ensure wallet backup and recovery processes
6. **Monitoring Setup:** Implement production monitoring and alerting

## Troubleshooting

---

### "Cannot contact store; retry later"

#### Possible Causes:

- Game Store Server is not running
- Project ID mismatch between server and game
- Environment ID mismatch between server and game
- Network connectivity issues

#### Resolution Steps:

1. Verify server is running and accessible
2. Confirm Project ID matches in both server config and game
3. Verify Environment ID of server matches editor or built game environment (found in Editor **Project Setting** → **Services** → **Environments** → **Environment**)
4. Check network connectivity and firewall settings

### Server Startup *AuthorizationException*

#### Possible Causes:

- Incorrect Project ID in server configuration
- Service Account keys don't match selected project
- Insufficient Service Account permissions

#### Resolution Steps:

1. Verify Project ID matches the project used for Service Account creation
2. Confirm Service Account has "Cloud Save Editor" role
3. Regenerate Service Account keys if necessary
4. Verify API Key and Secret are correctly entered

## Wallet Configuration Issues

#### Watch-Only Wallet:

- Verify X PUB format matches network type (*mainnet* vs. *testnet4*)
- Confirm X PUB is valid and properly exported
- Check wallet software documentation for export procedures

### Knots Wallet:

- Verify Knots installation in default directory
- Confirm RPC server is enabled in Knots settings
- Check Knots wallet creation and configuration
- Review Knots logs for startup errors

## Payment Processing Problems

### Transaction Not Detected:

- Verify payment sent to correct address
- Check sufficient network confirmations
- Confirm correct network (*mainnet* vs. *testnet4*)
- Review blockchain explorer for transaction status

### Incorrect Payment Amount:

- Verify exchange rate calculation
- Check for network fee deductions
- Confirm payment precision and rounding

## Debugging Tools

### Server Logging

- Enable verbose logging in Game Store Server for detailed operation information.
- Monitor Unity console for Crypto Game Store debug messages and errors.
- Monitor Game Store Server console for errors.

### Blockchain Explorers

- Mainnet: [blockstream.info](https://blockstream.info)
- Testnet4: [mempool.space/testnet4](https://mempool.space/testnet4)

## Support Resources

### Exchange Rate Information

Real-time cryptocurrency prices provided by [CoinGecko](https://www.coingecko.com/).

### Additional Documentation

- Unity In-App Purchasing documentation
- Bitcoin Knots setup guides
- Wallet-specific X PUB export guides

# Glossary

---

## Blockchain Terms

**Bitcoin (BTC)** : The first and most widely used cryptocurrency, operating on a decentralized blockchain network.

**Blockchain** : A distributed ledger technology that records transactions across multiple computers in a tamper-resistant manner.

**Confirmations** : The number of blocks mined on the blockchain after a purchase's block, indicating increasing security and finality.

**Faucet** : A website that distributes small amounts of test cryptocurrency for development and testing purposes.

**Mainnet** : The primary Bitcoin network where transactions use real Bitcoin with monetary value.

**Private Key** : A cryptographic key that allows spending of Bitcoin from a wallet address. Must be kept secret and secure.

**Public Key** : A cryptographic key derived from a private key that can be shared publicly without compromising wallet security.

**Testnet4** : Bitcoin's test network for development and testing, using test Bitcoin with no monetary value.

**Transaction Fee** : A fee paid to Bitcoin miners for including a transaction in a block.

## Wallet Terms

**Extended Public Key (XPUB)** : A public key that can generate multiple Bitcoin addresses for receiving funds without exposing private keys.

**Hardware Wallet** : A physical device that stores private keys offline for enhanced security.

**Hot Wallet** : A wallet connected to the internet, allowing both sending and receiving of cryptocurrency.

**Knots** : A Bitcoin full node implementation that validates transactions and maintains the blockchain.

**Non-Custodial Wallet** : A wallet where the user controls their own private keys.

**QR Code** : A matrix barcode containing payment information that can be scanned by mobile wallet applications.

**Watch-Only Wallet** : A wallet that can monitor addresses and generate receiving addresses but cannot spend funds.

## Unity and Development Terms

**Environment ID** : A Unity Services identifier that separates different deployment environments (development, staging, production).

**In-App Purchasing (IAP)** : Unity's system for handling purchases within games and applications.

**Project ID** : A unique identifier for a Unity project within Unity Services.

**Service Account** : A special account used for server-to-server communication with Unity Services APIs.

**Unity Services** : Cloud-based services provided by Unity, including Authentication, Cloud Save, and Analytics.

## Crypto Game Store Terms

**Fake Confirmations** : A development feature that simulates transaction confirmations for testing without waiting for blockchain confirmation.

**Game Store Server** : The self-hosted server component that manages wallet interactions.

**Payment Address** : A unique Bitcoin address generated for each transaction to receive customer payments.

**Payout Definition** : A Unity IAP structure defining the currency and amount for a product purchase.

**Server Locking** : A mechanism preventing multiple Game Store Server instances from running with the same configuration simultaneously.

*This documentation covers the complete setup and operation of the Crypto Game Store system. For additional support or updates, refer to the latest version of this documentation and the provided example implementations.*