Matryoshka Technical Guidelines

[Matryoshka Technical Guidelines](https://docs.google.com/document/d/1gV89BdyYUb6emc8qDpAW6B57EyrN83X_7oJigud_zNQ/edit#heading=h.tum79cqgdlx2)

[Summary](https://docs.google.com/document/d/1gV89BdyYUb6emc8qDpAW6B57EyrN83X_7oJigud_zNQ/edit#heading=h.232cu05cj7uv)

[Common requirements](https://docs.google.com/document/d/1gV89BdyYUb6emc8qDpAW6B57EyrN83X_7oJigud_zNQ/edit#heading=h.pcxi4dw0zyta)

[Game Engine: Unity](https://docs.google.com/document/d/1gV89BdyYUb6emc8qDpAW6B57EyrN83X_7oJigud_zNQ/edit#heading=h.dqjyfr57dftf)

[Ads](https://docs.google.com/document/d/1gV89BdyYUb6emc8qDpAW6B57EyrN83X_7oJigud_zNQ/edit#heading=h.ohg0ulm8e8sd)

[Steam](https://docs.google.com/document/d/1gV89BdyYUb6emc8qDpAW6B57EyrN83X_7oJigud_zNQ/edit#heading=h.rkcokqsx8x8d)

[Integration checklist](https://docs.google.com/document/d/1gV89BdyYUb6emc8qDpAW6B57EyrN83X_7oJigud_zNQ/edit#heading=h.s03w3vjri8y5)

[Install](https://docs.google.com/document/d/1gV89BdyYUb6emc8qDpAW6B57EyrN83X_7oJigud_zNQ/edit#heading=h.byn7ft6qvxh5)

[Minimal setup](https://docs.google.com/document/d/1gV89BdyYUb6emc8qDpAW6B57EyrN83X_7oJigud_zNQ/edit#heading=h.aaibwzv1m04a)

[User ID usage](https://docs.google.com/document/d/1gV89BdyYUb6emc8qDpAW6B57EyrN83X_7oJigud_zNQ/edit#heading=h.akta3dpl8kv9)

[IAP](https://docs.google.com/document/d/1gV89BdyYUb6emc8qDpAW6B57EyrN83X_7oJigud_zNQ/edit#heading=h.6yl7y04qe581)

[Simplified integration](https://docs.google.com/document/d/1gV89BdyYUb6emc8qDpAW6B57EyrN83X_7oJigud_zNQ/edit#heading=h.h9jowbkory3o)

[DLC](https://docs.google.com/document/d/1gV89BdyYUb6emc8qDpAW6B57EyrN83X_7oJigud_zNQ/edit#heading=h.5rvaoucy793d)

Summary

This document provides guidance for external developers on preparing their Unity projects for publication by Matryoshka. The instructions cover essential steps and requirements, such as integrating ads, setting up the Steam platform, implementing in-app purchases (IAP), and managing downloadable content (DLC). By following these guidelines, developers can ensure their projects are compatible with Matryoshka's publishing process.

Common requirements

Game Engine: Unity

* **Unity 2019.4**: Minimum supported version
* **Unity 2020.3**: Recommended target version until mid-2023
* **Unity 2021.3**: Recommended target version after mid-2023

Ads

For ad integration, use the proprietary Mercury AD Network package:

- **Package**: [Mercury AD Network](https://drive.google.com/drive/folders/1hb0FWX0HRERnbsnIj9aFu8-O6qTotxtv)

- **Documentation**: [Mercury AD Network Integration Guide](https://docs.matryoshka.com/MercuryAdNetwork/articles/integration_guide.html)

- **Project-specific config**: contact your Matryoshka producer

Supported platforms:

* Steam (Windows / MacOS)

Steam

Integration checklist

1. Request tester access to the app by sending an email address
2. Send us the first build (any condition, it’s required for SDK to be initialized)
3. The producer will set up the game page and upload that build
4. The producer will send an invitation to the organization
5. Accept the invitation
6. The producer will approve you and inform you of your tester access to the app
7. Download the app build through the Steam client
8. Replace the build files with your current in-progress build by navigating to Steam client > Library > your app > Properties > Local files > Browse
9. The producer will provide you with the AppID, IAP, and DLC IDs
10. Import the Steamworks SDK into your project and change the AppID
11. Test IAP purchases using the provided IDs
12. (Optional) Test DLC availability. Note that DLC is installed by default for test accounts and is not visible in the store before it is published

Detailed instructions are provided below.

Install

It’s recommended to use the <https://github.com/rlabrecque/Steamworks.NET> wrapper for Steam SDK:

* **Installation instructions**: [Steamworks.NET Installation](https://github.com/rlabrecque/Steamworks.NET#installation)
* **Sample projects**: [Steamworks.NET Samples](https://github.com/rlabrecque/Steamworks.NET#samples)

Minimal setup

1. Add Steamworks.NETpackage through **Package Manager** -> **Add package from git URL**: **https://github.com/rlabrecque/Steamworks.NET.git?path=/com.rlabrecque.steamworks.net** (or manually add "com.rlabrecque.steamworks.net": "https://github.com/rlabrecque/Steamworks.NET.git?path=/com.rlabrecque.steamworks.net" to **Packages/manifest.json**)
2. Copy [SteamManager.cs](https://raw.githubusercontent.com/rlabrecque/SteamManager/master/SteamManager.cs) to your project
3. Update **SteamManager**: change **AppId\_t.Invalid** to your own AppId (you can get it from your producer at Matryoshka)
4. Attach SteamManager to the game object. The script will handle initialization automatically.
5. Before doing any Steam SDK calls, ensure that initialization is finished completely (to do that you can add some sort of callback at the end of SteamManager.Awake method

User ID usage

If you’re using **Mercury AD Network** on Steam platform, you need to obtain and send **User ID**:

|  |
| --- |
| using Matryoshka.MercuryAdNetwork; using Steamworks; ... var adNetwork = AdNetworkFactory.Create(     ...     userIdCallback: () => SteamUser.GetSteamID().ToString(),     ... ); |

IAP

Your project's in-app purchases should be implemented using the [Steam Inventory Service](https://partner.steamgames.com/doc/features/inventory).

Provide your Matryoshka producer with the purchase information for each item:

* Name & description (for players)
* Price templates (<https://partner.steamgames.com/doc/features/inventory/schema#SpecifyPrices>)
* itemdefid (available in purchase URL)

Simplified integration

|  |
| --- |
| using System.Linq; using System.Globalization; using UnityEngine; using Steamworks;  public sealed class SteamPurchaseManager {   string DefaultCurrency = "USD";   string DefaultCulture  = "en-US";    // Current session currency settings   string Currency;   CultureInfo CurrencyCulture;    // Callbacks   CallResult<SteamInventoryRequestPricesResult\_t> \_requestPricesCallback;   CallResult<SteamInventoryStartPurchaseResult\_t> \_startPurchaseCallback;   Callback<SteamInventoryResultReady\_t>           \_steamInventoryResultReadyCallback;   Callback<SteamInventoryFullUpdate\_t>            \_steamInventoryFullUpdateCallback;    // Call this method to initialize purchases   public void InitializePurchases() {      var handle = SteamInventory.RequestPrices();      \_requestPricesCallback = CallResult<SteamInventoryRequestPricesResult\_t>.Create(OnRequestedPrices);      \_requestPricesCallback.Set(handle);     \_steamInventoryFullUpdateCallback = Callback<SteamInventoryFullUpdate\_t>.Create(OnSteamInventoryFullUpdate);     RequestUserInventory();   }    // Method to fetch user inventory   void RequestUserInventory() {     if ( SteamInventory.GetAllItems(out var inventoryResult) ) {       // Handle result     }   }    // This callback will be called when purchases are ready   void OnRequestedPrices(SteamInventoryRequestPricesResult\_t pCallback, bool bIOFailure) {      Currency        = pCallback.m\_rgchCurrency;      CurrencyCulture = ResolveCurrencyCulture(Currency, DefaultCulture);   }    // Helper method to get current culture from currency   CultureInfo ResolveCurrencyCulture(string currency, string defaultCulture) {      var culture = CultureInfo.GetCultures(CultureTypes.AllCultures & ~CultureTypes.NeutralCultures)         .FirstOrDefault(x => new RegionInfo(x.Name).ISOCurrencySymbol == currency);      return culture ?? new CultureInfo(defaultCulture);   }    // When initialization is finished you can perform purchase calls   public void PurchaseItem(int productID) {      var itemDef = new SteamItemDef\_t(productID);      var itemDefs = new [] {         itemDef      };      var quantities = new[] {         1U      };      var handle = SteamInventory.StartPurchase(itemDefs, quantities, (uint)itemDefs.Length);      \_startPurchaseCallback = CallResult<SteamInventoryStartPurchaseResult\_t>.Create(OnItemPurchaseCompleted);      \_startPurchaseCallback.Set(handle);      \_steamInventoryResultReadyCallback = Callback<SteamInventoryResultReady\_t>.Create(OnSteamInventoryResultReady);   }    void OnItemPurchaseCompleted(SteamInventoryStartPurchaseResult\_t pCallback, bool bIOFailure) {      if (bIOFailure || (pCallback.m\_result != EResult.k\_EResultOK)) {         Debug.LogError("Item purchase failed: " + pCallback.m\_result);         return;      }      Debug.Log("Item purchase successful!");   }    // Inventory callbacks    void OnSteamInventoryResultReady(SteamInventoryResultReady\_t result) {     if ( (result.m\_result == EResult.k\_EResultOK) && (result.m\_handle != SteamInventoryResult\_t.Invalid) ) {       ProcessInventoryResult(result.m\_handle);     }     \_steamInventoryResultReadyCallback.Unregister();     \_steamInventoryResultReadyCallback = null;     RequestUserInventory();   }    void OnSteamInventoryFullUpdate(SteamInventoryFullUpdate\_t result) {     if ( result.m\_handle != SteamInventoryResult\_t.Invalid ) {       ProcessInventoryResult(result.m\_handle);     }   }    // Method to process user inventory   void ProcessInventoryResult(SteamInventoryResult\_t inventoryResult) {     var resultStatus = SteamInventory.GetResultStatus(inventoryResult);     try {       if ( resultStatus != EResult.k\_EResultOK ) {         return;       }       uint itemCount = 0;       if ( !SteamInventory.GetResultItems(inventoryResult, null, ref itemCount) ) {         return;       }       SteamItemDetails\_t[] itemDetails = new SteamItemDetails\_t[itemCount];       if ( SteamInventory.GetResultItems(inventoryResult, itemDetails, ref itemCount) ) {         foreach ( var item in itemDetails ) {           // Handle item         }       } else {         return;       }     } finally {       SteamInventory.DestroyResult(inventoryResult);     }   } } |

After handling a purchase report revenue to **Mercury AD Network SDK** (but do not track IAP revenue for DLC purchases):

|  |
| --- |
| adNetwork.RegisterIAP(price); // pass USD price (purchase verification is recommended) |

DLC

You can use the code from the previous section to implement DLC purchases. and check purchased DLC with the code below (also works for DLC bought from the Steam store):

|  |
| --- |
| using Steamworks;  public sealed class SteamDlcManager {   public bool IsDlcPurchased(uint id) {      return SteamApps.BIsDlcInstalled(new AppId\_t(id));   } } |

When you detect that DLC is installed, report it to **Mercury AD Network SDK**:

|  |
| --- |
| adNetwork.RegisterDLC(dlcId); // pass unique ID associated with this DLC |

To trigger DLC purchase in-game use the overlay method:

|  |
| --- |
| SteamFriends.ActivateGameOverlayToStore(dlcId), EOverlayToStoreFlag.k\_EOverlayToStoreFlag\_None); |