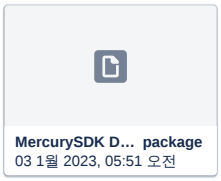
QUICK START



Sample 프로젝트

SDK PACKAGE

Quick Start Flow (목차)

1. Init.

AuthKey Setting

2. Basic Start

Connect To Server

Server Login

RoomJoin and Create

Network Object Synchronize

Transfrom Synchronize

Animator Synchronize

Chat

Error Handling

3. Advanced Start

RPC (Remote Procedure Call)

CustomData (사용자 정의 데이터 전송)

4. Example Code

Avatar Parts Synchronize (아바타 파츠 동기화 코드)

Shared Object Synchronize (기본 사용법)

Shared Object Synchronize (저장된 공유오브젝트 동기화 - 예) 나중에 들어온 플레이어)

Latancy Check (핑을 이용한 지연시간 체크)

5. Mercury Result Define

WEB RESULT

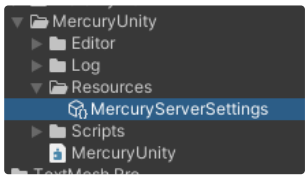
Mercury.Constants.WebResult

SERVER & SDK

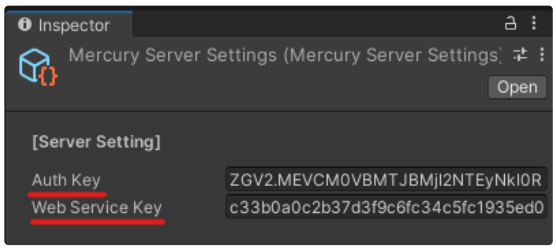
Mercury.Constants.Error

1. Init

Mercury SDK 를 Unity에 Import 한 뒤



MercuryUnity폴더의 Resources 폴더 내에 있는 `MercuryServerSetting` 을 선택합니다.



Inspector에 표시되는 `Auth Key`와 WebServiceKey Key를 입력합니다.

위 과정을 마치면 MercuryServerSetting이 완료 됩니다.

2. Basic Start

2-1. Connect to Server

가장 먼저 진행해야 되는 작업은 Client를 Mercury Server 에 연결하는 작업입니다.

※ override된 메소드를 사용하기 위해 구현부가 작성 될 클래스에 `MercuryBehaviourCallBacks` 를 상속합니다

```
protected void Start()
{
    MercuryNetwork.Connection();
}

public override void OnNetworkConnectionFail()
{
    Debug.Log("OnNetworkConnectionFail");
}

public override void OnConnected()
{
    Debug.Log("OnNetworkConnection");
}
```

2-2. Server Login

두번째로 진행해야 하는 작업은 연결된 서버에 Login을 진행해 해당 유저의 닉네임, 고유 ID값을 서버에 등록합니다.

로그인시 입력하는 UID(ulong) 값을 0 으로 등록할 시에 Mercury Server 에서 자체적으로 발급됩니다.

```
protected void Login()
{
    MercuryNetwork.Login("NickName", UID);
}

public override void OnLoginCompleted()
{
    Debug.Log("OnLoginComplete");
}
```

2-3. Room Join Create

로그인 후 방을 생성하거나 이미 생성되어 있는 방으로 입장합니다.

maxPlayers : 0으로 입력 시 Mercury 관리자 페이지에 설정되어 있는 최대 인원수로 방이 생성됩니다.

prefabName : 생성될 플레이어의 프리팹 경로입니다. 프리팹은 Resources 폴더 내에 위치해야 합니다.

userProperty : 플레이어의 기초 정보를 입력할 수 있습니다. 플레이어의 정보, 아바타의 형태, 장비 등 플레이어 간 공유될 정보를 입력받습니다.

OnInitPlayer(Player newPlayer) : 플레이어 생성 및 초기화 함수 입니다.

다른 플레이어가 입장시에도 호출됩니다.
– Player : JoinOrCreateRoom 호출 당시 입력한 prefabName, userProperty 등의 정보가 들어있습니다.

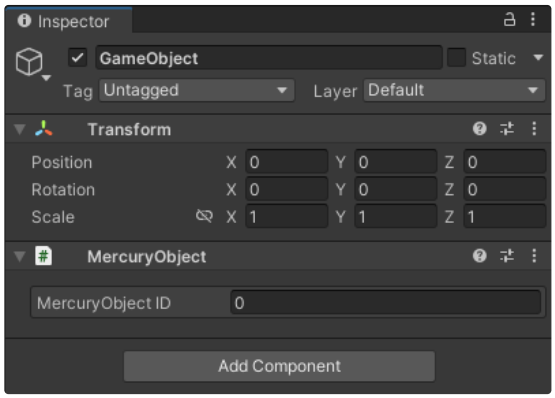
```
protected void JoinOrCreateRoom(string roomName, int maxPlayers)
{
    MercuryNetwork.JoinOrCreateRoom(new RoomOption
    {
        name = inpRoomId.text,
        maxPlayers = maxPlayers,
    }, $"Player/SamplePlayer{Random.Range(2,5):D2}",
        "hat:100,glove:200,clothes:32,shoes:302");
}

public override void OnJoinedRoom(RoomInfo roomInfo)
{
    Debug.Log("OnJoinedRoom");
}

public override void OnInitPlayer(Player newPlayer)
{
    var playerObj = MercuryNetwork.Instantiate(newPlayer, playerParent);
}
```

RoomOption : 방생성, 방입장시 설정할수 있는 옵션.

string name : 방 생성 시 정해지는 이름
maxPlayers : 방 생성 시 최대 인원수 (라이선스에 따른 숫자 제한 있음.)
isObserver : ture 일 경우 관전자 (관전자는 주변 아바타에 보이지 않음.)
isUnique : true 일 경우, 채널 확장이 안됨. (서비스내에 단일 방)

2-4. Network Object Add

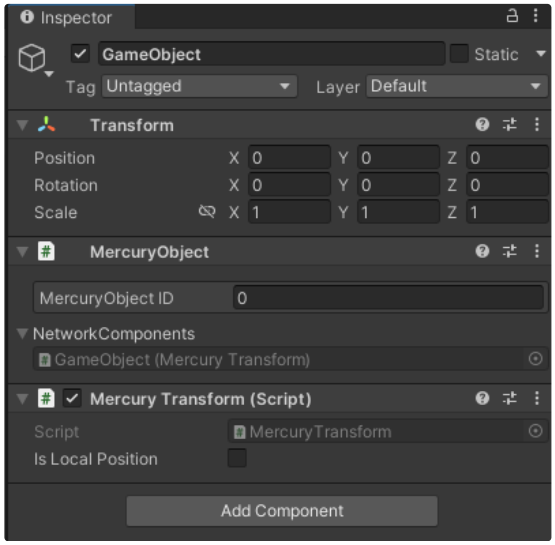
Unity Game Object를 동기화 하기 위해서는 MercuryObject Component 를 추가 합니다.



MercuryObjectID : 해당 오브젝트의 고유 값으로, 동기화 오브젝트의 구분을 위해 사용됩니다.

2-5. Transform Synchronize

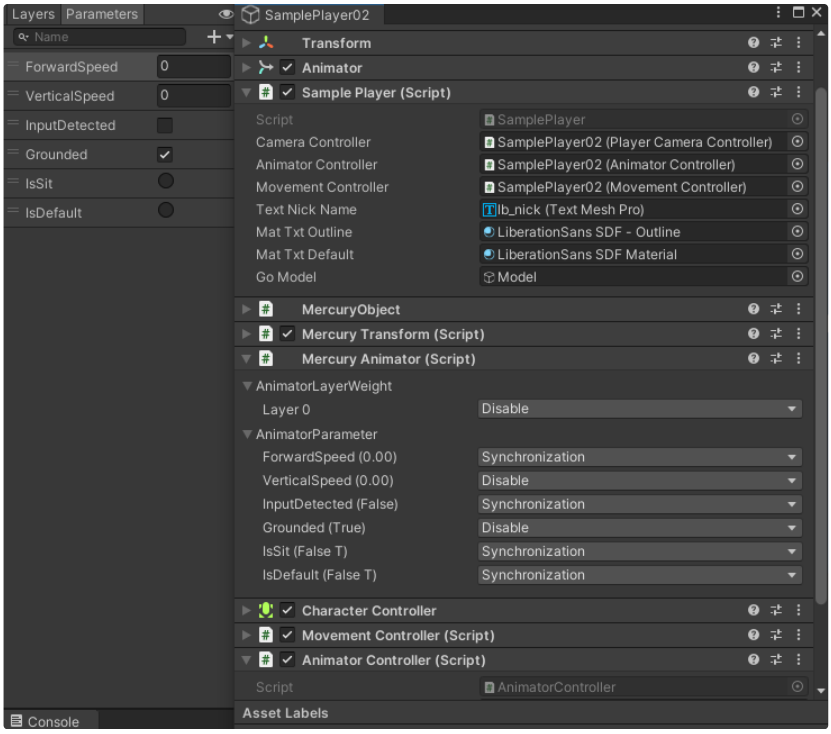
Transform 동기화를 하기 위해서는 Mercury Object가 추가되어 있는 상태에서 `Mercury Transform Component`를 추가 합니다.



2-6. Animator Synchronize

머큐리 SDK가 애니메이션을 동기화하는 방법에 대해 설명합니다.

머큐리 애니메이션은 애니메이터 정보를 관찰하여 데이터를 반영합니다. Trigger 형태의 애니메이터 값은 프레임을 놓치는 경우가 있어 직업 머큐리 애니메이션 정보를 업데이트 해야 합니다.



`Mercury Animator (Script)` AnimationParameter 아래에 속성들을 Synchronization 옵션으로 설정합니다.

2-7. Chat

머큐리 SDK가 메시지를 송수신하는 방법에 대해 설명합니다.
머큐리 채팅은 `MirrorGear.Mercury.MercuryNetwork` 싱글턴 인스턴스를 이용해 구현됩니다.

머큐리 채팅 송신

`MirrorGear.Mercury.MercuryNetwork.SendChatMessage`함수를 이용해 송신합니다.

```
public class UIChat : MercuryChatBehaviour
{
    private ChatType _chatType;
    public TMP_InputField inpChat;

    public void OnButton_btnSend()
    {
        if(string.IsNullOrEmpty(inpChat.text.Trim()) == true)
            return;

        MercuryNetwork.SendChatMessage(inpChat.text, _chatType);
        inpChat.text = "";
    }
}
```

머큐리 채팅 수신

`MirrorGear.Mercury.Chat.IMercuryChatCallback` 인터페이스를 구현합니다.
채팅을 수신하기 위해서는 아래와 같이 구현 합니다.

```
public abstract class MercuryChatBehaviour : MonoBehaviour, IMercuryChatCallback
{
    public abstract void OnReceiveRoomChat(ChatData data);
    public abstract void OnReceiveWorldChat(ChatData data);
```

```
    public abstract void OnReceiveGlobalChat(ChatData data);
    public abstract void OnError(Error error);
}
```

```
public class UIChat : MercuryChatBehaviour
{
    public UIChatItem pfChatItem;
    public Transform chatItemParent;

    public override void OnReceiveRoomChat(ChatData data)
    {
        var uiItem = Instantiate(pfChatItem, chatItemParent);
        uiItem.gameObject.SetActive(true);
        uiItem.SetText($"RoomChat -> {data.message}");
    }

    // 그 외 IMercuryChtaCallback
    // 구현 (e.g. ReceiveWorldChat, ReceiveGlobalChat, OnError)
    // ...
}
```

`MirrorGear.Mercury.MercuryNetwork.AddChatCallBack` 함수를 사용해 등록합니다.

```
public abstract class MercuryChatBehaviour : MonoBehaviour, IMercuryChatCallback
{
    public virtual void OnEnable()
    {
        MercuryNetwork.AddChatCallBack(this);
    }

    // IMercuryChatCallback 구현
    // ...
}
```

`MirrorGear.Mercury.MercuryNetwork.AddChatCallBack(this)` 메서드 실행 이후, 새 채팅 메시지가 수신 될 때 `MirrorGear.Mercury.Chat.IMercuryChatCallback` 인터페이스를 구현한 `UIChat` 의 메서드가 자동으로 실행합니다.

ChatType (채팅 범위)

RoomChat : 현재 내가 진입한 룸 내부 통신.

WorldChat: 룸이 확장된 채널들끼리 통신.

GlobalChat: 전체 채팅. (운영 공지용. 연속 입력 방지를 위한 GapTime 존재함.)

2-8. Error Code Handling

`MercuryBehaviourCallBacks` 상속 이후 아래와 같이 override 합니다.

```
public override void OnNetworkError(Error Code)
{
    Debug.Log("NetWorkError:" + code );
}

public override void OnPacketError(Error Code)
```

```
{
    Debug.Log("PacketError:" + code) ;
}

```

자세한 Error Code **5. Mercury Result Define 참고.**

3. Advanced Start

3-1. RPC (Remote Procedure Call)

Remote Procedure Calls을 사용해 원격 클라이언트에 함수를 호출 할 수 있습니다.

원격 호출을 활성화 하려면 `[MercuryRPC]` 속성을 적용 해야 합니다.

```
[MercuryRPC]
public void LogText(string log)
{
    Debug.Log($"LogText : {log}")
}
```

RPC의 호출 예

```
mercuryObject.RPC("LogText", RPCTarget.Other, "RPC Log");
```

3-2. CustomData Send

사용자 정의 이벤트를 보낼 수 있습니다.

DynamicData Class를 이용해 Mercury Server 전달 할 데이터를 만든 후 `MercuryNetwork.SendData` 를 사용해 사용자 정의 이벤트를 보낼 수 있습니다.

accountId : 보내는 사람의 Account ID 입니다.

broadCastType : 해당 메세지를 받을 Player의 범위 입니다.

targetAccount : BroadCastType 이 Target 일 때 이벤트를 받을 대상입니다.

```
public void SendDynamicData()
{
    var data = new DynamicData(DynamicCode.None)
    {
        accountId = MercuryNetwork.LocalPlayer.accountId,
        broadCastType = BroadCastType.RangeME,
        targetAccount = 0,
    };

    var datas = new List<object>();
    datas.Add(1);
    datas.Add("string");
    ///...

    data.datas = datas;

    MercuryNetwork.SendData(MercurySyncType.Dynamic, MercuryNetwork.LocalPlayer, data);
}
```

```
19
20 public void OnReceiveDynamicData(DynamicData data)
21 {
22     Debug.Log($"Receive Dynamic Data : {data.datas[0] as int}, {data.datas[1] as string}");
23 }
```

4. Example Code

코드가 길어 지거나, 복합적인 기능들의 코드들을 나열 합니다.

QUICK START - Example Code