UGI WebSocketManager
Web Socket Request Manager + Transaction Tracking
MLH Fall Fellowship
Star Atlas Team
12/22

1 Overview

To implement the request manager a *Game-Instance* object was created; ensuring that the manager is persistent and accessible from any world asset. Even though a specific instance is required, there are a variety of static functions that work as helper function.

This manager sends JSON API request to the Solana Web Socket server and stores the response in the corresponding object in the Subscription Map.

```
class  FOUNDATION_API UGI_WebSocketManager:  public UGameInstance
{
        GENERATED_BODY()
public:
        virtual void Init() override;
        virtual void Shutdown() override;
        virtual void OnStart() override;
        TSharedPtr<IWebSocket> WebSocket;
        DECLARE_EVENT(UGI_WebSocketManager, FSocketConnected);

        static int64 GetNextSubID();
        static int64 GetLastSubID();
        inline  static TMap<int, FSubscriptionData*> ActiveSubscriptions;

        void Subscribe(FSubscriptionData* SubData);
        void Unsubscribe(int subID);
        FSubscriptionData* GetSubData(int SubID);
        void InitializeHeartbeat();
        UFUNCTION()
        void HeartbeatHelper();

private:
        inline static FSocketConnected OnConnected;
        static void OnResponse(const FString &Response);
        static void ParseNotification(const FString &Response);
        static void ParseSubConfirmation(const FString &Response);
        static void OnConnected_Helper();
};
```

1.1 Members

Member	Description	Type	Scope
WebSocket	Corresponds to the UE web socket object that will be connected.	TSharedPtr IWebSocket	public
ActiveSubscriptionsMap	Hash map that stores the subscriptions currently active. They key of the map is the request ID and the value is the FSubscritionData structure.	TMap (int, FSubscriptionData)	public

1.2 Methods

1.2.1 Connection and Parsing methods

Method	Description	Type	Scope
Init	Initializes the web socket, connecting it to the specified network. (Defaults to devnet)	virtual void	public
Shutdown	Closes the web socket connection.	virtual void	public
OnConnected	Binding that executes when connection successfully. Currently it declares an event and prints a debug message.	inline static FSocketConnected	private
OnReponse	Binding that executes when the web socket receives a message. Based on the message it decides if it is a confirmation or notification, calling the corresponding helper functions.	static void	private
ParseSubConfirmation	Takes as argument the string from the successful subscription. Reads the subscription number from the *result* attribute from the JSON response and updates the corresponding subscription data.	static void	private
ParseNotification	Takes as argument the string from a subscription update. Uses the ID to find the subscription in the map and updates the *response* attribute	static void	private

1.2.2 Auxiliary Methods

Method	Description	Type	Scope
Subscribe	Takes as an argument the subscription structure that we want to subscribe	void	public
Unsubscribe	Takes an argument the ID of the subscription we want to eliminate from the subscription map	void	public

1.2.3 Subscription methods

Method	Description	Type	Scope
GetNextSubID	Returns the next ID for creating requests	static int64	public
GetLastSubID	Returns the last ID for creating requests	static int64	public
GetSubData	Given a ID, it looks for that subscription in the active subscription map and returns the response value.	void	public
InitializeHeartbeat	Initializes a recurrent timer that triggers the HearbeatFunction periodically every 30 seconds	void	public
HeartbeatHelper	Sends a ping request to keep alive the web socket connection.	UFUNCTION (void)	public

1.3 Subscription Data Structure

The request manager stores the data to make the request and the received information in a custom structure.

```
struct FOUNDATION_API FSubscriptionData
{
        FSubscriptionData() {}
        FSubscriptionData( UINT id ) { Id = id; }

        UINT Id;
        UINT SubscriptionNumber = 0;
        FString Body;
        FString UnsubMsg;
        TSharedPtr<FJsonObject> Response;
};
```

2 Implementation

To implement, within another section we can create a subscription/ unsubscription function that takes as arguments as **an FString** corresponding to the request to subscribe (which is generated using SubscriptionUtils); as well as the request manager object.

```
void UWalletAccount::Sub2AccountInfo(const FString& pubKey,
↪  UFRequestManager_WB* &SocketManager)
{
        SocketManager->HeartbeatInit();
        FSubscriptionData* SubRequest =
↪  FSubscriptionUtils::AccountSubscribe(pubKey);
        SocketManager->RequestSubscription(SubRequest);
}

void UWalletAccount::UnSub2AccountInfo(const int& ID2Remove,
↪  UFRequestManager_WB*& SocketManager)
{
        FSubscriptionUtils::AccountUnsubscribe(
↪  SocketManager->ActiveSubscriptionsMap[ID2Remove] );
        SocketManager->Unsubscribe(ID2Remove);
}
```

3 Transaction Tracking

To be able to follow the status of a transaction we simply have to implement a subscription to a given signature. This will update us on changes to said transaction with an err attribute that will indicate if there are any errors or not.

The proposed implementation uses a method to subscribe to a transaction, and two methods to get information regarding the error value and context.

```
{
        public:
                static void Sub2Transaction(const FString TransactionSignature,
        UGI_WebSocketManager* &SocketManager);
                static int GetTransactionErr(const int TransactionID,
        UGI_WebSocketManager* &SocketManager);
                static int GetTransactionSlot(const int TransactionID,
        UGI_WebSocketManager* &SocketManager);

};
```