Message System User Guide

Introduction

This guide provides a comprehensive overview of the Message System, detailing its features and usage. The system supports various messaging channels, allowing for robust communication within applications and games.

MessageChannels Enum

The `MessageChannels` enum defines various channels for message communication.

```
public enum MessageChannels
{
    System,
    Logging,
    Debug,
    Error,
    Warning,
    Info,
    Gameplay,
    Level,
    Quest,
    Achievement,
    Combat,
    AI,
    Physics,
    Animation,
    Player,
    PlayerStats,
    PlayerInventory,
    PlayerSkills,
    PlayerActions,
    PlayerHealth,
    PlayerMovement,
    Enemy,
    EnemyStats,
    EnemyAI,
    EnemyHealth,
    EnemyActions,
    Items,
    ItemPickup,
    ItemDrop,
    ItemUse,
    ItemCrafting,
    InventoryManagement,
    UI,
    UINotifications,
    UIMenus,
    UIButtons,
    UIDialogs,
    UILoadingScreen,
    UIPopup,
```

```
    Network,
    NetworkConnect,
    NetworkDisconnect,
    NetworkError,
    NetworkData,
    Audio,
    Music,
    SoundEffects,
    Voice,
    Input,
    KeyPress,
    MouseClick,
    Touch,
    Time,
    DayNightCycle,
    Timer,
    Weather,
    Environment,
    NPC,
    Scripting,
    SaveLoad,
    Cutscene,
    Tutorial,
    Economy,
    Trade,
    Dialogue,
    Camera,
    UI_HUD,
    UI_Inventory,
    UI_QuestLog,
    UI_SkillTree,
    Social,
    Chat,
    Mail,
    FriendRequest,
    Clan,
    Group
}
```

Core Components

Message Envelope

The `IMessageEnvelope` interface and its implementation `MessageEnvelope` are used to wrap messages.

```
public interface IMessageEnvelope
{
    Type MessageType { get; }
}

public interface IMessageEnvelope<out T> : IMessageEnvelope
{
    T? Message { get; }
```

```
}

public class MessageEnvelope<T> : IMessageEnvelope, IMessageEnvelope<T?>
{
    public T? Message { get; private set; }
    public Type MessageType { get; private set; } = typeof(T);

    public MessageEnvelope(T? message)
    {
        Message = message;
    }
}
```

Message Manager

The `MessageManager` class handles the registration, sending, and processing of messages.

Registering Handlers

Register a handler for a specific channel.

```
MessageSystem.MessageManager.RegisterForChannel<GameplayMessage>
(MessageChannels.Gameplay, GameplayMessageHandler);
```

Register a handler for multiple channels.

```
MessageSystem.MessageManager.RegisterForChannel<StringMessage>
(StringMessageHandler, 0, MessageChannels.Gameplay, MessageChannels.System,
MessageChannels.UI);
```

Unregistering Handlers

Unregister a handler for a specific channel.

```
MessageSystem.MessageManager.UnregisterForChannel<GameplayMessage>
(MessageChannels.Gameplay, GameplayMessageHandler);
```

Unregister a handler for multiple channels.

```
MessageSystem.MessageManager.UnregisterForChannel<GameplayMessage>
(StringMessageHandler, MessageChannels.System, MessageChannels.Gameplay,
MessageChannels.UI);
```

Sending Messages

Send a message immediately.

```
MessageSystem.MessageManager.SendImmediate(MessageChannels.Gameplay, new
GameplayMessage("Hello"));
```

Send a message to be processed later.

```
MessageSystem.MessageManager.Send(MessageChannels.Gameplay, new
GameplayMessage("Queued Message"));
```

Send a message immediately to multiple channels.

```
MessageSystem.MessageManager.SendImmediate(new StringMessage("Hello"),
MessageChannels.Gameplay, MessageChannels.System, MessageChannels.UI);
```

Broadcast a message to all channels immediately.

```
MessageSystem.MessageManager.BroadcastImmediate(new GameplayMessage("Broadcast
Message"));
```

Broadcast a message to all channels to be processed later.

```
MessageSystem.MessageManager.Broadcast(new GameplayMessage("Broadcast Message"));
```

Processing Messages

Process queued messages.

```
MessageSystem.MessageManager.ProcessMessages();
```

Process queued messages asynchronously.

```
await MessageSystem.MessageManager.ProcessMessagesAsync();
```

Sending Messages Asynchronously

Send a message immediately asynchronously.

```
await MessageSystem.MessageManager.SendImmediateAsync(MessageChannels.Gameplay,
new GameplayMessage("Hello Async"));
```

Send a message to be processed later asynchronously.

```
await MessageSystem.MessageManager.SendAsync(MessageChannels.Gameplay, new
GameplayMessage("Queued Message Async"));
```

Broadcast a message to all channels immediately asynchronously.

```
await MessageSystem.MessageManager.BroadcastImmediateAsync(new
GameplayMessage("Broadcast Message Immediate Async"));
```

Broadcast a message to all channels to be processed later asynchronously.

```
await MessageSystem.MessageManager.BroadcastAsync(new GameplayMessage("Broadcast
Message Async"));
```

Serialization

Serialize Message to JSON

```
var serializedMessage = MessageSystem.MessageManager.SerializeMessageToJson(new
GameplayMessage("Serialize Test"));
```

Deserialize Message from JSON

```
var deserializedMessage =
MessageSystem.MessageManager.DeserializeMessageFromJson<GameplayMessage>
(serializedMessage);
```

Serialize Message to Binary

```
var serializedMessage = MessageSystem.MessageManager.SerializeMessageToBinary(new
GameplayMessage("Serialize Test"));
```

Deserialize Message from Binary

```
var deserializedMessage =
MessageSystem.MessageManager.DeserializeMessageFromBinary<GameplayMessage>
(serializedMessage);
```

Examples

Sending a Complex Message

Here's an example of sending a complex message with multiple properties.

```
public struct ItemMessage
{
    public IItem Item { get; }
    public IActor? Source { get; }
    public IActor? Target { get; }

    public ItemMessage(IItem item, IActor? source, IActor? target)
    {
        Item = item;
        Source = source;
        Target = target;
    }
}

// Sending an ItemMessage
var item = new Item();
var sourceActor = new Actor();
var targetActor = new Actor();

MessageSystem.MessageManager.SendImmediate(MessageChannels.Items, new
ItemMessage(item, sourceActor, targetActor));
```

Test Class

```
public class TestClass
{
    public string stringField;
    public int intField;
    public string StringProp { get; set; }
}
```

Messaging Tests

Example test cases for the messaging system.

Test Sending Immediate Message

```
[Test]
public void Message_Send_String_To_Gameplay_Immediate()
{
    var text = "Test Taco";
    MessageSystem.MessageManager.SendImmediate(MessageChannels.Gameplay, new
GameplayMessage(text));
    Assert.AreEqual(testObject.stringField, text);
}
```

Test Sending Queued Message

```
[Test]
public void Message_Send_String_To_Gameplay_Queued()
{
    var text = "Test Message";
    MessageSystem.MessageManager.Send(MessageChannels.Gameplay, new
GameplayMessage(text));
    MessageSystem.MessageManager.ProcessMessages();
    Assert.AreEqual(testObject.stringField, text);
}
```

Conclusion

This guide provides an overview of the Message System, detailing its core components, usage, and examples. For more information and advanced usage, refer to the source code and tests.