Dragon Lens Message System User Guide v2.2.0

🌐 Introduction

Welcome to the **Dragon Lens Message System v2.2.0** user guide! This version introduces full **string-based channel support**, which allows developers to send, register, and broadcast messages using simple string identifiers in addition to the existing enum-based and custom channels.

This guide provides a detailed reference for every aspect of the message system, including real-world use cases, handler design, asynchronous processing, serialization, and best practices.

🤖 Core Concepts

IMessageChannel Interface

```
public interface IMessageChannel
{
    string Name { get; }
}
```

Defines a contract for all channels. This makes it easy to abstract between enum, custom, or string-based channels.

DefaultMessageChannel

```
public class DefaultMessageChannel : IMessageChannel
{
    public MessageChannels Channels { get; private set; }
    public string Name => Channels.ToString();
}
```

Encapsulates `MessageChannels` enum values. Best used for predefined, strongly typed channels.

CustomChannel

```
public class CustomChannel : IMessageChannel
{
    public string Name { get; private set; }
    public CustomChannel(string name) => Name = name;
}
```

Allows flexible creation of channels at runtime using plain strings.

🚀 v2.2.0 Feature Spotlight: String Channel Support

You can now use raw strings instead of enums or custom class instances:

```
MessageSystem.MessageManager.RegisterForChannel<MyMessage>("GameplayItems",
MyHandler);
MessageSystem.MessageManager.SendImmediate("GameplayItems", new MyMessage());
```

All registration, sending, and broadcasting methods have overloads for string channels.

📄 Registering Handlers

Using String Channels

```
MessageSystem.MessageManager.RegisterForChannel<MyMessage>("CustomGameplay",
MyMessageHandler);
```

Multiple String Channels

```
MessageSystem.MessageManager.RegisterForChannel<MyMessage>(MyMessageHandler, 0,
"One", "Two", "Three");
```

Using Enums (Strong Typing)

```
MessageSystem.MessageManager.RegisterForChannel<ItemMessage>
(MessageChannels.Items, ItemMessageHandler);
```

CustomChannel (Wrapped String)

```
var myChannel = new CustomChannel("Crafting");
MessageSystem.MessageManager.RegisterForChannel<CraftMessage>(myChannel,
CraftMessageHandler);
```

📢 Sending Messages

Send Immediately

```
MessageSystem.MessageManager.SendImmediate("ItemsChannel", new ItemMessage(...));
```

Send to Enum Channel

```
MessageSystem.MessageManager.SendImmediate(MessageChannels.ItemPickup, new
ItemMessage(...));
```

Send Queued

```
MessageSystem.MessageManager.Send("InventoryQueue", new ItemMessage(...));
```

Send to Multiple Channels

```
MessageSystem.MessageManager.SendImmediate(new ItemMessage(...), "ItemsChannel",
"LogChannel");
```

⏳ Processing Messages

Manual

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

Asynchronous

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

⏱️ Async Messaging

Send Immediately Async

```
await MessageSystem.MessageManager.SendImmediateAsync("AsyncItem", new
ItemMessage(...));
```

Send Queued Async

```
await MessageSystem.MessageManager.SendAsync("AsyncDelayed", new
ItemMessage(...));
```

📢 Broadcasting

Broadcast Immediate

```
MessageSystem.MessageManager.BroadcastImmediate(new ItemMessage(...));
```

Broadcast Queued

```
MessageSystem.MessageManager.Broadcast(new ItemMessage(...));
```

Broadcast Async

```
await MessageSystem.MessageManager.BroadcastImmediateAsync(new ItemMessage(...));
await MessageSystem.MessageManager.BroadcastAsync(new ItemMessage(...));
```

🔧 Real World Example: `ItemMessage`

```
public struct ItemMessage
{
    public IItem Item { get; }
    public IActor? Source { get; }
    public List<IActor?>? Targets { get; }

    public ItemMessage(IItem item, IActor? source, List<IActor?>? targets)
    {
        Item = item;
        Source = source;
        Targets = targets;
    }
}
```

Registering a Handler

```
MessageSystem.MessageManager.RegisterForChannel<ItemMessage>
(MessageChannels.Items, HandleItemMessage);
```

Handling the Message

```
private void HandleItemMessage(IMessageEnvelope envelope)
{
    if (!envelope.Message<ItemMessage>().HasValue) return;
    var data = envelope.Message<ItemMessage>().GetValueOrDefault();

    // Example use
    Console.WriteLine($"Item used: {data.Item.Name} by {data.Source?.Name}");
    data.Targets?.ForEach(target => Console.WriteLine($"Affected Target:
{target?.Name}"));
}
```

🔢 Serialization

```
var serialized = MessageSystem.MessageManager.SerializeMessageToJson(new
ItemMessage(...));
var deserialized =
MessageSystem.MessageManager.DeserializeMessageFromJson<ItemMessage>(serialized);
```

🌟 Summary Table

Feature	Description
String Channels	Dynamic and easy-to-use identifiers
Enum Channels	Strong typing and organization
CustomChannel	Reusable named wrappers for strings
Sync & Async Messaging	Supports queued/immediate with awaitable methods
Multiple Channel Support	Send or register across several channels at once
JSON Serialization	Useful for persistence, debugging

🔮 Best Practice: Message Handling Pattern

When creating a handler, always validate and extract safely:

```
private void AwesomeCustomChannelHandler(IMessageEnvelope message)
{
    if (!message.Message<SystemMessage>().HasValue) return;
    var data = message.Message<SystemMessage>().GetValueOrDefault();
    testObject.intField = data.TestObject.intField;
    testObject.stringField = data.TestObject.stringField;
}
```

This ensures type safety and guards against null values. Always pattern match using `HasValue` and then `.GetValueOrDefault()`.

🎓 Learn More

- Explore the full `MessageManager` API
- Read tests and examples in the repo
- Join the community and contribute enhancements!

Thanks for using Dragon Lens Message System v2.2.0! ✨

Crafted with ❤️‍🔥 to keep your systems modular, clean, and powerful.