Practical ZeroMQ

Elton Stoneman
geekswithblogs.net/eltonstoneman
@EltonStoneman

pluralsight
hardcore dev and IT training

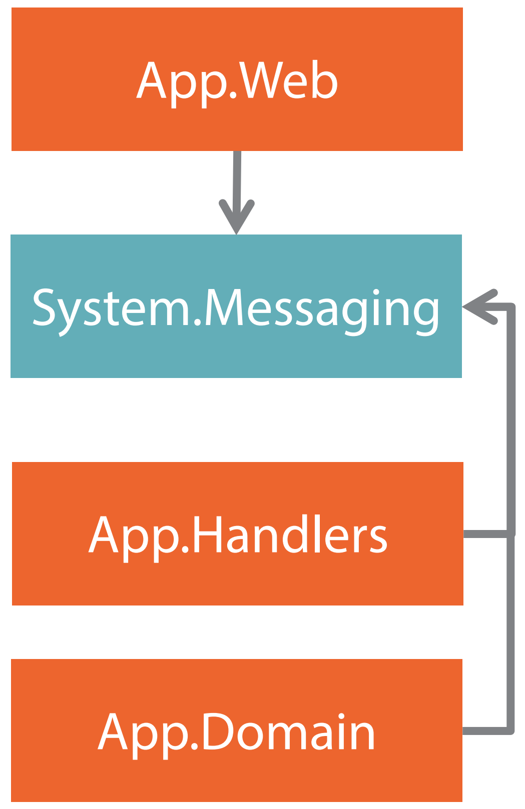
Practical ZeroMQ

Implement **request-response** & **publish-subscribe**

Interactive user requests & **event driven** workflows

Practical considerations for messaging

Dependencies



Messaging decouples components

Multiple dependencies on messaging

Core dependency

MSMQ leaks into app code

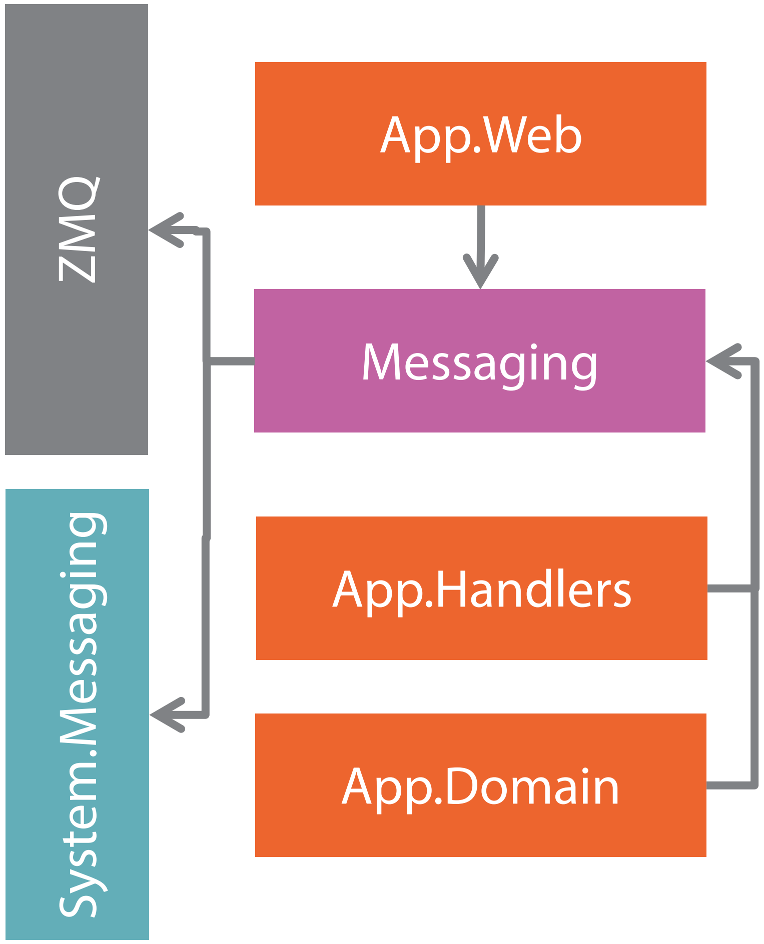
Difficulty replacing MSMQ

Messaging as infrastructure

Vertical layer

Injected component

Abstraction



Abstract messaging layer
App code uses abstraction

Abstraction uses implementation
MSMQ, ZeroMQ, etc.

Decouples messaging implementation
Supports technology swap
Or use of multiple technologies

Abstracted Messaging Layer

Technology-agnostic implementations

Pattern-based client interface

Clean abstraction surface

Demo 1: Messaging Layer

Feature

Abstracted messaging layer

Task

Design & use of IMessageQueue and Message

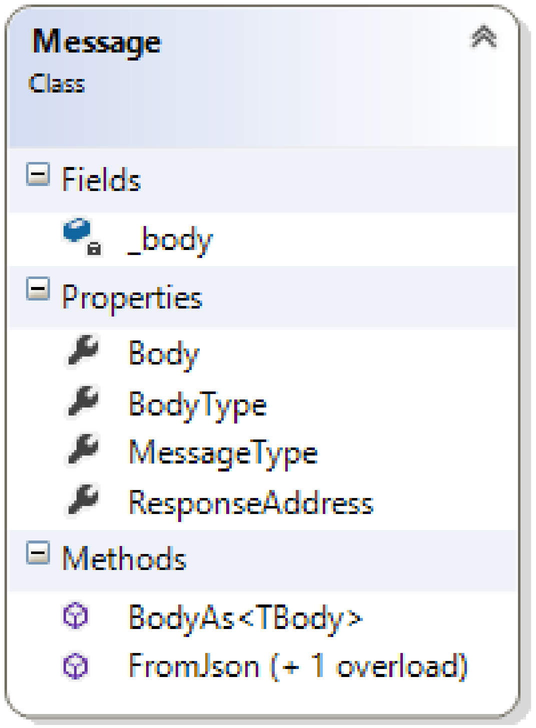
Task

MSMQ implementation of IMessageQueue

Demo 1: Messaging Layer

Demo 1: Messaging Layer

- Message envelope



Message
Class

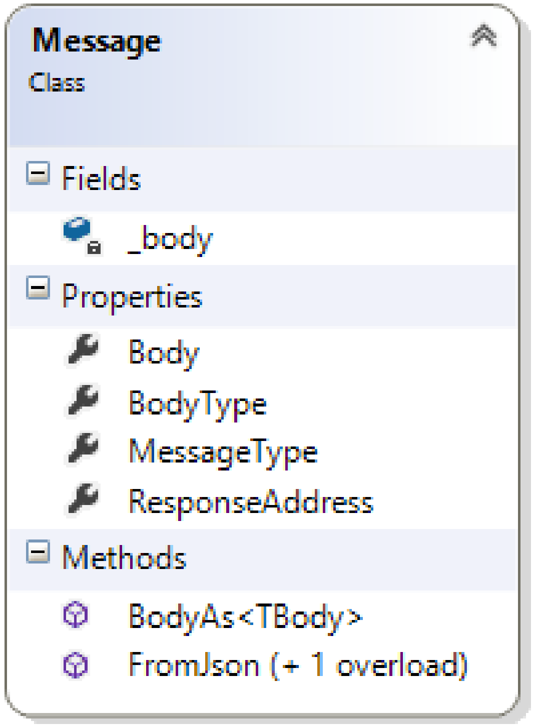
Fields
_body

Properties
Body
BodyType
MessageType
ResponseAddress

Methods
BodyAs<TBody>
FromJson (+ 1 overload)

Demo 1: Messaging Layer

- Message envelope



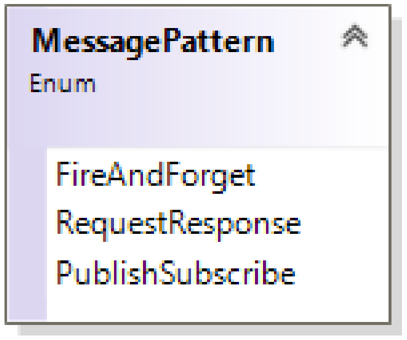
Message
Class

Fields
_body

Properties
Body
BodyType
MessageType
ResponseAddress

Methods
BodyAs<TBody>
FromJson (+ 1 overload)

- Messaging pattern

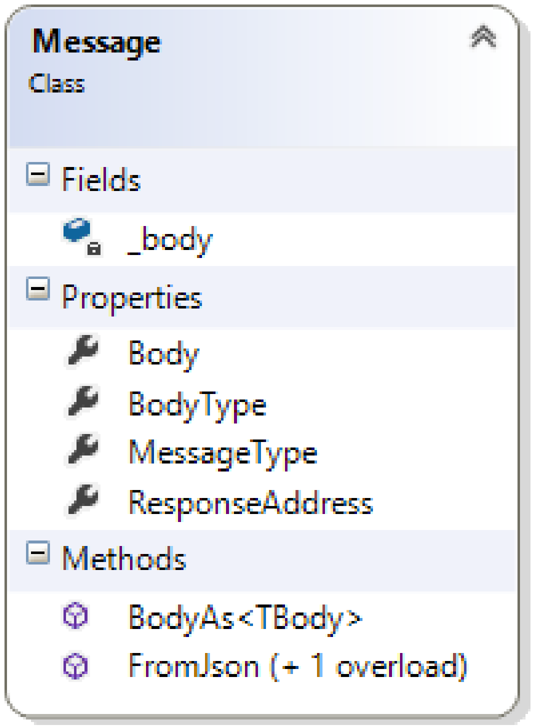


MessagePattern
Enum

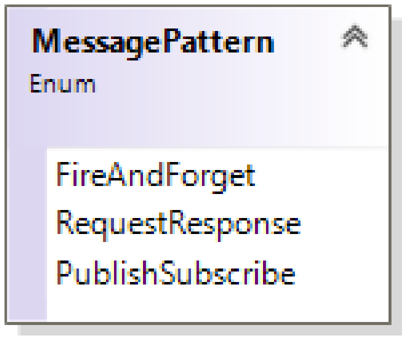
FireAndForget
RequestResponse
PublishSubscribe

Demo 1: Messaging Layer

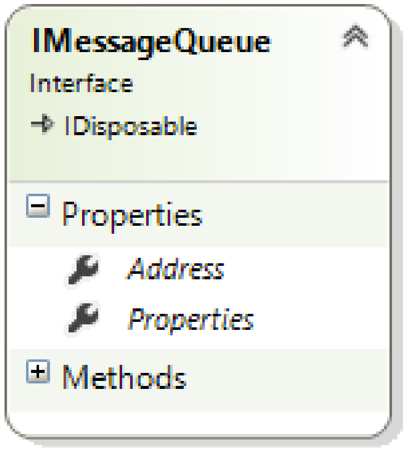
- Message envelope



- Messaging pattern



- Abstract message queue



ZeroMQ

C

ZMQ

Implement IMessageQueue

Using Context and Sockets

Need to keep objects alive

Connected Sockets

Connect, disconnect & reconnect

Not typical usage

MessageQueueFactory

Caches IMessageQueue instances

Reuse for process lifetime

Demo 2: IMessageQueue Implementation

Feature

Implement ZeroMQ messaging layer

Task

Implement IMessageQueue request-response

Task

Add fire-and-forget & publish-subscribe

Demo 2: IMessageQueue Implementation

- **ZeroMqMessageQueue**
  - Inherit from common base
  - Initialise for outbound messaging

```
    EnsureContext();
    Initialise(Direction.Outbound, name, pattern, properties);
    switch (Pattern)
    {
        case MessagePattern.RequestResponse:
            _socket = _Context.Socket(SocketType.REQ);
            _socket.Connect(Address);
            break;
```

Demo 2: IMessageQueue Implementation

- **ZeroMqMessageQueue**
  - Inherit from common base
  - Initialise for outbound messaging

```
        case MessagePattern.RequestResponse:
            _socket = _Context.Socket(SocketType.PUSH);
            _socket.Connect(Address);
            break;

        case MessagePattern.FireAndForget:
            _socket = _Context.Socket(SocketType.PUB);
            _socket.Bind(Address);
            break;
```

Demo 2: IMessageQueue Implementation

- **ZeroMqMessageQueue**
  - Initialise for inbound messaging

```
        case MessagePattern.RequestResponse:
            _socket = _Context.Socket(SocketType.REP);
            _socket.Bind(Address);
            break;

        case MessagePattern.FireAndForget:
            _socket = _Context.Socket(SocketType.PULL);
            _socket.Bind(Address);
            break;

        case MessagePattern.PublishSubscribe:
            _socket = _Context.Socket(SocketType.SUB);
            _socket.Connect(Address);
            _socket.Subscribe("", Encoding.UTF8);
            break;
```

Demo 2: IMessageQueue Implementation

- **ZeroMqMessageQueue**
  - Ensure a single Context instance is shared

```
    private static void EnsureContext()
    {
        if (_Context == null)
        {
            lock (_ContextLock)
            {
                if (_Context == null)
                {
                    _Context = new Context();
```

Demo 2: IMessageQueue Implementation

- **ZeroMqMessageQueue**
  - Look up address based on queue name

```
    switch (name.ToLower())
    {
        case "unsubscribe":
            return "tcp://127.0.0.1:5555";

        case "doesuserexist":
            return "tcp://127.0.0.1:5556";

        case "unsubscribed-event":
            return "pgm://127.0.0.1;239.192.1.1:5557";

        case "unsubscribe-legacy":
            return "pgm://127.0.0.1;239.192.1.1:5557";
            //etc
```

Demo 2: IMessageQueue Implementation

- **ZeroMqMessageQueue**
  - Send message

```
public override void Send(Message message)
{
    var messageJson = message.ToJsonString();
    _socket.Send(messageJson, Encoding.UTF8);
}
```

  - Receive next message

```
public override void Receive(Action<Message> onMessageReceived)
{
    var inbound = _socket.Recv(Encoding.UTF8);
    var message = Message.FromJson(inbound);
    onMessageReceived(message);
}
```

Demo 2: IMessageQueue Implementation

- **ZeroMqMessageQueue**
  - Listen for all messages

```
public override void Listen(Action<Message> onMessageReceived)
{
    while (true)
    {
        Receive(onMessageReceived);
    }
}
```

Demo 2: IMessageQueue Implementation

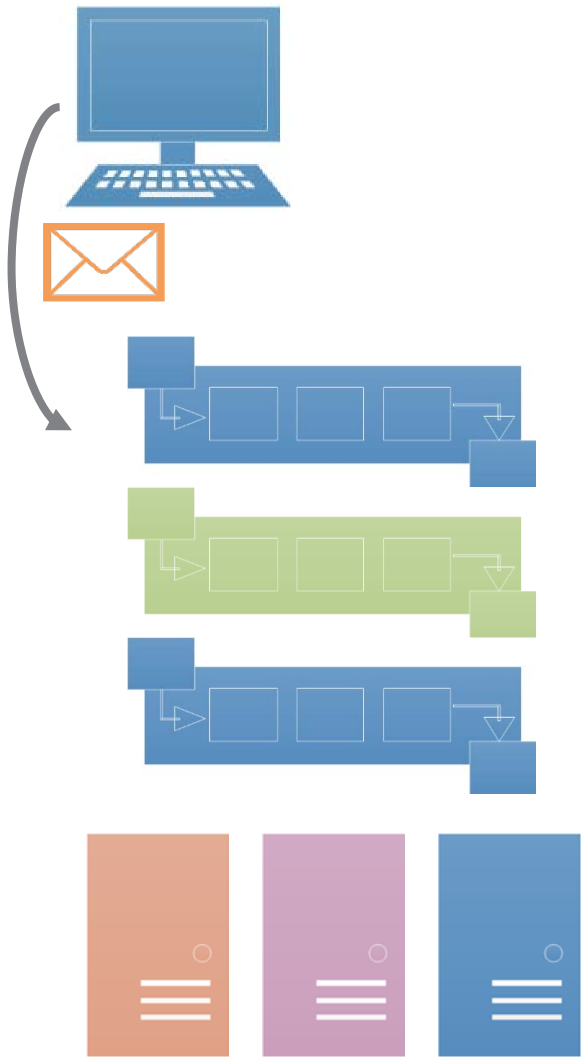
- **ZeroMqMessageQueue**
  - Get response queue to use with request message

```
public override IMessageQueue GetResponseQueue()
{
    return this;
}
```

  - Get reply queue to use for response message

```
public override IMessageQueue GetReplyQueue(Message message)
{
    return this;
}
```

Automated Testing



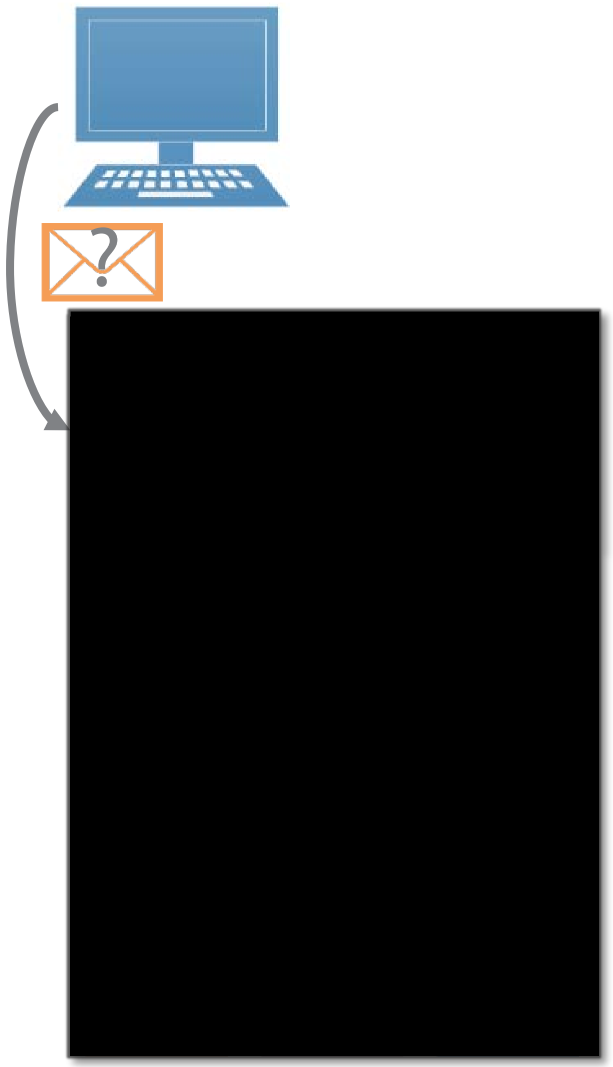
Loosely-coupled components

No internal visibility

Difficult to track progress

- and identify problems

Automated Testing



Loosely-coupled components

No internal visibility

Difficult to track progress

- and identify problems

Automated Testing

Unit testing

Mock<IMessageQueue>

Assert sender behaviour

Assert receiver behaviour

Automated Testing

m

Integration testing

MSMQ/ZeroMQ queues

Assert client behaviour

Assert queue behaviour

Automated Testing

End-to-end testing

Queues & dependencies

Assert sender & receiver behaviour

Assert client & queue behaviour

Assert system behaviour

End-to-End Testing: Considerations

Managing dependencies

Verifying outcomes

Allowing for long-running steps

Language & technology

Demo 3: End-to-End Testing

Feature

Verify all steps in the workflow are performed with an automated test

Task

Walkthrough an end-to-end test using SpecFlow

Task

Verify workflow, swapping between ZeroMQ and MSMQ

Demo 3: End-to-End Testing

Demo 3: End-to-End Testing

- **SpecFlow uses Gherkin**
  - Define system behaviour in English

```
Given the message handlers are running
When a user submits the unsubscribe form with email address xyz
Then the user will receive a Confirmation response
And they should be flagged in the database as unsubscribed within ...
And they should be unsubscribed from the legacy system within 5 seconds
And they should be unsubscribed from CRM within 5 seconds
And they should be unsubscribed from the mail fulfilment system ...
```

Demo 3: End-to-End Testing

- **Process management**
  - Start and stop handlers using [Assembly] attributes

```
[AssemblyInitialize]
public static void Start(TestContext context)
{
    Process.Start(new ProcessStartInfo("StartHandlers.cmd")
                        { WindowStyle = ProcessWindowStyle.Hidden });
    Thread.Sleep(5000);
}

[AssemblyCleanup]
public static void Stop()
{
    Process.Start(new ProcessStartInfo("StopHandlers.cmd")
                        {  WindowStyle=ProcessWindowStyle.Hidden });
}
```

Demo 3: End-to-End Testing

- **Verifying asynchronous functions**
  - Write database events as part of workflow processing
  - Check for events by retrying assertions within a timeout

```
RetryAssert.WithinTimeout(() =>
{
    using (var context = new UserModelContainer())
    {
        return context.UserEvents.Count(x =>
                                x.User.EmailAddress == _emailAddress &&
                                x.EventCode == expectedEventCode &&
                                x.RecordedAt > _testStartedAt) == 1;
               }
           }, 250, 1000 * timeoutSeconds,
                             "Expected event code: {0} recorded after: {1}", 
                                    expectedEventCode, _testStartedAt);
```

Summary

- **Abstract messaging** ✓
  - IMessageQueue
  - MSMQ implementation
- **ZeroMQ implementation** ✓
  - Using Socket and Context
  - Modified MessageQueueFactory
- **End-to-end testing** ✓
  - SpecFlow defines expected behaviour
  - Managing dependencies
  - Verifying outcomes efficiently

Cloud Message Queues (Azure & AWS)