

Implementing the Queueing Code



Stephen Haunts

DEVELOPER, LEADER, AUTHOR AND TRAINER

@stephenhaunts www.stephenhaunts.com



Overview



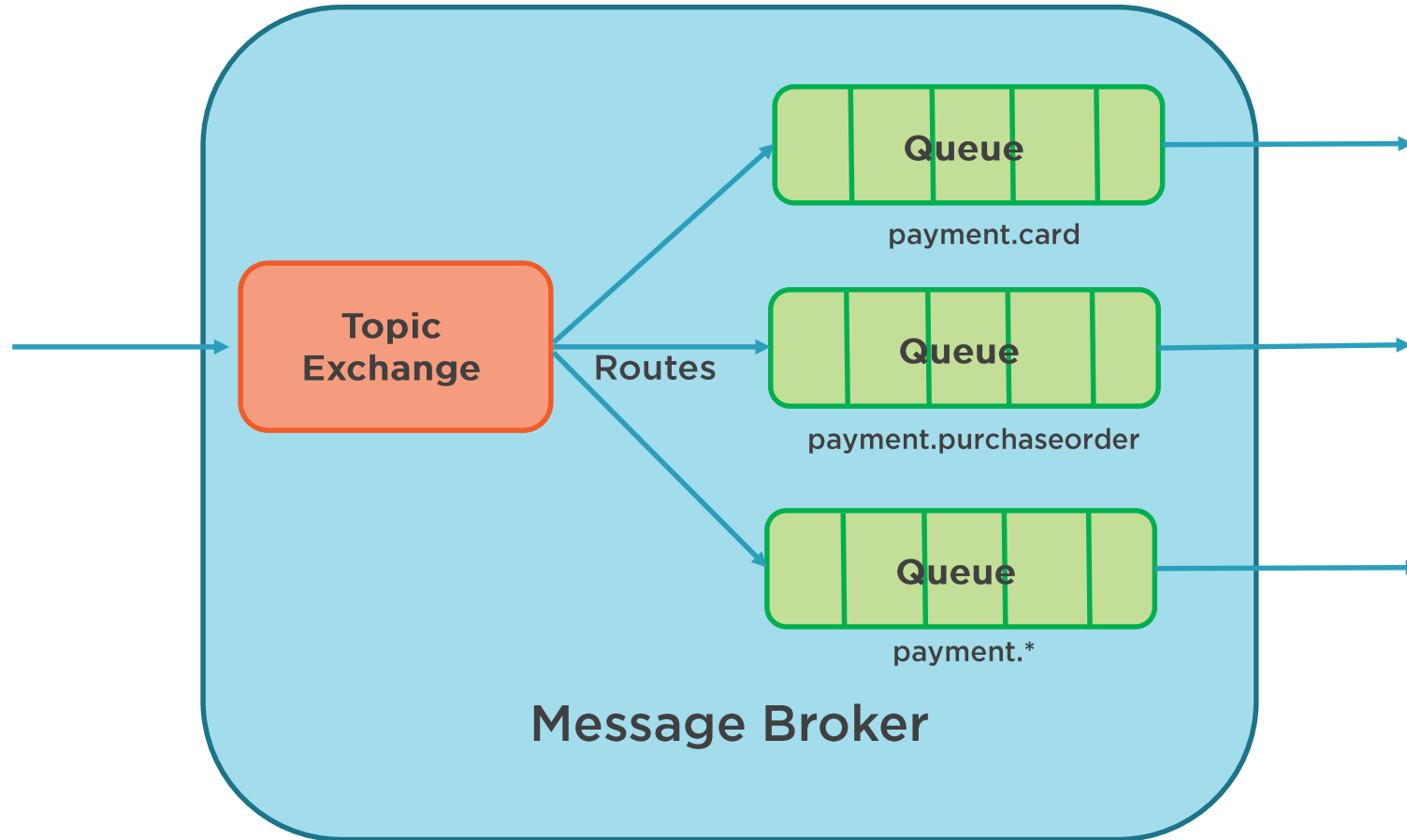
Implement our WebAPI and consumers

Topic based publisher and subscribe

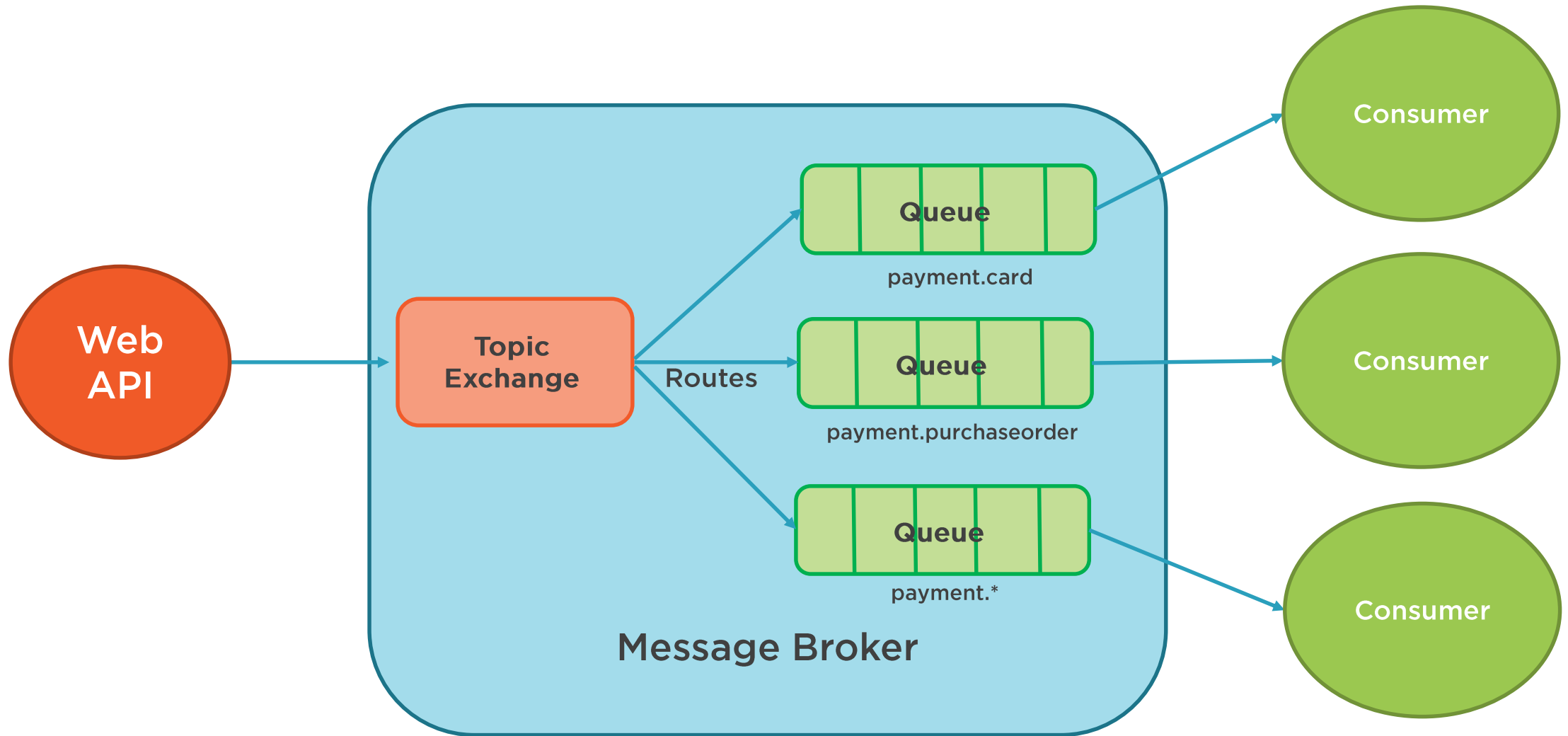
Remote procedure calls














Topic Based Publisher and Subscribe



Topic Based Publisher and Subscribe














Topic Based Publisher and Subscribe

- ▲  **Payments API**
 - ▶  Properties
 - ▶  References
 - ▶  App_Start
 - ▲  Controllers
 - ▲  **DirectCardPaymentController.cs**
 - ▲  DirectCardPaymentController
 - ⊗ MakePayment(CardPayment) : IHttpActionResult
 - ▲  **QueueCardPaymentController.cs**
 - ▲  QueueCardPaymentController
 - ⊗ MakePayment(CardPayment) : IHttpActionResult
 - ▲  **QueuePurchaseOrderController.cs**
 - ▲  QueuePurchaseOrderController
 - ⊗ MakePayment(PurchaseOrder) : IHttpActionResult



Topic Based Publisher and Subscribe

- ▲  **Payments API**
 - ▷  Properties
 - ▷  References
 - ▷  App_Start
 - ▲  Controllers
 - ▲  DirectCardPaymentController.cs
 - ▲  DirectCardPaymentController
 - ⌚ MakePayment(CardPayment) : IHttpActionResult
 - ▲  QueueCardPaymentController.cs
 - ▲  QueueCardPaymentController
 - ⌚ MakePayment(CardPayment) : IHttpActionResult
 - ▲  QueuePurchaseOrderController.cs
 - ▲  QueuePurchaseOrderController
 - ⌚ MakePayment(PurchaseOrder) : IHttpActionResult



Topic Based Publisher and Subscribe

```
try
{
    RabbitMQClient client = new RabbitMQClient();
    client.SendPayment(payment);
    client.Close();
}
catch (Exception)
{
    return StatusCode(HttpStatusCode.BadRequest);
}
return Ok(payment);
```



Topic Based Publisher and Subscribe

```
try
{
    RabbitMQClient client = new RabbitMQClient();
    client.SendPayment(payment);
    client.Close();
}
catch (Exception)
{
    return StatusCode(HttpStatusCode.BadRequest);
}
return Ok(payment);
```



Topic Based Publisher and Subscribe

```
try
{
    RabbitMQClient client = new RabbitMQClient();
    client.SendPayment(payment);
    client.Close();
}
catch (Exception)
{
    return StatusCode(HttpStatusCode.BadRequest);
}
return Ok(payment);
```



Topic Based Publisher and Subscribe

```
try
{
    RabbitMQClient client = new RabbitMQClient();
    client.SendPayment(payment);
    client.Close();
}
catch (Exception)
{
    return StatusCode(HttpStatusCode.BadRequest);
}
return Ok(payment);
```



Topic Based Publisher and Subscribe

```
try
{
    RabbitMQClient client = new RabbitMQClient();
    client.SendPayment(payment);
    client.Close();
}
catch (Exception)
{
    return StatusCode(HttpStatusCode.BadRequest);
}
return Ok(payment);
```



Topic Based Publisher and Subscribe

```
private static ConnectionFactory _factory;
```

```
private static IConnection _connection;
```

```
private static IModel _model;
```

```
private const string ExchangeName = "Topic_Exchange";
```

```
private const string CardPaymentQueueName = "CardPaymentTopic_Queue";
```

```
private const string PurchaseOrderQueueName = "PurchaseOrderTopic_Queue";
```

```
private const string AllQueueName = "AllTopic_Queue";
```



Topic Based Publisher and Subscribe

```
private static ConnectionFactory _factory;
```

```
private static IConnection _connection;
```

```
private static IModel _model;
```

```
private const string ExchangeName = "Topic_Exchange";
```

```
private const string CardPaymentQueueName = "CardPaymentTopic_Queue";
```

```
private const string PurchaseOrderQueueName = "PurchaseOrderTopic_Queue";
```

```
private const string AllQueueName = "AllTopic_Queue";
```



Topic Based Publisher and Subscribe

```
private static void CreateConnection()  
{  
    _factory = new ConnectionFactory { HostName = "localhost",  
                                       UserName = "guest",  
                                       Password = "guest" };  
  
    _connection = _factory.CreateConnection();  
    _model = _connection.CreateModel();  
    _model.ExchangeDeclare(ExchangeName, "topic");  
}
```



Topic Based Publisher and Subscribe

```
_model.QueueDeclare(CardPaymentQueueName, true, false, false, null);
```

```
_model.QueueDeclare(PurchaseOrderQueueName, true, false, false, null);
```

```
_model.QueueDeclare(AllQueueName, true, false, false, null);
```

```
_model.QueueBind(CardPaymentQueueName, ExchangeName, "payment.card");
```

```
_model.QueueBind(PurchaseOrderQueueName, ExchangeName, "payment.purchaseorder");
```

```
_model.QueueBind(AllQueueName, ExchangeName, "payment.*");
```



Topic Based Publisher and Subscribe

```
_model.QueueDeclare(CardPaymentQueueName, true, false, false, null);
```

```
_model.QueueDeclare(PurchaseOrderQueueName, true, false, false, null);
```

```
_model.QueueDeclare(AllQueueName, true, false, false, null);
```

```
_model.QueueBind(CardPaymentQueueName, ExchangeName, "payment.card");
```

```
_model.QueueBind(PurchaseOrderQueueName, ExchangeName, "payment.purchaseorder");
```

```
_model.QueueBind(AllQueueName, ExchangeName, "payment.*");
```



Topic Based Publisher and Subscribe

```
public void SendPayment(CardPayment payment)
```

```
{
```

```
    SendMessage(payment.Serialize(), "payment.card");
```

```
}
```

```
public void SendPurchaseOrder(PurchaseOrder purchaseOrder)
```

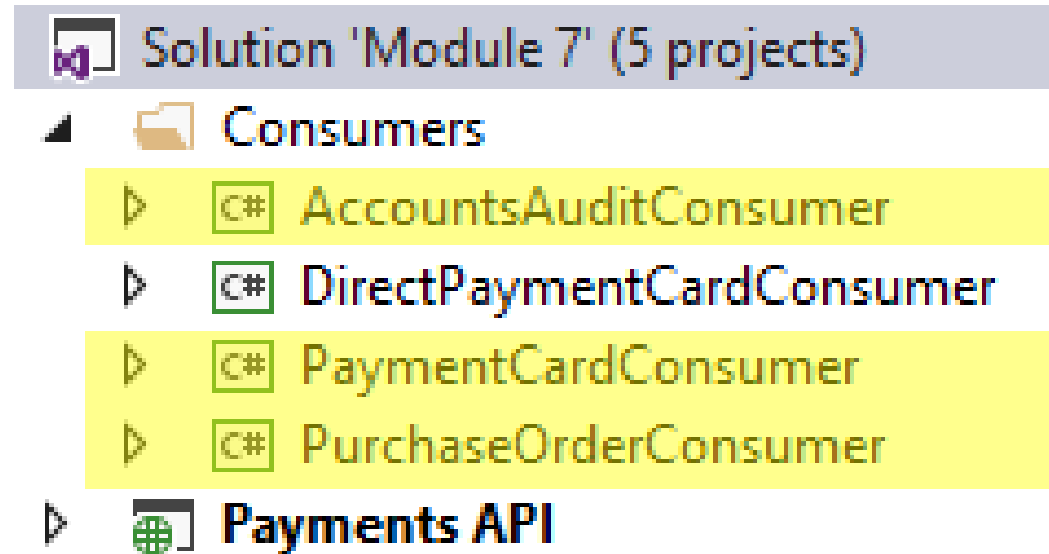
```
{
```

```
    SendMessage(purchaseOrder.Serialize(), "payment.purchaseorder");
```

```
}
```



Topic Based Publisher and Subscribe



Topic Based Publisher and Subscribe

```
RabbitMQConsumer client = new RabbitMQConsumer();  
client.CreateConnection();  
client.ProcessMessages();
```



Topic Based Publisher and Subscribe

```
private static ConnectionFactory _factory;
```

```
private static IConnection _connection;
```

```
private const string ExchangeName = "Topic_Exchange";
```

```
private const string CardPaymentQueueName = "CardPaymentTopic_Queue";
```



Topic Based Publisher and Subscribe

```
public void CreateConnection()  
{  
    _factory = new ConnectionFactory { HostName = "localhost",  
                                       UserName = "guest",  
                                       Password = "guest" };  
}
```



Topic Based Publisher and Subscribe

```
public void ProcessMessages()  
{  
    using (_connection = _factory.CreateConnection())  
    {  
        using (var channel = _connection.CreateModel())  
        {  
            channel.ExchangeDeclare(ExchangeName, "topic");  
            channel.QueueDeclare(CardPaymentQueueName, true, false, false, null);  
            channel.QueueBind(CardPaymentQueueName, ExchangeName,  
                             "payment.cardpayment");  
        }  
    }  
}
```



Topic Based Publisher and Subscribe

```
public void ProcessMessages()  
{  
    using (_connection = _factory.CreateConnection())  
    {  
        using (var channel = _connection.CreateModel())  
        {  
            channel.ExchangeDeclare(ExchangeName, "topic");  
            channel.QueueDeclare(CardPaymentQueueName, true, false, false, null);  
            channel.QueueBind(CardPaymentQueueName, ExchangeName,  
                             "payment.cardpayment");  
        }  
    }  
}
```



Topic Based Publisher and Subscribe

```
channel.BasicQos(0, 1, false);
```

```
Subscription subscription = new Subscription(channel,  
                                              CardPaymentQueueName,  
                                              false);
```



Topic Based Publisher and Subscribe

```
channel.BasicQos(0, 1, false);
```

```
Subscription subscription = new Subscription(channel,  
                                             CardPaymentQueueName,  
                                             false);
```



Topic Based Publisher and Subscribe

```
while (true)
{
    BasicDeliverEventArgs deliveryArguments = subscription.Next();

    var message =
        (CardPayment)deliveryArguments.Body.Deserialize(typeof(CardPayment));

    var routingKey = deliveryArguments.RoutingKey;

    subscription.Ack(deliveryArguments);
}
```



Topic Based Publisher and Subscribe

```
while (true)
{
    BasicDeliverEventArgs deliveryArguments = subscription.Next();

    var message =
        (CardPayment)deliveryArguments.Body.Deserialize(typeof(CardPayment));

    subscription.Ack(deliveryArguments);
}
```



Topic Based Publisher and Subscribe

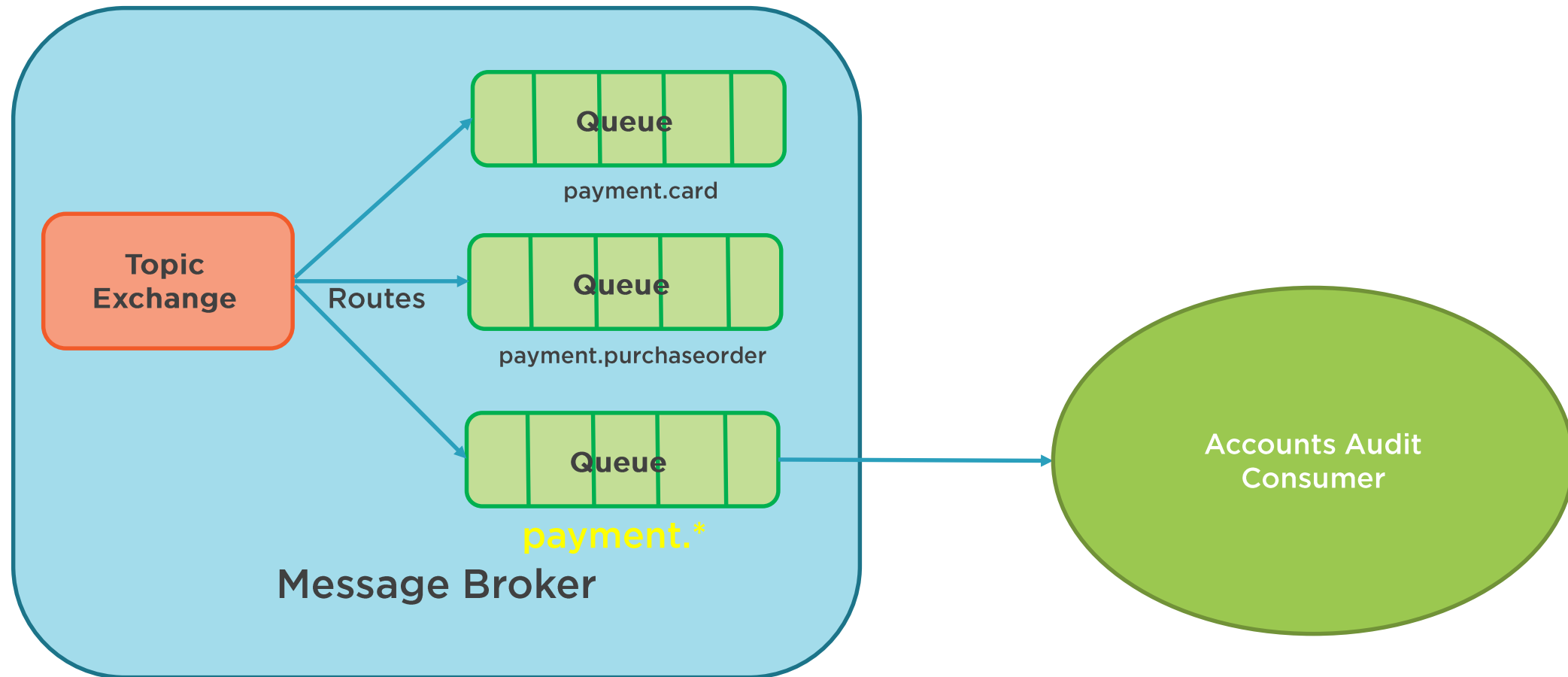
```
while (true)
{
    BasicDeliverEventArgs deliveryArguments = subscription.Next();

    var message =
        (CardPayment)deliveryArguments.Body.Deserialize(typeof(CardPayment));

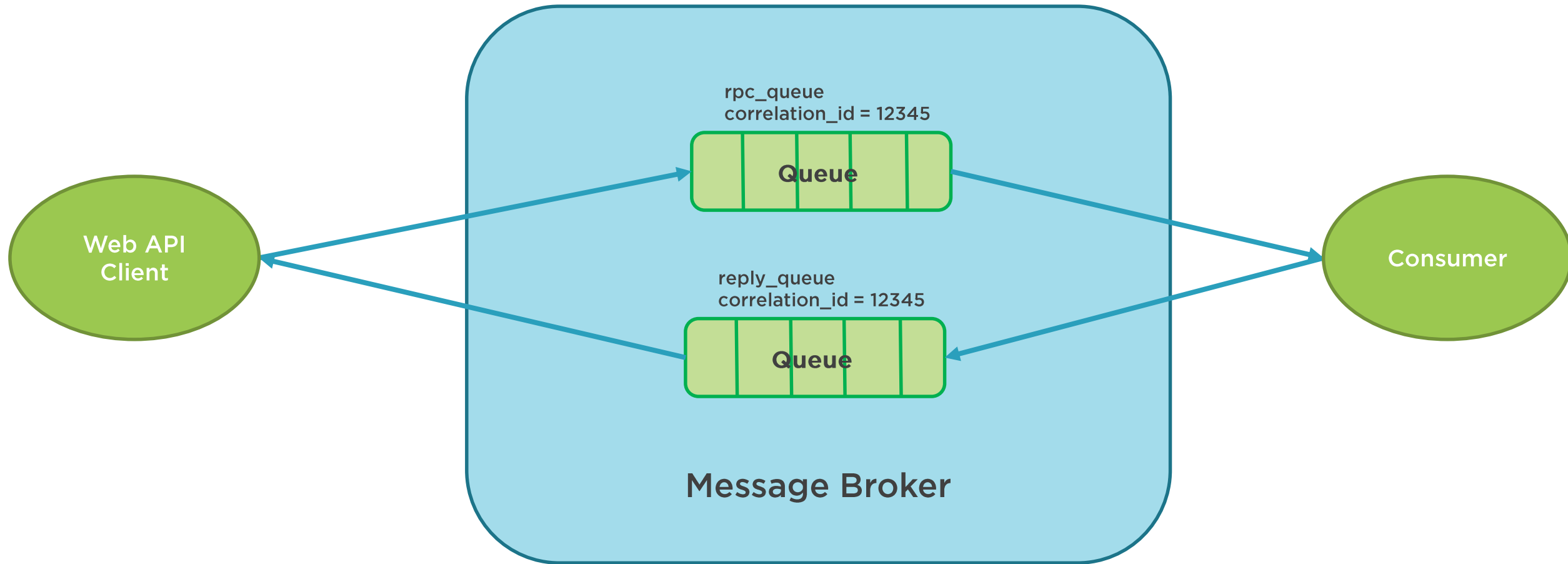
    subscription.Ack(deliveryArguments);
}
```



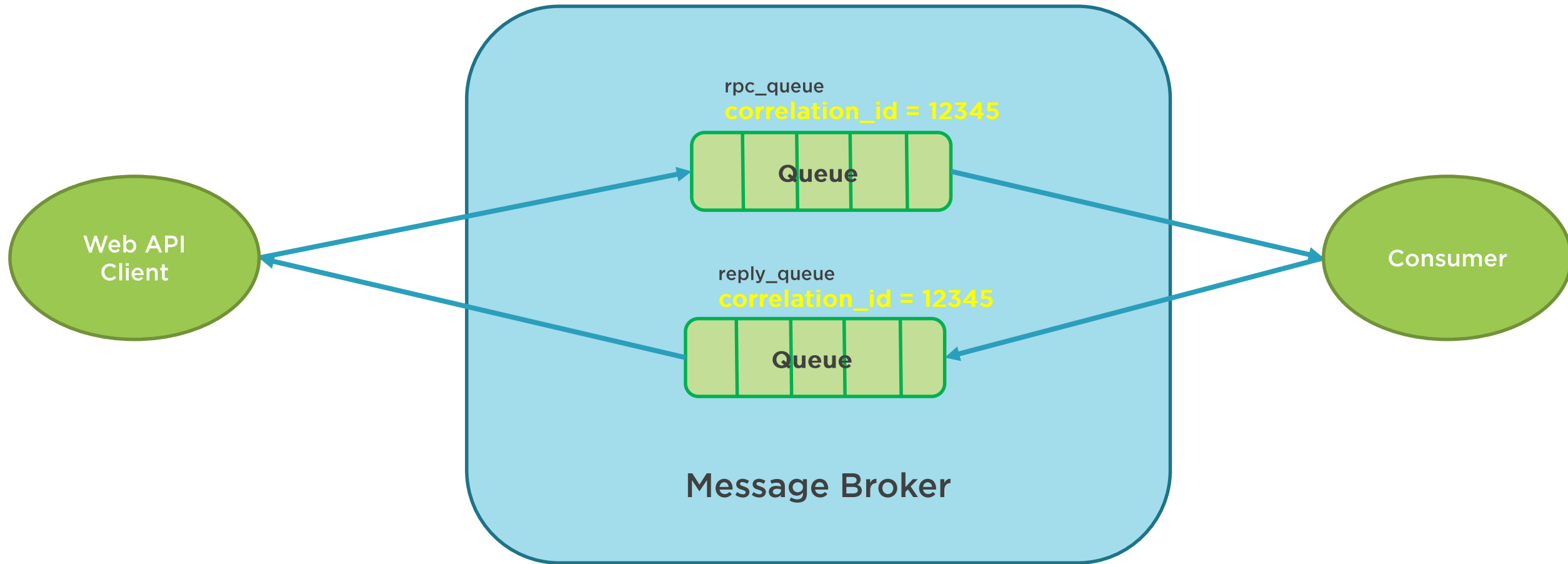
Topic Based Publisher and Subscribe



Remote Procedure Calls



Remote Procedure Calls



Remote Procedure Calls

```
try
{
    RabbitMQDirectClient client = new RabbitMQDirectClient();

    client.CreateConnection();

    reply = client.MakePayment(payment);

    client.Close();
}

catch (Exception)
{
    return StatusCode(HttpStatusCode.BadRequest);
}
```



Remote Procedure Calls

```
public void CreateConnection()  
{  
    var factory = new ConnectionFactory { HostName = "localhost",  
                                           UserName = "guest",  
                                           Password = "guest" };  
  
    _connection = factory.CreateConnection();  
    _channel = _connection.CreateModel();  
}
```



Remote Procedure Calls

```
_replyQueueName = _channel.QueueDeclare("rpc_reply", true, false,  
                                         false, null);
```

```
_consumer = new QueueingBasicConsumer(_channel);
```

```
_channel.BasicConsume(_replyQueueName, true, _consumer);
```

```
}
```



Remote Procedure Calls

```
_replyQueueName = _channel.QueueDeclare("rpc_reply", true, false,  
                                         false, null);
```

```
_consumer = new QueueingBasicConsumer(_channel);
```

```
_channel.BasicConsume(_replyQueueName, true, _consumer);
```

```
}
```



Remote Procedure Calls

```
public string MakePayment(CardPayment payment)
{
    var props = _channel.CreateBasicProperties();
    props.ReplyTo = _replyQueueName;
    props.CorrelationId = Guid.NewGuid().ToString();

    _channel.BasicPublish("", "rpc_queue", props, payment.Serialize());
}
```



Remote Procedure Calls

```
public string MakePayment(CardPayment payment)
{
    var props = _channel.CreateBasicProperties();
    props.ReplyTo = _replyQueueName;
    props.CorrelationId = Guid.NewGuid().ToString();

    _channel.BasicPublish("", "rpc_queue", props, payment.Serialize());
}
```



Remote Procedure Calls

```
while (true)
{
    var ea = _consumer.Queue.Dequeue();

    if (ea.BasicProperties.CorrelationId != corrId) continue;

    var authCode = Encoding.UTF8.GetString(ea.Body);

    return authCode;
}
```



Remote Procedure Calls

```
while (true)
{
    var ea = _consumer.Queue.Dequeue();

    if (ea.BasicProperties.CorrelationId != corrId) continue;

    var authCode = Encoding.UTF8.GetString(ea.Body);

    return authCode;
}
```



Remote Procedure Calls

The screenshot displays the Postman application interface. At the top, the 'Builder' tab is active. The URL bar shows 'http://localhost:53195/api/'. Below the URL bar, the request method is set to 'POST' and the endpoint is 'http://localhost:53195/api/DirectCardPayment'. The 'Body' tab is selected, and the request body is formatted as 'JSON (application/json)' with the following content:

```
{
  "Amount": 15.5,
  "CardNumber": "1234123412341234",
  "Name": "Mr S Haunts"
}
```

The response section at the bottom shows a status of '200 OK' and a time of '28 ms'. The response body is displayed in 'Pretty' format as a single JSON array element:

```
[
  "85729888"
]
```



Remote Procedure Calls

The screenshot displays a REST client interface with a top navigation bar containing icons for Runner, Import, Builder, and Team Library, along with status indicators for IN SYNC and a notification bell. The main interface shows a POST request to `http://localhost:53195/api/DirectCardPayment` with a body of JSON data: `{ "Amount": 15.5, "CardNumber": "1234123412341234", "Name": "Mr S Haunts" }`. The response section shows a status of `200 OK` and a time of `28 ms`. The response body is displayed in a table with one row containing the value `"85729888"`, which is highlighted by a red rectangle.

Runner Import Builder Team Library IN SYNC

`http://localhost:53195/api/` No environment

POST `http://localhost:53195/api/DirectCardPayment` Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Generate Code

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 {
2   "Amount": 15.5,
3   "CardNumber": "1234123412341234",
4   "Name": "Mr S Haunts"
5 }
```

Body Cookies Headers (6) Tests Status: 200 OK Time: 28 ms

Pretty Raw Preview JSON

1	"85729888"
---	------------



Remote Procedure Calls

```
_channel.QueueDeclare("rpc_queue", false, false, false, null);
```

```
_channel.BasicQos(0, 1, false);
```

```
_consumer = new QueueingBasicConsumer(_channel);
```

```
_channel.BasicConsume("rpc_queue", false, _consumer);
```

```
_rnd = new Random();
```



Remote Procedure Calls

```
_channel.QueueDeclare("rpc_queue", false, false, false, null);
```

```
_channel.BasicQos(0, 1, false);
```

```
_consumer = new QueueingBasicConsumer(_channel);
```

```
_channel.BasicConsume("rpc_queue", false, _consumer);
```

```
_rnd = new Random();
```



Remote Procedure Calls

```
var ea = _consumer.Queue.Dequeue();
```

```
var props = ea.BasicProperties;
```

```
var replyProps = _channel.CreateBasicProperties();
```

```
replyProps.CorrelationId = props.CorrelationId;
```



Remote Procedure Calls

```
var ea = _consumer.Queue.Dequeue();
```

```
var props = ea.BasicProperties;
```

```
var replyProps = _channel.CreateBasicProperties();
```

```
replyProps.CorrelationId = props.CorrelationId;
```



Remote Procedure Calls

```
try{  
    response = MakePayment(ea);  
}  
finally {  
    if (response != null)  
    {  
        var responseBytes = Encoding.UTF8.GetBytes(response);  
        _channel.BasicPublish("", props.ReplyTo, replyProps, responseBytes);  
    }  
    _channel.BasicAck(ea.DeliveryTag, false);  
}
```



Remote Procedure Calls

```
try{  
    response = MakePayment(ea);  
}  
finally {  
    if (response != null)  
    {  
        var responseBytes = Encoding.UTF8.GetBytes(response);  
        _channel.BasicPublish("", props.ReplyTo, replyProps, responseBytes);  
    }  
    _channel.BasicAck(ea.DeliveryTag, false);  
}
```



Remote Procedure Calls

```
try{  
    response = MakePayment(ea);  
}  
finally {  
    if (response != null)  
    {  
        var responseBytes = Encoding.UTF8.GetBytes(response);  
        _channel.BasicPublish("", props.ReplyTo, replyProps, responseBytes);  
    }  
    _channel.BasicAck(ea.DeliveryTag, false);  
}
```



Remote Procedure Calls

```
C:\Users\Stephen haunts\Dropbox (Personal)\Pluralsight\RabbitMQ By Example\r...
-----
Payment - 1234123412341234 : £15.5 : Auth Code <40992432>
Correlation ID = 0eb24672-9b41-4895-97ee-c9221600e685
-----

Payment - 1234123412341234 : £15.5 : Auth Code <91353014>
Correlation ID = 6f7d3adf-b585-4e3f-9b93-8b1b804253be
-----

Payment - 1234123412341234 : £15.5 : Auth Code <44855265>
Correlation ID = 6d13716f-3960-4c80-9587-2ce5260f9e64
-----

Payment - 1234123412341234 : £15.5 : Auth Code <56519483>
Correlation ID = 172118d6-9eda-40b6-b332-76d1efb8117f
-----

Payment - 1234123412341234 : £15.5 : Auth Code <33898864>
Correlation ID = aac4b607-6b40-49f3-b2ab-e3445dc6d1
-----

Payment - 1234123412341234 : £15.5 : Auth Code <99809055>
Correlation ID = 121a7a04-23ee-4097-a84d-8a2612d22500
-----

Payment - 1234123412341234 : £15.5 : Auth Code <93937511>
Correlation ID = e7dc036a-9eff-4951-9ddf-79863ba357cd
-----

Payment - 1234123412341234 : £15.5 : Auth Code <85729888>
Correlation ID = 11a22b20-7228-489c-8fca-7b666edd696f
-----
```

