

Creating and Sending Messages



Alan Smith

ACTIVE SOLUTION

@alansmith www.cloudcasts.net

Overview



Working with Messages

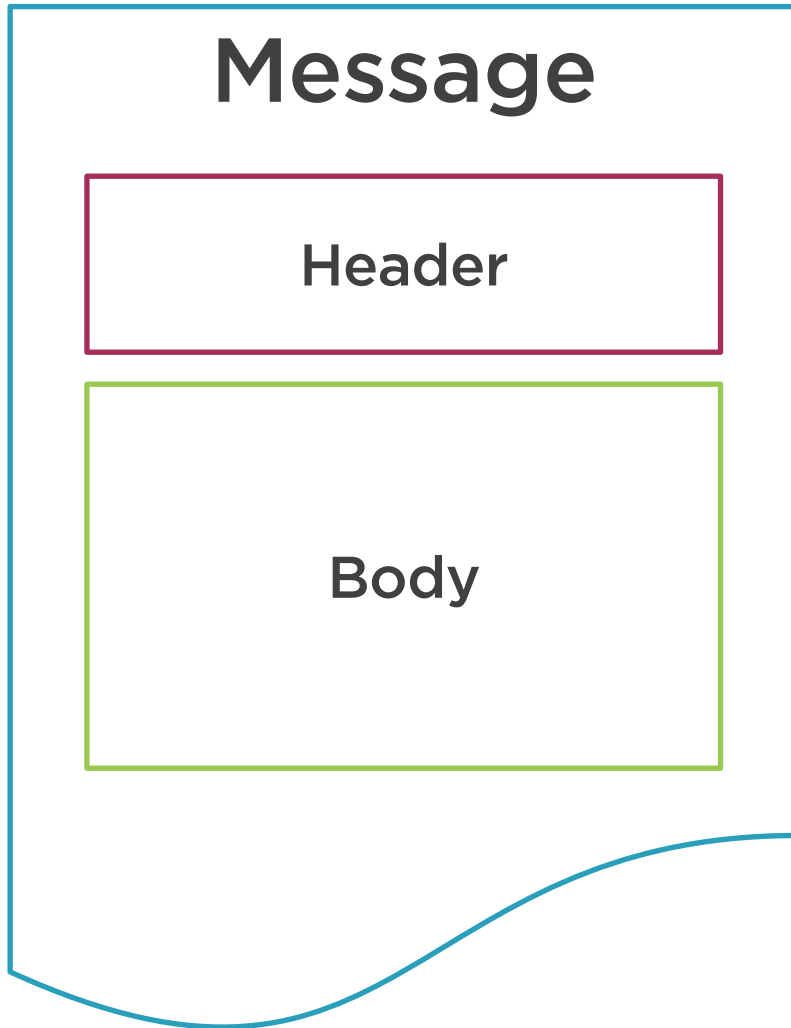
Message Serialization

Creating and Sending Messages

Demo: Creating and Sending Messages

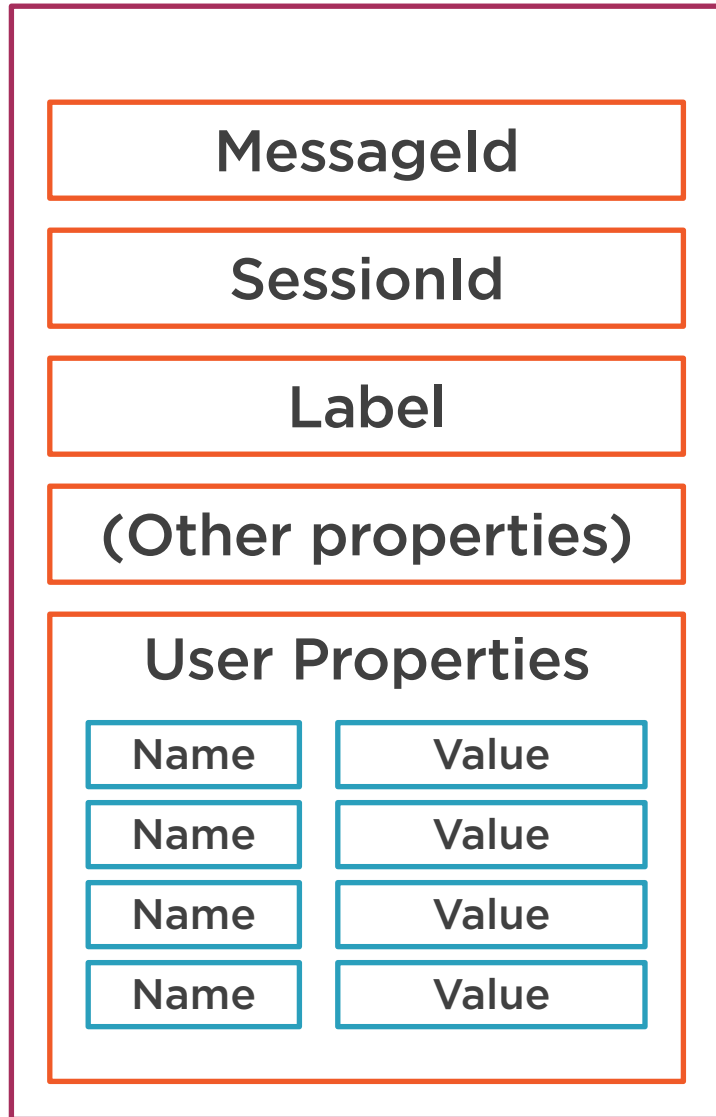
Working with Messages

Messages



- **Message Header**
 - Message context
 - Message properties
- **Message Body**
 - Message content
 - Byte array
 - Serialized object

Message Header



- **Other Properties**
 - ContentType
 - CorrelationId
 - SequenceNumber
 - ReplyTo
 - ReplyToSessionId
 - ScheduledEnqueueTimeUtc
 - TimeToLive

Message Properties – Set by Service

Property	Usage
SequenceNumber	Incrementing number assigned by queue or topic as message is enqueued.
DeliveryCount	The number of times the message has been received by a receiving application.
EnqueuedTimeUtc	The UTC date and time that the message was enqueued.
ExpiresAtUtc	The UTC date and time at which the message is set to expire.
LockedUntilUtc	The UTC date and time at which a message will be released from a lock when received using the peak-lock receive mode.
LockToken	The token assigned by the service bus to a locked message.
Size	The size of the message in bytes.
State	The state of the message.

Message Properties – Set by User

Property	Usage
MessageId	Unique identifier for message, used for duplicate detection.
ContentType	Identifies the content type of the message body.
CorrelationId	Identifier to be used in correlation filter expressions.
Label	Information about the message for receiving application.
SessionId	Unique Session ID should be set for all messages in a session.
TimeToLive	Time message will live on queue before expiring.
ScheduledEnqueueTimeUtc	Message will be stored in service bus, then enqueued at specified time.
To	Specifies recipient of message.
ReplyTo	Reply details to use when sending response in request/response operations.
ReplyToSessionId	Session ID to use when sending response in request/response operations.
UserProperties	String/object dictionary, used for filter expressions and actions.

Message Size Limitations

Message size is currently limited to 256 KB in Basic and Standard tier

- 1024 KB in premium tier

Header size is currently limited to 64 KB

Message Serialization


Message Bodies

Message.Body Property

Namespace: [Microsoft.Azure.ServiceBus](#)

Assembly: Microsoft.Azure.ServiceBus.dll

C#

 Copy

```
public byte[] Body { get; set; }
```

Property Value

[Byte\[\]](#)

Applies to

Azure SDK for .NET

Latest

Text Messages

Serializing a Text Message

```
// Create a text message
var messageText = "Hello, world!";
var message = new Message(Encoding.UTF8.GetBytes(messageText));
```

Deserializing a Text Message

```
// Deserialize a text message
var messageText = Encoding.UTF8.GetText(message.Body);
```

Using the Label Property

```
// Create a message
var message = new Message() { Label = "Hello, world!" };

// Retrieve the message text
var messageText = message.Label;
```

JSON Message Serialization

- **JSON – JavaScript Object Notation**
- **Provides cross-platform message serialization options**
- **JSON strings can be serialized to and from message bodies**
- **More efficient than XML serialization**
- **Allows looser coupling between applications**


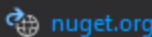
JSON Message Serialization

```
class PizzaOrder
{
    public string CustomerName { get; set; }
    public string Type { get; set; }
    public string Size { get; set; }
}
```



```
{
  "CustomerName": "Alan",
  "Type": "Kebab",
  "Size": "Extra Large"
}
```

Newtonsoft.Json

 **Newtonsoft.Json** 

Installed: 10.0.3

Version: Latest stable 12.0.3

Uninstall

Update

Options

Description

Json.NET is a popular high-performance JSON framework for .NET

Version: 12.0.3

Author(s): James Newton-King

License: [MIT](#)

Date published: Saturday, November 9, 2019 (11/9/2019)

Project URL: <https://www.newtonsoft.com/json>

Report Abuse: <https://www.nuget.org/packages/Newtonsoft.Json/12.0.3/ReportAbuse>

Tags: json

Serializing JSON Messages

```
// Create a new pizza order
var order = new PizzaOrder()
{
    CustomerName = "Alan", Type = "Kebab", Size = "Extra Large"
};

// Serialize the order object
string jsonPizzaOrder = JsonConvert.SerializeObject(order);

// Create a Message
var message = new Message(Encoding.UTF8.GetBytes(jsonPizzaOrder))
{
    Label = "PizzaOrder",
    ContentType = "application/json"
};

// Send the message...
var client = new QueueClient(Settings.ConnectionString, Settings.QueueName);
await client.SendAsync(message);
```

Creating Event and Control Messages

```
// Construct a message with no body content.  
Message eventMsg = new Message ();  
  
// Set the appropriate message properties.  
eventMsg.ContentType = "Event";  
eventMsg.CorrelationId = "OD1345";  
eventMsg.Label = "Processed";
```

- Messages have no body
- Information is conveyed in message header

Sending Messages

Useful Classes

Class	Usage
QueueClient	Send and receive messages from a service bus queue
TopicClient	Send messages to a service bus topic
MessageSender	Send messages to a service bus queue or topic

IMessageSender

Property	Usage
Path	The path to the service bus messaging entity
OperationTimeout	The maximum time for messaging operations to complete

Method	Usage
SendAsync(Message)	Sends a single message
SendAsync(IList<Message>)	Sends a batch of messages
ScheduleMessageAsync	Schedules a message to be enqueued at a later time
CancelScheduledMessageAsync	Cancels a previously scheduled message
CloseAsync	Closes the connection to the service bus entity

Sending Messages

```
// Create a text message
var messageText = "Hello, world!";
var message = new Message(Encoding.UTF8.GetBytes(messageText));

// Create a queue client
var client = new QueueClient(connectionString, queueName);

// Send the message
await client.SendAsync(message);

// Always close the client!
await client.CloseAsync();
```

Sending Message Batches

```
// Create a queue client
var client = new QueueClient(connectionString, queueName);

// Create a message list
var messageList = new List<Message>();

// Create batch of messages
foreach (var pizzaOrder in pizzaOrderList)
{
    var jsonPizzaOrder = JsonConvert.SerializeObject(pizzaOrder);
    var message = new Message(Encoding.UTF8.GetBytes(jsonPizzaOrder));
    messageList.Add(message);
}

// Send the message batch
await client.SendAsync(messageList);

// Always close the client!
await client.CloseAsync();
```

Demo



Creating and Sending Messages

- Creating string messages
- Sending messages
- Serializing objects as messages
- Sending control messages
- Sending message batches

Summary



The Azure Service Bus SDK uses the Message class to represent messages

Messages are comprised of two parts

- Header or context
- Body or content

Using JSON serialization can provide a more cross-platform and loosely coupled solution