MSMQ

MICROSOFT MESSAGE QUEUING

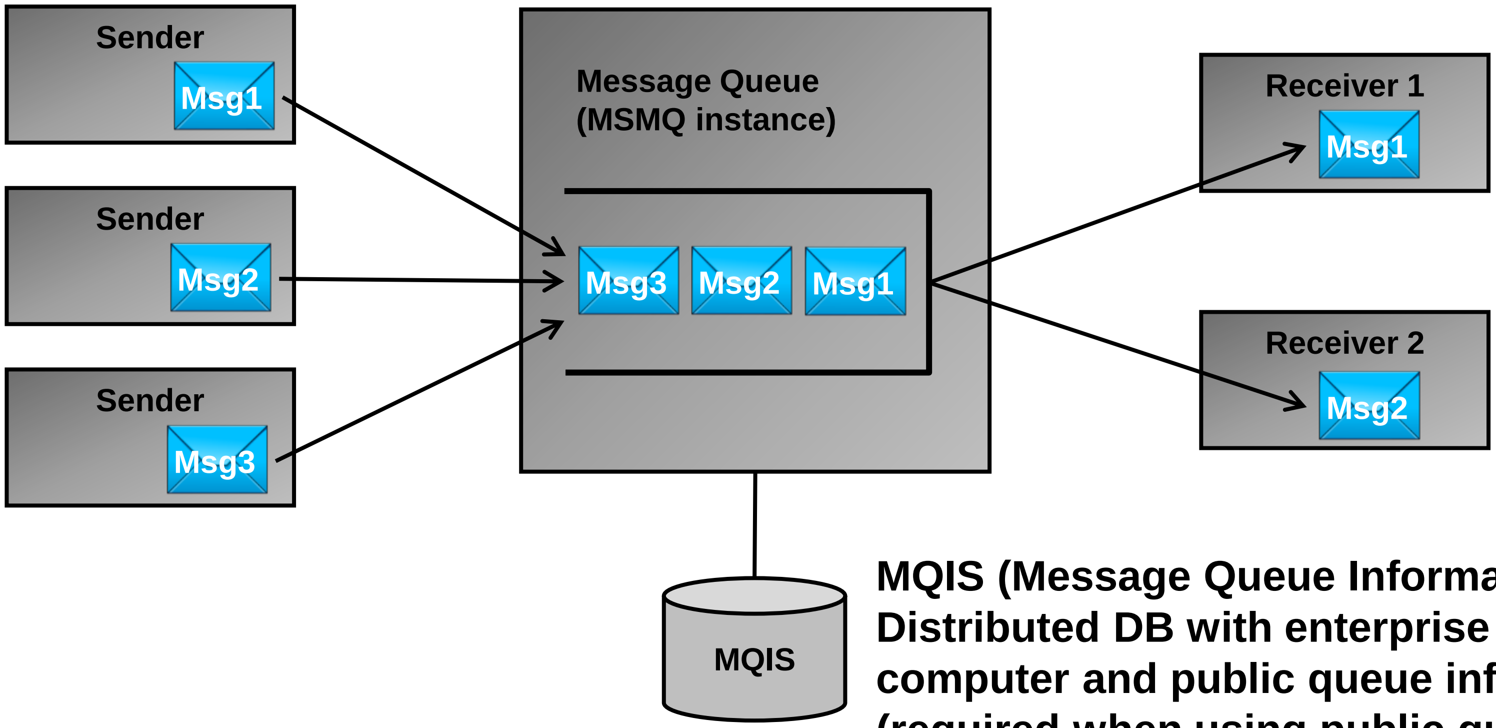
AN INTRODUCTION TO MSMQ, MICROSOFTS MESSAGE QUEUING TECHNOLOGY

Peter R. Egli
INDIGOO.COM

Contents

1. What is MSMQ?

MSMQ is Microsofts message queue system.
MSMQ is a MOM (Message Oriented Middleware) technology for distributed applications.
The messaging allows temporal decoupling of the participants.



MQIS (Message Queue Information Store):
Distributed DB with enterprise topology, computer and public queue information (required when using public queues).

2. Main features of MSMQ

Guaranteed message delivery:
MSMQ guarantees delivery of a message even when the receiver of the message is temporarily down.

Message routing:
MSMQ takes care of routing messages to the addressed destination.

Security:
Optionally, MSMQ messaging supports authentication and encryption of messages.

Priority-based messaging:
MSMQ allows giving messages priorities to expedite the delivery of higher priority messages over lower priority messages.

Different message transport protocols:
MSMQ can be configured to use either TCP or UDP transport protocols directly or use RPC (Remote Procedure Call) as transport protocol.

Transactions:
MSMQ allows packing multiple messages into a transaction. Either all messages pertaining to the transaction are delivered to the destination or none.

MSMQ – Microsoft Message Queuing

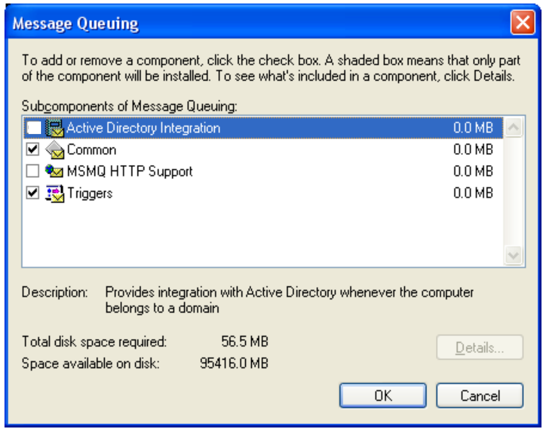
3. MSMQ modes

MSMQ can be run in the 2 modes 'workgroup' and 'domain'.

Workgroup mode:

MSMQ only supports private queues in workgroup mode.

**Installation components for workgroup mode:
Common (MSMQ base functionality).**



Message Queuing

To add or remove a component, click the check box. A shaded box means that only part of the component will be installed. To see what's included in a component, click Details.

Subcomponents of Message Queuing:

☐	Active Directory Integration	0.0 MB
✓	Common	0.0 MB
☐	MSMQ HTTP Support	0.0 MB
✓	Triggers	0.0 MB

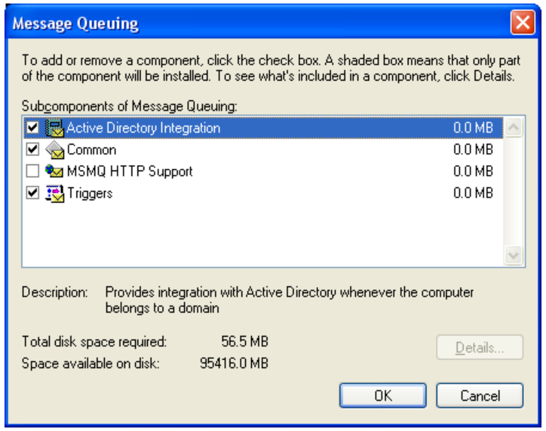
Description: Provides integration with Active Directory whenever the computer belongs to a domain

Total disk space required: 56.5 MB
Space available on disk: 95416.0 MB

Domain mode:

**MSMQ in domain mode allows sending messages between multiple domains (multiple message hops).
In domain mode, MSMQ supports both private and public queues.**

**Installation components for domain mode:
Active Directory Integration, Common, Triggers.**



Message Queuing

To add or remove a component, click the check box. A shaded box means that only part of the component will be installed. To see what's included in a component, click Details.

Subcomponents of Message Queuing:

✓	Active Directory Integration	0.0 MB
✓	Common	0.0 MB
☐	MSMQ HTTP Support	0.0 MB
✓	Triggers	0.0 MB

Description: Provides integration with Active Directory whenever the computer belongs to a domain

Total disk space required: 56.5 MB
Space available on disk: 95416.0 MB

4. MSMQ queue types (1/3)

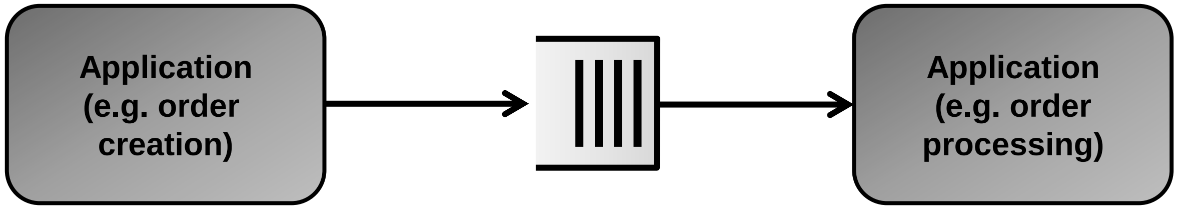
MSMQ defines user message queues and system message queues. User message queues are either private or public queues.

1. Private queues:
Arbitrary applications can read and write to a specific private message queue.

MSMQ does not support message routing with private queues.

The MSMQ service (the message queue actually) needs to be installed on one machine. Different applications may send to or receive from that MSMQ instance.

Typical applications for private queues:
Client and server are relatively close and there is a reliable connection between client and server (e.g. both client and server are hosted on the same machine).


Application (e.g. order creation)
Application (e.g. order processing)

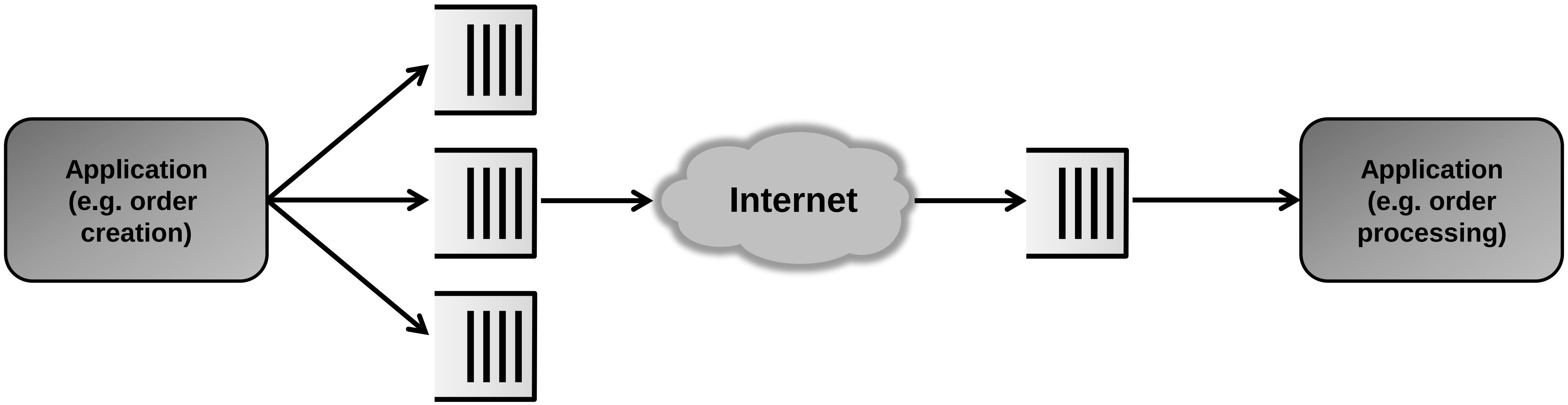
4. MSMQ queue types (2/3)

2. Public queues:

Public queues allow for multi-hop scenarios where messages are replicated through the Active Directory service.

Public queues are published in and hosted by MQIS (MSMQ Information Store = distributed database for MSMQ).

Typical applications: Coupling 2 applications over an unreliable network such as the Internet.


Application (e.g. order creation)
Internet
Application (e.g. order processing)

4. MSMQ queue types (3/3)

3. System queues:

System queues are used for management purposes.

a. Journal message queue:

Journal message queues store copies of messages sent to, through or from a host if the property `UseJournalQueue` is set to true on the message but to false on the receiving queue. This allows getting a message copy in the receiving queue or globally get copies of all messages of all queues in the journal message system queue.
Usage of journal messages: Logging / backup of messages.
N.B.: Regularly purge journal message queues (otherwise they will grow beyond all bounds).

b. Dead-letter message queues:

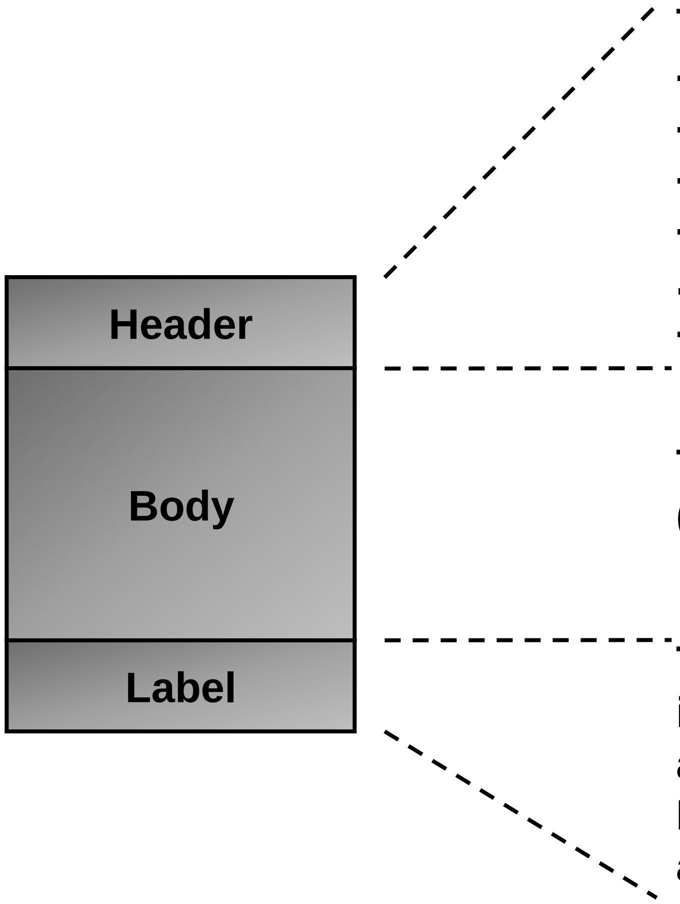
Dead-letter message queues store undeliverable messages or messages that expire before they are received (final resting place for undeliverable messages).

c. Transactional dead-letter message queues:

Transactional dead-letter queues behave as dead-letter messages, but are used for transactional messages.

5. Structure of an MSMQ message

A message contains a range of properties that are carried from the sending MSMQ system to the receiving message queue.



The header carries various properties:
- Message priority
- Message lifetime (how long is message valid)
- Use dead-letter queue flag
- Acknowledge type
...
- UseTracing flag

The body contains the formatted user message (XML, binary or ActiveX format).

The label carries additional application specific information (carried transparently from the sending application to receiving application).
E.g. the label could be used to carry an application-specific priority (urgent processing, normal).

6. Triggers

MSMQ triggers allow an application to receive notifications in the application when the message queue received a message.
In general, triggers should be chosen over queue polling (asynchronous reading of messages from the message queue).

.Net allows using an `AsyncCallback` method instead of the MSMQ triggers as exemplified below:

```
private void receiveOnQueue(IAsyncResult asyncResult)
{
  System.Messaging.Message msg = this.rxMsgQueue.EndReceive(asyncResult);
  //re-arm the callback delegate
  this.rxMsgQueue.BeginReceive(System.TimeSpan.FromMinutes(60), null, new
  AsyncCallback(this.receiveOnQueue));
}

...
rxMsgQueue.BeginReceive(System.TimeSpan.FromMinutes(60), null, new
AsyncCallback(this.receiveOnQueue));
...
```

7. MSMQ queue path syntax (1/3)

MSMQ queues are addressed with a queue path. There are 2 general path syntaxes:

1. FormatName syntax (1/2):

A FormatName uniquely identifies an MSMQ queue using connection details and the queue's path.
No additional name resolving is necessary (Windows directory service is not contacted for resolving the queue name).
Example public queue: `"FormatName:DIRECT=OS:MyMachine\MyQueue"`
Example private queue: `"FormatName:DIRECT=OS:MyMachine\Private$\MyQueue"`
When to use FormatName syntax:

a. Send message directly to a computer
b. Send messages over the Internet
c. Send / receive messages to any queue while operating in domain, workgroup, or offline mode
d. Message routing, authentication and encryption not needed (these require the domain controller)
→ In general, when working without domain controller, use the FormatName syntax.

7. MSMQ queue path syntax (2/3)

1. FormatName syntax (2/2)

FormatName syntax is different for public queues and private queues.

Public queue:

```
FormatName:DIRECT=<protocol>:<dest. IP address or hostname *>\<queue name>
```

Private queue:

```
FormatName:DIRECT=<protocol>:<dest. IP address or hostname *>\Private$<queue name>
```

*** If the message sender and receiver are co-located on the same machine, the destination address can be substituted by a dot (.).**
Protocol: TCP, SPX, OS (native computer naming), HTTP, HTTPS.

2. Path name syntax:

Path name syntax is a shorthand notation for identifying a queue.
When using the queue path, the Windows domain controller (LDAP DB, CIFS server) translates the path name into the associated FormatName before accessing the queue.
Thus the path name syntax requires a windows domain and domain controller.
Example: `"MyMachine\Private$\MyQueue"`

7. MSMQ queue path syntax (3/3)

Queue paths for the different queue types:

Queue type	Path syntax
Public queue	<*Computername*>\<queue name> .\<queue name> (dot shorthand only when sender and receiver are on the same machine) FormatName:Public=<queue GUID> (1)
Private queue	<*Computername*>\Private$\QueueName .\Private$\<queue name> (dot shorthand only when sender and receiver are on the same machine) FormatName:DIRECT=SPX:<network number>:<host number>\<queue name> (1) FormatName:DIRECT=TCP:<IP address>\<queue name> (1) FormatName:DIRECT=OS:<computer name>\<queue name> (1) FormatName:DL=<distribution list GUID> (2) FormatName:DIRECT=http://<client name>/msmq/<queue name> (1)
Journal queue	<*Computername*>\<queue name>\Journal$
Computer journal queue	<*Computername*>\Journal$
Computer dead-letter queue	<*Computername*>\Deadletter$
Computer transaction dead-letter queue	<*Computername*>\XactDeadletter$

(1) FormatName syntax.
(2) FormatName syntax for sending messages to a distribution list.

8. MSMQ programming guidelines (1/2)

Message queueing, as every other form of distributed computing, needs careful consideration as regards performance. The following guidelines should be followed when programming an MSMQ application.

Do only local receives:

Receiving from remote queues on other machines is costly (network delay, protocol overhead). Read operations from queues should be done from an application running on the same host as the message queue.

Avoid accesses to MQIS (Microsoft Queue Information Store):

MQIS accesses are costly. Therefore, operations such as dynamic lookups of queue references should be avoided where possible.
One way to achieve this is using hardcoded queue references. The drawbacks of this approach (reduced flexibility) has to be weighed against performance gains.

Decouple message receiver from application:

The message receiving part of an application should be decoupled from the rest of the application so that it does not block and can react to other events in case there are no messages to be received.
This can be achieved with a receiver thread or by using receive operation timeouts.

8. MSMQ programming guidelines (2/2)

Prudent use of transactions:

Transactions, like other special message queueing features, are costly (slower than normal message passing).
Therefore, use transactions only where necessary.

Static queue creation:

Queue creation is costly, too. Message queues should be created administratively before starting the application and not dynamically at run time.

Careful use of acknowledgments:

Use message acknowledgments only where the application requires confirmation of successful delivery of messages.
In most cases, acknowledgments can be omitted and message delivery failures handled in the application itself.

9. Programmatic access to MSMQ

MSMQ can be accessed programmatically through native Win32 and COM APIs.
.Net provides access to the MSMQ system through the `System.Messaging` namespace allowing to:
A. Connect to queues, send / receive messages
B. Monitor queues
C. Administer queues

Important classes of `System.Messaging`:

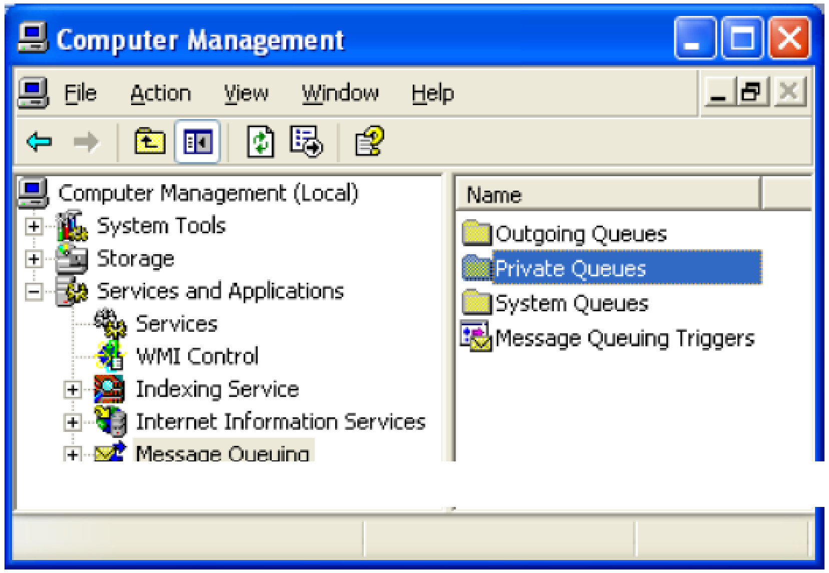
`System.Messaging.Message`	The message class.
`System.Messaging.MessageQueue`	The message queue.
`System.Messaging.XmlMessageFormatter`	Encapsulates the message body in XML format.
`System.Messaging.BinaryMessageFormatter`	Encapsulates the message body in binary format.
`System.Messageing.ActiveXFormatter`	Encapsulates the message body in ActiveX compatible format.

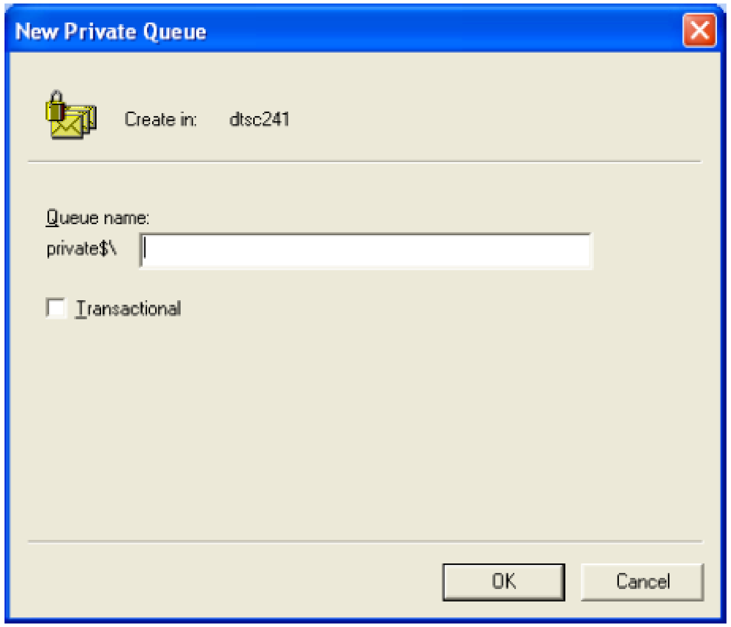
10. Creating queues

Queues can be created either administratively or programmatically.

1. Create queues administratively:

Control Panel → Administrative Tools → Computer Management.





2. Create queues programmatically:

```
MessageQueue mq = MessageQueue.Create(<queue path>);
```

11. Sending messages to multiple destinations

There are different ways with MSMQ to send a message to multiple destinations.

1. Distribution list:

Distribution lists are special format names that are stored in the Active Directory Domain Service (using distribution lists requires a running AD).
Format name: `DL=<dist. list identifier>`
The format name is resolved using the AD.
Using distribution lists allows hiding the location of the receiver queue to the sending and receiving applications (transparency).

2. Multicast address:

When using multicast IP addresses, the distribution to multiple receivers is performed by the network and not MSMQ.
Format name: `MULTICAST=<Address>:<Port>`
Multicast addresses are in the IP range 224.0.0.0 through 239.255.255.255.

3. Multiple-element format name:

Multiple destinations can be specified in the format name with a list.
Multiple-element format name = list of comma-separated format names.

```
<format name 1>,<format name 2>,<format name 3>...<format name n>
```

12. Reliable messaging (1/3)

MSMQ stores messages in volatile memory (RAM) in order to increase performance.
In case of a crash it is possible that messages are lost.
There are different ways to make the message transmission reliable.

1. MSMQ transactions:
Message transactions allows sending a sequence of messages that are either received in their entirety or no message at all ("all or nothing").
N.B.: The transaction support of MSMQ is limited to messaging itself. It is not possible to include, for example, database accesses in the transaction (but MSMQ can participate in MS DTC transactions).

Example:

```
//transactional send
MessageQueueTransaction msgTrans = new MessageQueueTransaction();
msgTrans.Begin();
msgQ.send("Hello World", msgTrans); msgQ.send("Goodbye World", msgTrans);
msgTrans.Commit(); //now the messages are sent
//transactional receive
msgTrans.Begin();
Message msg;
msg = msgQ.Receive(msgTrans);
msgTrans.Commit();
```

DTC: Distributed Transaction Coordinator

12. Reliable messaging (2/3)

2. Acknowledgment types (1/2):

MSMQ provides a range of different acknowledgment types for reliable messaging.
Acknowledgments are special MSMQ messages sent back to the sending MSMQ system to inform about positive (message successfully received) or negative (message timed out, i.e. not received from the queue by an application) reception of a message.
Typical sequence for message sender:

```
MessageQueue msgQ = new MessageQueue(@".\private$\Orders");
msgQ.DefaultPropertiesToSend.AcknowledgeType = AcknowledgeTypes.FullReachQueue |
AcknoledgeTypes.FullReceive;
msgQ.DefaultPropertiesToSend.AdministrationQueue = new MessageQueue(@".\private$\Ack");
msgQ.Send("Sample");

//Receive ack message
Message msg;
msg = msgQ.Receive(new TimeSpan(0), MessageQueueTransactionType.Single);
//Receive Acknowldgement Messages
MessageQueue adminQ = new MessageQueue(@".\private$\Ack");
adminQ.MessageReadPropertyFilter.CorrelationId = True;
msg = adminQ.Receive(new TimeSpan(0,0,5));
  if (msg.MessageType = MessageType.Acknowledgment) {
    switch(msg.Acknowledgment) {
      case Acknowledgment.ReachQueue:
        break;
      case Acknowledgment.Receive:
        break;
...
```

12. Reliable messaging (3/3)

2. Acknowledgment types (2/2):

There are different acknowledgment types available. The `Message.AcknowledgeType` property is a bit mask, so different acknowledgment types can be combined.

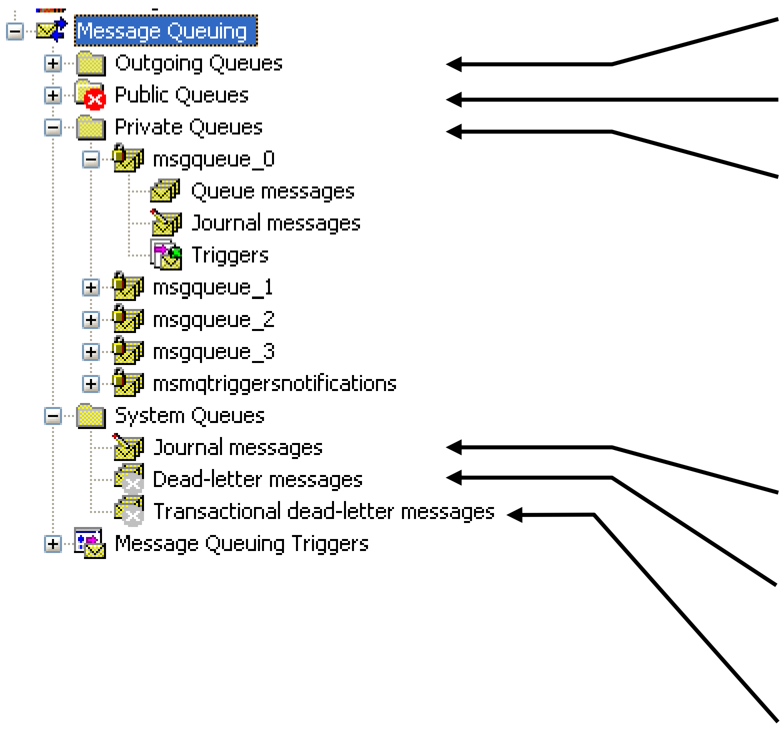
Acknowledgment type	Description
AcknowledgmentTypes.FullReachQueue	Success and failure indication to be delivered to the queue.
AcknowledgmentTypes.FullReceive	Success and failure indication to be received from the queue by the application.
AcknowledgmentTypes.NegativeReceive	Negative acknowledgment to be sent when the application fails to receive the message from the queue (explicit negative acknowledgment).
AcknowledgmentTypes.NotAcknowledgeReachQueue	A mask used to request a negative acknowledgment when the original message cannot reach the queue. This can happen when the time-to-reach-queue timer expires or when a message cannot be authenticated.
AcknowledgmentTypes.NotAcknowledgeReceive	A mask used to request a negative acknowledgment when an error occurs that prevents the original message from being received from the queue before its time-to-be-received timer expires.
AcknowledgmentTypes.PositiveArrival	Successful delivery to the queue.
AcknowledgmentTypes.PositiveReceive	Successful receive from the queue by the application.
AcknowledgmentTypes.None	Request for no acknowledgment at all (neither positive nor negative).

3. Property Message.Recoverable:

If the property `Message.Recoverable` is set to true, MSMQ stores the message to hard disk so that in case of a crash the message may be recovered after a reboot.

13. Queue management

A minimal tool for administering MSMQ can be found under Control Panel → Administrative Tools → Computer Management.



Undelivered messages

Public queues (created administratively or programmatically)

Private queues with:

a. Queue messages (successfully delivered to queue, but not retrieved by an application)

b. Journal message (copies of messages in „Queue messages“ if the property `UseJournalQueue` is set to true on the queue)

c. Triggers (contains triggers that fire when a message is placed in this queue)

Global journal message queue

MSMQ places a copy of every undeliverable message or message that expires before it is received into this queue.

Same as the dead-letter queue, but for all transactional messages.

14. MSMQ and WCF

WCF (Windows Communication Foundation) is a unified communication framework for distributed applications.
WCF defines a common programming model and unified API for clients and services to send messages between each other.
WCF is very flexible and allows using different transport protocols for delivering messages.
MSMQ is one of the transport mechanisms supported by WCF.

The transport protocol is selected through a binding (the 'B' in 'ABC').
WCF supports 2 bindings based on MSMQ:
A. NetMsmqBinding:
NetMsmqBinding is suitable when .Net WCF is used on both sides of the communication (sender and receiver).

B. MsmqIntegrationBinding:
MsmqIntegrationBinding may be used to connect a .Net WCF application to an existing native MSMQ application already in place.