

Integrating with TCP/IP using WebSphere Message Broker

Preetha Ghosh
Ben Thompson

April 27, 2011

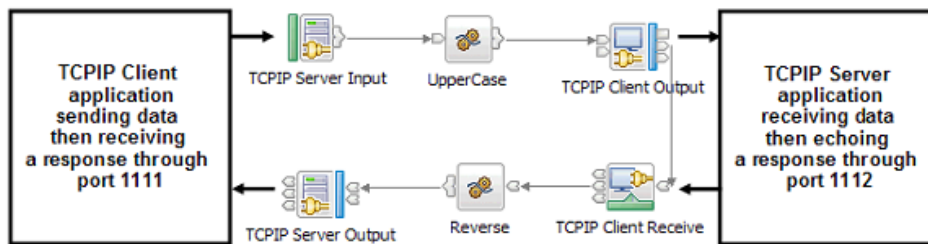
WebSphere Message Broker provides support for integrating applications using TCP/IP connections. This article describes the basics of TCP/IP communication, and uses sample solutions from the airline and banking industries to explain how Message Broker adds value to architectures based on TCP/IP integration.

The TCP/IP communication protocol

TCP/IP is a communication protocol used for transferring data between two applications, which can be running on different computers connected by a network. The TCP/IP protocol is a fast networking option for low bandwidth applications, and is ideal for small devices or for cases where a cost is associated with every byte of data transferred. Data transfer can be bidirectional, and the sequence of data transfer is also maintained. However, there are some limitations to the protocol:

- It is non-transactional -- there is no transactional coordination between sender and receiver.
- It is non-persistent -- the data is written to an in-memory buffer between sender and receiver.
- It has no built-in security.
- It has no standard way of signalling the start and end of a message.

Because of the above restrictions, some higher-level messaging protocols have been developed to run on top of the TCP/IP stack, such as HTTP and MQ, but of course these solutions are not always feasible. If an existing application uses raw TCP/IP sockets to transfer data, and its interface cannot be changed, then a good solution to expand connectivity is to use IBM® WebSphere® Message Broker (hereafter called Message Broker) and its TCP/IP nodes. You can add Message Broker to a system that uses TCP/IP for transport in order to generate a more flexible architecture for communication between the components, as shown in the diagram below. For example, in circumstances where either the client or server application could go out of service, it may be easier to keep Message Broker online. Another example is using Message Broker as a router to help integrate two systems without making any changes to their existing interfaces.

Figure 1. Message Broker added to a system that uses TCP/IP for transport

Every TCP/IP connection has a server end and a client end, where the server application listens for a connection on a given port and the client application requests a connection from the listening server application.

Many systems and applications follow an architectural model that divides the responsibility for providing functionality between a server and a client. Such distributed applications are useful because the client and server can be installed on different hardware and in different physical locations, enabling workloads to be split based on where the best processing power is available. Typically, such architectures include one or more clients that communicate with a single central server that resides on a different host. The client requests some data from a server, which then responds with the answer.

Prior to the availability of message-oriented middleware (for example, in products such as WebSphere MQ), traditional client/server architectures ran synchronously, with both halves online at the same time. The server half of the architecture carries out a job on behalf of the client, so normally the interaction begins when a client communicates with a server by sending a request over a network. The server then acts on the request and responds to the client. A common example of this architecture is a Web browser that behaves as a client when a URL is requested over HTTP. A Web server then responds by providing the HTML of the Web page.

TCP/IP works by having a client socket and a server socket. Sockets are provided by the operating system as a way to receive data using the TCP/IP protocol stack, and supply it to a running process or application. The address of a socket is a combination of the host's IP address (or hostname) plus a port number. TCP/IP sockets are one of the simplest methods to connect two applications. The server end of the socket is created on the machine where the server application is running and given a port, and it then listens on this port for a connection from a client application. The client connects to the server using a machine address and port number. The client can be on any machine with network access to the machine running the server:

Figure 2. Connection between client and server application

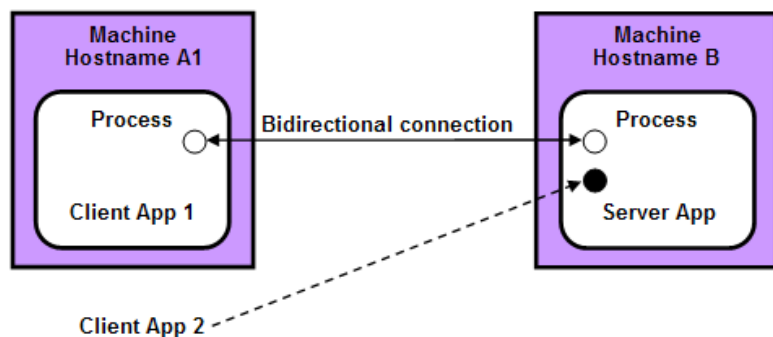
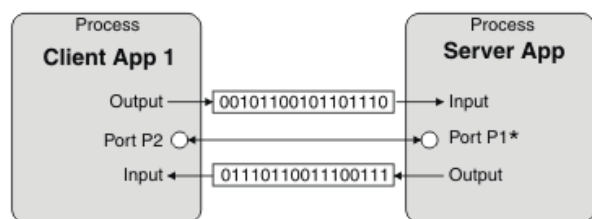


Figure 2 shows the formation of a bi-directional client/server connection. The server port is still available to receive further client connections from other applications, as indicated by the black circle and dashed line.

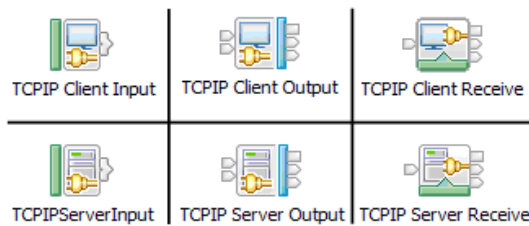
The architectural diagram below in Figure 3 shows how the client and server applications have access to a socket that has an input and an output stream. The client end of the socket is created on the machine where the client application is running and given a port, and it then listens on this port for a connection from a server application. Once the connection is made by the server, both the server and client can use the socket to send and receive information. The output stream of the socket becomes the input stream to the server application and vice-versa.

Figure 3. Connection between client socket and server application



WebSphere Message Broker TCP/IP nodes

WebSphere Message Broker V6.1 provided six new nodes that let flow developers access the TCP/IP input and output stream. The client nodes and server nodes have identical function in terms of accessing data streams -- their main functional difference is that one set uses a client connection and the other set a server connection. Message Broker provides two output nodes: TCPIPServer and TCPIPClient. These nodes are placed at the end of a message flow and are used to send data out to a TCP/IP client application and TCP/IP server application respectively. Message Broker also provides two receive nodes: TCPIPServer and TCPIPClient. These nodes are placed in the middle of a message flow and receive data that arrives on connections that have already been established.

Figure 4. Message Broker TCP/IP nodes

- TCPIPClient Input node -- Lets you start a message flow on receiving data from a TCP/IP connection to a server application.
- TCPIPClient Output node -- Obtains a connection to a server application and sends data over that connection.
- TCPIPClient Receive node -- Lets you receive data from a TCP/IP connection to a server application in the middle of a message flow.
- TCPIPServer Input node -- Lets you start a message flow on receiving data from a TCP/IP connection to a client application.
- TCPIPServer Output Node -- Obtains a connection to a client application and sends data over that connection.
- TCPIPServer Receive Node -- Lets you receive data from a TCP/IP connection to a client application in the middle of a message flow.

TCP/IP node threading model

The TCP/IP nodes do not directly create or manage TCP/IP connections -- that job is done by the connection manager, which is part of the Message Broker execution group. This internal part of Message Broker product is not exposed to developers, but it is useful to see how it works so that you can understand the architectural scalability of message flows. Consider data coming into Message Broker. It provides two different kinds of TCP/IP Input node: TCPIPServer Input node and TCPIPClient Input node:

TCPIPServer Input node threading model

A message flow that contains a TCPIPServer Input node listens on a port, waiting for a client application to try to connect. Any application that tries to connect to the Broker's hostname on the port number specified by the TCPIPServer Input node's properties will attempt to start a new connection. By default, the broker will accept up to 100 such connections, and you can alter this value using the MaximumConnections property of the node's associated configurable service. Each connection that is made will try to begin a message flow thread. By default there will only be one thread available, unless you have configured additional instances for the TCPIPServer Input node.

If further connections are made, and that thread is busy, then the Broker's connection manager will wait until the thread finishes its work and becomes available. As soon as an application connects, an empty message will be propagated to the node's output terminal named Open. Use this branch of a flow to define logic that should be executed when a connection has been opened but no data has been processed on it yet. As data is then sent over the connection, the node will send it down the output terminal named Out. The data will be parsed according to the settings on both the Input Message Parsing Tab and the Records and Elements tab. Many different options are

possible depending on the physical format of the data. The node also lets you specify under what circumstances a node should close an existing connection, and whether the input stream of data should be reserved for use by later nodes in the flow, such as a TCPIPServer Receive node, as shown in the later example that demonstrates the dynamic length feature.

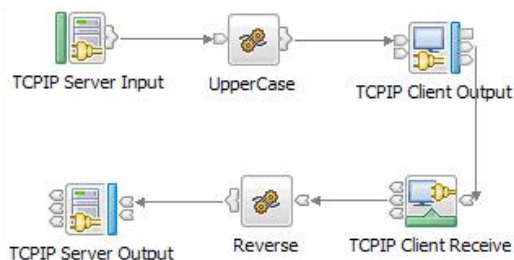
TCPIPClient Input node threading model

A message flow that contains a TCPIPClient Input node listens on a port, waiting for a server application to try to connect. By default, no connections are created when a message flow containing a TCPIPClient Input node is deployed, unless its associated configurable service has configured its MinimumConnections property to be greater than zero. With this configuration, the node will wait for a connection to be established by another output or request node. In addition to a connection being available, there must also be data presented to the socket in order for a flow driven by a TCPIPClient Input node to begin processing.

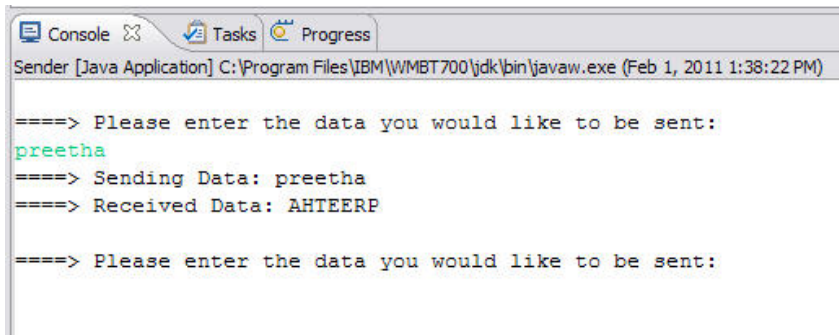
If the MinimumConnections property of the node's configurable service has been set to a value greater than zero, then when the flow is deployed, the broker will create connections until the MinimumConnections figure is reached. These connections will be available, but the flow is paused until data arrives on the socket to be processed. Bearing in mind this architectural design, to successfully scale message flows that contain TCPIPClient Input nodes, you must consider adding additional instances to the node, and configuring the MinimumConnections and MaximumConnections of its associated configurable service.

Client and server example

Figure 5. Client and server example message flow



WebSphere Message Broker helps you create more flexible architectures for systems that use TCP/IP for communication between their components, as shown below in the example message flow. This example uses two Java™ applications. The first one is the provided sample application named Sender.java, which lets a user enter character data on a command line and send it to a socket. This data entry starts a propagation through the message flow. The message flow contains a TCPIPServer Input node, which receives a TCP/IP stream into Socket 1112. The compute node UpperCase turns the text character received into UPPERCASE and then sends out the message through the TCPIPClient Output node to Socket 1113. A second provided Java application, Receiver.java, is a simple echo application that listens for a TCP/IP stream and sends the same data back into the same socket (in this example, Port 1113). Once the data is sent back by Receiver.java, the TCPIPClient Receive node receives the data and uses the Compute node (named Reverse), to reverse the order of the characters before returning the data to the original Sender application on Socket 1112 through the TCPIPServer Output node.

Figure 6. Command-line output of sender application


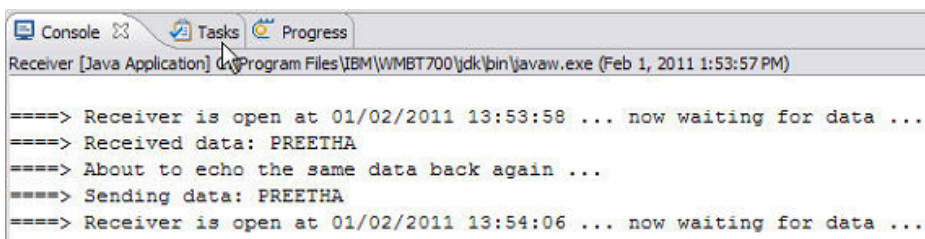
```

Sender [Java Application] C:\Program Files\IBM\WMBT700\jdk\bin\javaw.exe (Feb 1, 2011 1:38:22 PM)

====> Please enter the data you would like to be sent:
preetha
====> Sending Data: preetha
====> Received Data: AHTEERP

====> Please enter the data you would like to be sent:

```

Figure 7. Command-line output of receiver application


```

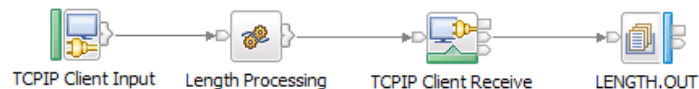
Receiver [Java Application] C:\Program Files\IBM\WMBT700\jdk\bin\javaw.exe (Feb 1, 2011 1:53:57 PM)

====> Receiver is open at 01/02/2011 13:53:58 ... now waiting for data ...
====> Received data: PREETHA
====> About to echo the same data back again ...
====> Sending data: PREETHA
====> Receiver is open at 01/02/2011 13:54:06 ... now waiting for data ...

```

Dynamic length example

In more complex scenarios, you can dynamically control a TCP/IP Receive node in defining the number of bytes of data it should take from the input TCP/IP stream. Setting the Length property in the local environment can override the TCP/IP connection used by the TCPIPClient Receive node or the TCPIPServer Receive node. If the input tree to the node has a local environment including a value for \$LocalEnvironment/TCPIP/Receive/Length, then this value controls the number of bytes to be read. The local environment value overrides the Length (bytes) property specified on the node. If the Record detection property is set to anything other than Fixed Length, then the local environment setting is ignored. If the local environment setting is not present or evaluates to null, it is ignored and the value hard-coded on the node is used. The local environment can be set up by any appropriate node (such as the Compute, JavaCompute, or PHP node) prior to the TCPIPClient Receive node. When using the Compute node for this purpose, remember to change the default Compute Mode property, so that the ESQL contained in the node has control over the local environment tree. The example below demonstrates this concept:

Figure 8. Dynamic length example message flow

As soon as the data is received by the TCPIPClient Input node, the node reads the first (fixed-length) four bytes of data from the input bitstream and propagates it down the Out terminal. The value of four bytes has been set on the TCPIPClient Input node statically. The next node is a Compute node named Length Processing, which interprets the first four bytes of the message data. In this example, these four bytes contain the length of the remainder of the message. This length value is set in the local environment, as shown in Listing 1 below, and it dynamically

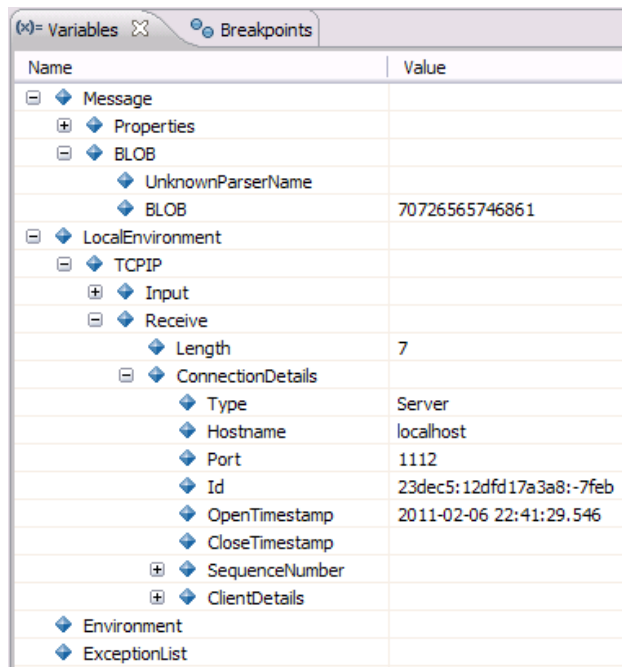
controls the following TCIPClient Receive node. The TCIPClient Receive node takes data (of the dynamically configured length) from the bitstream and propagates it to the following node as a BLOB domain message. Finally, the MQOutput node writes an output message to a queue named LENGTH.OUT. The output message is four bytes shorter than the original data in the TCP/IP stream.

Listing1. Compute node ESQL

```
CREATE COMPUTE MODULE FunctionalFlow_Length_Processing
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
    SET OutputLocalEnvironment = InputLocalEnvironment;
    SET OutputLocalEnvironment.TCIP.Receive.Length =
        CAST(InputRoot.BLOB.BLOB AS CHAR CCSID 1208);
    RETURN TRUE;
END;
END MODULE;
```

Figure 9 demonstrates the use of the Dynamic Length feature. In this example, the stream of text sent over TCP/IP was 0007preetha, which in hex is x3030303770726565746861. In this screen shot, taken from the debugger, the local environment contains the four-byte value of 0007, which causes the following TCIPClient Receive node to read seven bytes of data into the Message section of the tree. The screen shot shows the message immediately prior to being written out to the output queue at the end of the flow:

Figure 9. Environment tree showing received length



Name	Value
Message	
Properties	
BLOB	
UnknownParserName	
BLOB	70726565746861
LocalEnvironment	
TCIP	
Input	
Receive	
Length	7
ConnectionDetails	
Type	Server
Hostname	localhost
Port	1112
Id	23dec5:12dfd17a3a8:-7feb
OpenTimestamp	2011-02-06 22:41:29.546
CloseTimestamp	
SequenceNumber	
ClientDetails	
Environment	
ExceptionList	

Real-life applications

Application communication via TCP/IP and sockets is widespread in many industries, so the Message Broker TCP/IP nodes have many practical uses. This article focuses on two industry examples to demonstrate practical applications of the technology:

- Airline industry: MATIP and BATAP
- Banking industry: ISO8583

For more information on how the Message Broker TCP/IP nodes might be able to help your business, please contact the authors.

Airline industry: MATIP and BATAP

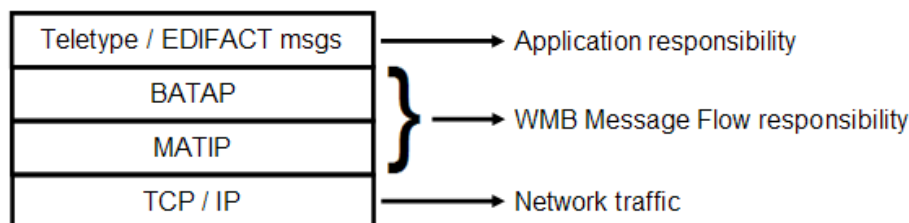
The airline industry uses the Mapping of Airline Traffic over Internet Protocol (MATIP) communication standard to receive and transmit messages over TCP/IP networks between many kinds of applications, such as airline reservation and ticketing systems. Adoption of MATIP is widespread both internally on company intranets, and in B2B integration scenarios linking private networks and partner extranets. The most common types of messages that use the MATIP standard are EDIFACT messages and International Air Transport Association / Société Internationale de Télécommunications Aéronautique (IATA / SITA) teletype messages. (IATA is an international trade body representing the airline industry in general and the majority of international airlines, while SITA is a company that provides network solutions for the airline industry.

Typically, Message Broker's routing and transformation capabilities are used to bridge applications between TCP/IP endpoints and WebSphere MQ queues. This usage extends the reach of airline industry applications beyond their usual boundaries, as in the private SITA network or the StarNet infrastructure, which links the networks of the Star Alliance partner airlines. The pattern discussed below provides a sample development accelerator that decreases the time an integration developer needs to build a MATIP solution. MATIP support is provided for communicating two main types of airline network traffic:

- **Transactional traffic** -- Also known as **Type A**, provides real-time query and response integration. It is used for higher priority messages, and if a message is lost in transit, it must be resent.
- **Messaging traffic** -- Also known as **Type B**, provides higher protection but is less immediate. Although data is sent synchronously, the protocol provides for resending messages lost during network transmission.

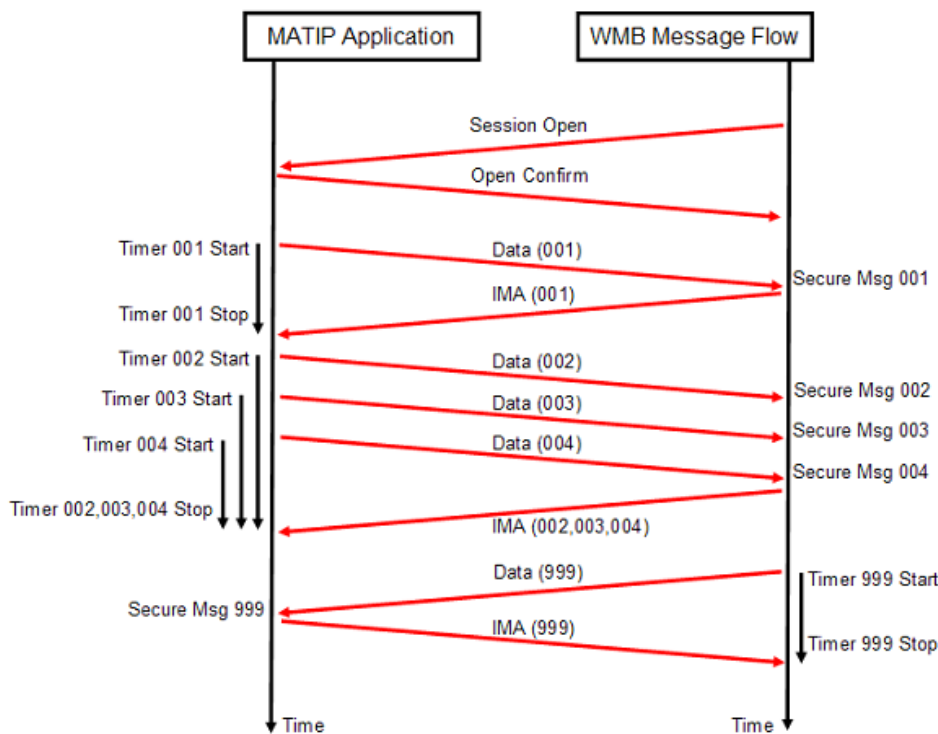
From this point forward, MATIP will be used as a shorthand for MATIP Type B traffic. Message Broker takes responsibility for the persistence, acknowledgement, and resend requirements of the MATIP stack. The actual data messages themselves (normally containing teletype or EDIFACT, though the flows don't care) are transported through the message flows unchanged. In other words, the application layer of the protocol stack is left as the responsibility of the application to interpret, without needing to be concerned about the handshake messages that establish and maintain the communication sessions. Figure 10 and the notes below show the ordering of the protocol stack and the purpose of each layer:

Figure 10. Protocol layers and division of responsibility

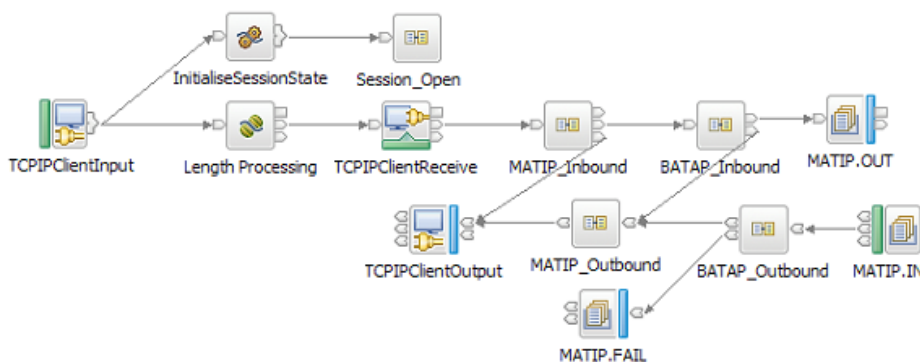


1. A raw TCP/IP stream of data is received and the MATIP messages it contains are interpreted to establish a session. The MATIP messages include a BATAP envelope.
2. The Type B Application To Application Protocol (BATAP) layer of the stack ensures the transfer of responsibility for the data -- things like appending a sequence number to messages, acknowledging receipt of the data, and resending the data if acknowledgements are not received in a timely manner.
3. The application layer of the stack is concerned with parsing and interpreting the data itself. This layer is not part of the pattern considered here.

To understand the role that Message Broker can play in MATIP scenarios, consider the typical interactions that occur when communicating with a MATIP application, shown below in Figure 11. The passage of time runs vertically downwards in the figure, and the horizontal red lines show data transmission between the MATIP application and the Message Broker message flow. Message Broker can act as a TCP/IP client or a TCP/IP server when interacting with a MATIP application. The sample message flow discussed below acts as the client, which means that it is responsible for sending a "Session Open" message to initiate the session. Once an "Open Confirm" message has been received by the flow in response, data can be exchanged.

Figure 11. Interactions between MATIP application and Message Broker

The message flow can receive data messages over TCP/IP from a MATIP application and secure them on a WebSphere MQ queue. Likewise, it can receive messages from a WebSphere MQ queue and transmit them over TCP/IP. These two directions of data flow are shown by the two main horizontal branches of the message flow shown below:

Figure 12. MATIP client message flow

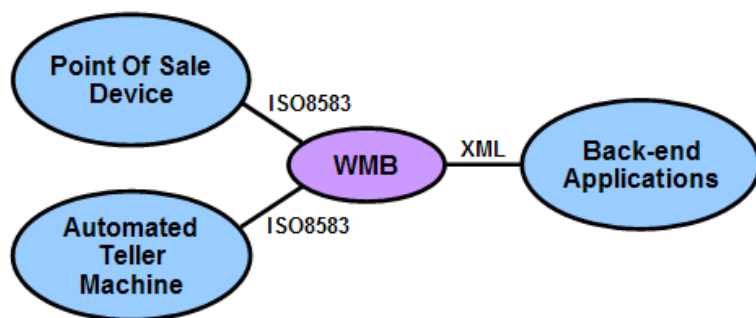
The purpose of the BATAP envelope that is used when communicating with MATIP applications is to assign a three-digit sequence number to every message that is transmitted, known as an SRLN (SeRial Number). When a message has been safely received and secured, the receiving application sends a BATAP acknowledgement message, commonly known as an IMA. This message contains the BATAP sequence number. An application is also allowed to use a single IMA message to acknowledge the safe arrival of multiple inbound data messages (as implied by the IMA for messages 002, 003, and 004, shown in Figure 11 above).

When exchanging messages and IMA acknowledgements, the sending side of a pair of MATIP applications also maintains what is known as a window. The window size determines the number of messages that have been sent but are currently outstanding, waiting for an acknowledgement to be received. Each time an IMA is received, a space in the window becomes available allowing another data message to be sent. A timer is started when each message is sent. If a predefined time is exceeded, then the data should be retransmitted. This kind of message is known as a Possible Duplicate Message (PDM). The MATIP and BATAP protocols do not specify a maximum number of retransmissions (to protect against the possibility of infinitely repeating a message), but MATIP applications can implement their own policies. All of these behaviours are catered to by the message flow, so that someone wanting to integrate a WebSphere MQ based application with a MATIP network can do so quickly without having to worry about some of the protocol's finer points of detail.

Banking industry: ISO8583

The International Standards Organization (ISO) is an international network of several groups who are responsible for the development, definition, and publication of data standards used by the public and private sectors in many industries. A common ISO standard for the banking and financial services sector is ISO8583, and it specifies a common interface by which messages originating from credit and debit cards can be interchanged between devices and card issuers. Typical uses of ISO8583 are to define the message format for data exchanged with a point-of-sale device or automated teller machine (ATM). Messages commonly contain information about the value of a transaction, where it originated (which store or ATM), card account number, and bank sort code. Back-end applications to which data is sent can have a variety of purposes, such as core banking systems, bank statement retrievals, transfer of funds between bank accounts, payment of bills, or purchase of mobile phone credit. Message Broker can assist in these kinds of middleware scenarios by enabling the transformation of data between the ISO8583 standard and more convenient data formats such as XML-based Web services), and via other protocols such as WebSphere MQ, FTP, or HTTP. This role is summarised below:

Figure 13. Role of Message Broker in ISO8583 message scenarios

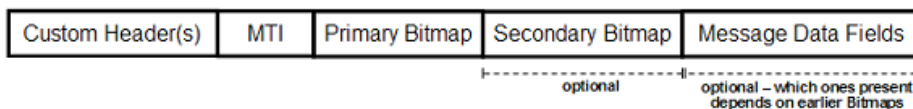


Often, ISO8583 messages are preceded by customised headers, but the core ISO8583 message, shown below in Figure 14, always contains three sections:

- **Message type indicator (MTI)** -- Four numeric digits that specify the version of the ISO8583 standard, message class, message function, and message origin

- **One or more bitmaps** -- A message will always have a one-byte primary bitmap whose individual bits indicate which later fields are present in this particular message instance. The primary bitmap specifies whether fields 1 to 64 are present. If a secondary bitmap is also included, it specifies whether fields 65 to 128 exist. A tertiary bitmap contains information about the next 64 fields, and so on. Primary and secondary bitmaps are common, but messages requiring a tertiary bitmap are very rare.
- **Message fields** -- These elements are defined by the ISO8583 standard, and contain information about the transaction, such as amounts, dates, times, and country codes.

Figure 14. Layout of a typical ISO8583 message



Message Broker TCP/IP nodes can be used to receive data from a point-of-sale device. In order to determine the length of an individual message received from the inbound socket's bitstream, the bitmaps at the beginning of the message must be interpreted in order to deduce which fields are present in the message. This information, combined with knowledge of how long each field is (the ISO8583 standard defines each field with a data type and fixed length), lets you calculate the total length of the message, which can vary from message to message.

Consider the following example. Figure 15 shows a graphical representation of a primary bitmap for an example authorization request message. The eight bytes shown make up the primary bitmap portion of a message. The ticks indicate which message field numbers are available. This example message contains field numbers 4,7,11,12,24,37,38,39,48 and 49. Byte01 represents the fact that field 4 and field 7 exist, which means that Byte01 takes a value in Binary notation of 00010010, which is the hexadecimal value of x12.

Figure 15. Detailed example of a primary bitmap

Byte 01							
01	02	03	04	05	06	07	08
			✓			✓	

Byte 02							
09	10	11	12	13	14	15	16
		✓	✓				

Byte 03							
17	18	19	20	21	22	23	24
							✓

Byte 04							
25	26	27	28	29	30	31	32

Byte 05							
33	34	35	36	37	38	39	40
				✓	✓	✓	

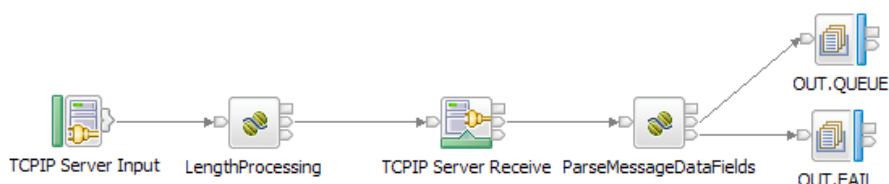
Byte 06							
41	42	43	44	45	46	47	48
							✓

Byte 07							
49	50	51	52	53	54	55	56
✓							

Byte 08							
57	58	59	60	61	62	63	64

Applying the same logic to the other bytes produces a primary bitmap that takes the hexadecimal value x123001000E018000. Figure 16 below shows a sample message flow that demonstrates how the TCP/IP nodes can be used in combination with flow logic in order to receive and interpret an ISO8583 message. In this example, the data follows the same structure as shown in Figure 15 above. The custom header component of the message contains two fixed length fields -- an ATM Identifier and an Encoded Length. The MTI (4 bytes long) follows this header, and then a Primary Bitmap (8 bytes long), and then finally the core part of the message containing the Data Fields.

Figure 16 below shows the sample message flow that provides the Message Broker solution. The TCPIPServer Input node receives a fixed length of 30 bytes (all the fields up to and including the Primary Bitmap) from the input stream and then the Java Compute node named LengthProcessing places the value of the Encoded Length in the local environment. This length is then used to dynamically drive the TCPIPServer Receive node to take the data for the rest of the message from the input stream. The second Java Compute node named ParseMessageDataFields splits the data into its constituent fields. The reason a Java Compute node has been used at this point in the flow is because this code needs to be able to interpret individual bits of a single byte of data. The message set feature (providing message modelling) of Message Broker cannot specify a length smaller than a byte for any individual item of data. If you require bitwise representations within a message model, you can use a map in the WebSphere TX node -- more on this later in the article.

Figure 16. ISO8583 sample message flow

For reference purposes in understanding the attached message flow and code, Table 1 below summarises the inbound message data fields and their values:

Table 1. Input data for ISO8583 message sample

Field name	Length	Data (ASCII format)	Data (Hexadecimal format)
ATM Identifier	15	Ben_ATM_ID00001	x42656E5F41544D5F4944303030303031
Encoded Length	3	123	x313233
MTI	4	1100	31313030
Primary Bitmap	8	Not Applicable (not intended to be readable)	x123001000e018000
Field4 -- Amount of Transaction	12	000000000500	x30303030303030303030353030
Field7 -- Transmission Date Time	14	20110124092315	x3230313130313234303932333135
Field11 -- System Trace Number	6	999999	x393939393939
Field12 -- Local Transaction Date Time	14	20110124092315	x3230313130313234303932333135
Field24 -- Network International ID	3	111	x313131
Field37 -- Retrieval Reference	12	222222	x323232323232202020202020
Field38 -- Approval Code	6	BEN590	x425350353930
Field39 -- Response Code	2	77	x3737
Field48 -- Additional Data Private	Variable	048;T123456;Y0000001;Y0001;Y1234567890;313233 etc.	
Field49 -- Transaction Currency Code	3	978	x393738

In order to drive the sample message flow, a simple Java application is provided that can generate the fixed input message that has been discussed. Figure 17 shows the command-line output of the application in action:

Figure 17. Java sample test program to generate ISO8583 message

[illegible]

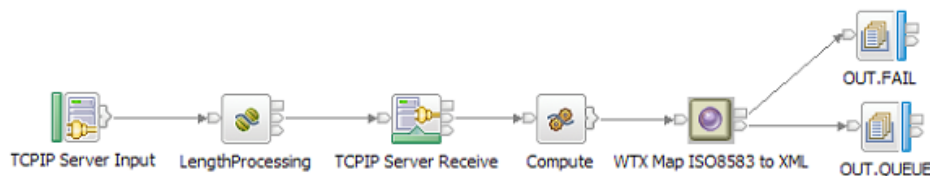
The screen shot below shows the parsing of the data into the environment tree. This parsed data layout was taken at the end of the flow, immediately before the message was written out to the output queue named OUT.QUEUE.

Figure 18. Environment tree showing parsed fields of the ISO8583 message

Name	Value
Message	
LocalEnvironment	
Environment	
ATMIdentifier	Ben_ATM_ID00001
EncodedLength	123
MTI	1100
PrimaryBitmap	123001000e018000
Field4	000000000500
Field7	20110124092315
Field11	999999
Field12	20110124092315
Field24	111
Field37	222222
Field38	BEN590
Field39	77
Field48	048;T123456;Y0000001;Y0001;Y123456781996 ;
Field49	978
ExceptionList	

An alternative method for parsing ISO8583 data is shown in the next example, which uses a WebSphere TX map for the purpose. This pattern uses a message flow shown below in Figure 19, which receives data over TCP/IP, parses it using a WebSphere TX node, and writes it to a WebSphere MQ queue. The TCPIPServer Input node receives 30 bytes of data from the input stream. This data contains some header values, including the length of the ISO8583 message. The first JavaCompute node, named LengthProcessing, determines this length, and places the value in the local environment tree. This specification dynamically controls the following TCPIPServer Receive node, which receives the required number of bytes following the header. These bytes are sent to the next node in the BLOB domain, and the Compute node that follows takes the PrimaryBitmap content (retrieved using the TCPIPServer Input node) and concatenates it with the main message content (retrieved using the TCPIPServer Receive node). The WebSphere TX node that follows uses a map to convert the ISO8583 message into an equivalent XML format.


Figure 19. ISO8583 Sample message flow using WebSphere TX map



The WebSphere TX parser examines the primary bitmap to determine which fields are present in the message. The type tree also has information contained within it that defines the ISO8583 model (how long each field is defined to be). Figure 20 shows the component rules in the

WebSphere TX type tree that enable WebSphere TX to use the information in the primary bitmap to decide whether each field is present or not:

Figure 20. ISO8583 WebSphere TX map component rules

Component	Rule
PrimaryBitmap ISO8583_Fields	
Byte1 BitmapHeader	
Byte2 BitmapHeader	
Byte3 BitmapHeader	
Byte4 BitmapHeader	
Byte5 BitmapHeader	
Byte6 BitmapHeader	
Byte7 BitmapHeader	
Byte8 BitmapHeader	
AmountOfTransaction ISO8583_Fields (0:1)	TESTON(Byte1 BitmapHeader.PrimaryBitmap ISO8583_Fields,4)
TransmissionDateTime ISO8583_Fields (0:1)	TESTON(Byte1 BitmapHeader.PrimaryBitmap ISO8583_Fields,7)
SystemTraceNumber ISO8583_Fields (0:1)	TESTON(Byte2 BitmapHeader.PrimaryBitmap ISO8583_Fields,3)
LocalTransactionDateTime ISO8583_Fields (0:1)	TESTON(Byte2 BitmapHeader.PrimaryBitmap ISO8583_Fields,4)
NetworkInternationalID ISO8583_Fields (0:1)	TESTON(Byte3 BitmapHeader.PrimaryBitmap ISO8583_Fields,8)
RetrievalReferenceNumber ISO8583_Fields (0:1)	TESTON(Byte5 BitmapHeader.PrimaryBitmap ISO8583_Fields,5)
ApprovalCode ISO8583_Fields (0:1)	TESTON(Byte5 BitmapHeader.PrimaryBitmap ISO8583_Fields,6)
ResponseCode ISO8583_Fields (0:1)	TESTON(Byte5 BitmapHeader.PrimaryBitmap ISO8583_Fields,7)
AdditionalDataPrivate ISO8583_Fields (0:1)	TESTON(Byte6 BitmapHeader.PrimaryBitmap ISO8583_Fields,8)
FieldLength3 (0:1) 	
AdditionalData (0:1)	
TransactionCurrencyCode ISO8583_Fields (0:1)	TESTON(Byte7 BitmapHeader.PrimaryBitmap ISO8583_Fields,1)

If you would like to learn more about the examples contained in this article, the examples are available as fully documented patterns in the [Message Broker Patterns Repository](#).

Downloadable resources

Description	Name	Size
Code sample	WMB_TCPIP.zip	7 KB

Related topics

- **WebSphere Message Broker resources**

- [WebSphere Message Broker developer resources page](#)
Technical resources to help you use WebSphere Message Broker for connectivity, universal data transformation, and enterprise-level integration of disparate services, applications, and platforms to power your SOA.
- [WebSphere Message Broker product page](#)
Product descriptions, product news, training information, support information, and more.
- [WebSphere Message Broker V7 information center](#)
A single Web portal to all WebSphere Message Broker V7 documentation, with conceptual, task, and reference information on installing, configuring, and using your WebSphere Message Broker environment.
- [What's new in WebSphere Message Broker V7](#)
WebSphere Message Broker V7 provides universal connectivity with its ability to route and transform messages from anywhere to anywhere. Through its simple programming model and a powerful operational management interface, it makes complex application integration solutions much easier to develop, deploy, and maintain. This article describes the major enhancements in V7.
- [Download free trial version of WebSphere Message Broker V7](#)
WebSphere Message Broker V7 is an ESB built for universal connectivity and transformation in heterogeneous IT environments. It distributes information and data generated by business events in real time to people, applications, and devices throughout your extended enterprise and beyond.
- [WebSphere Message Broker documentation library](#)
WebSphere Message Broker specifications and manuals.
- [WebSphere Message Broker forum](#)
Get answers to your technical questions and share your expertise with other Message Broker users.
- [WebSphere Message Broker support page](#)
A searchable database of support problems and their solutions, plus downloads, fixes, and problem tracking.

- **WebSphere resources**

- [developerWorks WebSphere developer resources](#)
Technical information and resources for developers who use WebSphere products. developerWorks WebSphere provides product downloads, how-to information, support resources, and a free technical library of more than 2000 technical articles, tutorials, best practices, IBM Redbooks, and online product manuals.
- [developerWorks WebSphere application connectivity developer resources](#)
How-to articles, downloads, tutorials, education, product info, and other resources to help you build WebSphere application connectivity and business integration solutions.
- [Most popular WebSphere trial downloads](#)
No-charge trial downloads for key WebSphere products.
- [WebSphere forums](#)

Product-specific forums where you can get answers to your technical questions and share your expertise with other WebSphere users.

- [WebSphere on-demand demos](#)

Download and watch these self-running demos, and learn how WebSphere products and technologies can help your company respond to the rapidly changing and increasingly complex business environment.

- [developerWorks WebSphere weekly newsletter](#)

The developerWorks newsletter gives you the latest articles and information only on those topics that interest you. In addition to WebSphere, you can select from Java, Linux, Open source, Rational, SOA, Web services, and other topics. Subscribe now and design your custom mailing.

- [WebSphere-related books from IBM Press](#)

Convenient online ordering through Barnes & Noble.

- [WebSphere-related events](#)

Conferences, trade shows, Webcasts, and other events around the world of interest to WebSphere developers.

- **developerWorks resources**

- [Trial downloads for IBM software products](#)

No-charge trial downloads for selected IBM® DB2®, Lotus®, Rational®, Tivoli®, and WebSphere® products.

- [developerWorks blogs](#)

Join a conversation with developerWorks users and authors, and IBM editors and developers.

- [developerWorks tech briefings](#)

Free technical sessions by IBM experts to accelerate your learning curve and help you succeed in your most difficult software projects. Sessions range from one-hour virtual briefings to half-day and full-day live sessions in cities worldwide.

- [developerWorks podcasts](#)

Listen to interesting and offbeat interviews and discussions with software innovators.

- [developerWorks on Twitter](#)

Check out recent Twitter messages and URLs.

- [IBM Education Assistant](#)

A collection of multimedia educational modules that will help you better understand IBM software products and use them more effectively to meet your business requirements.

© Copyright IBM Corporation 2011

(www.ibm.com/legal/copytrade.shtml)

Trademarks

(www.ibm.com/developerworks/ibm/trademarks/)