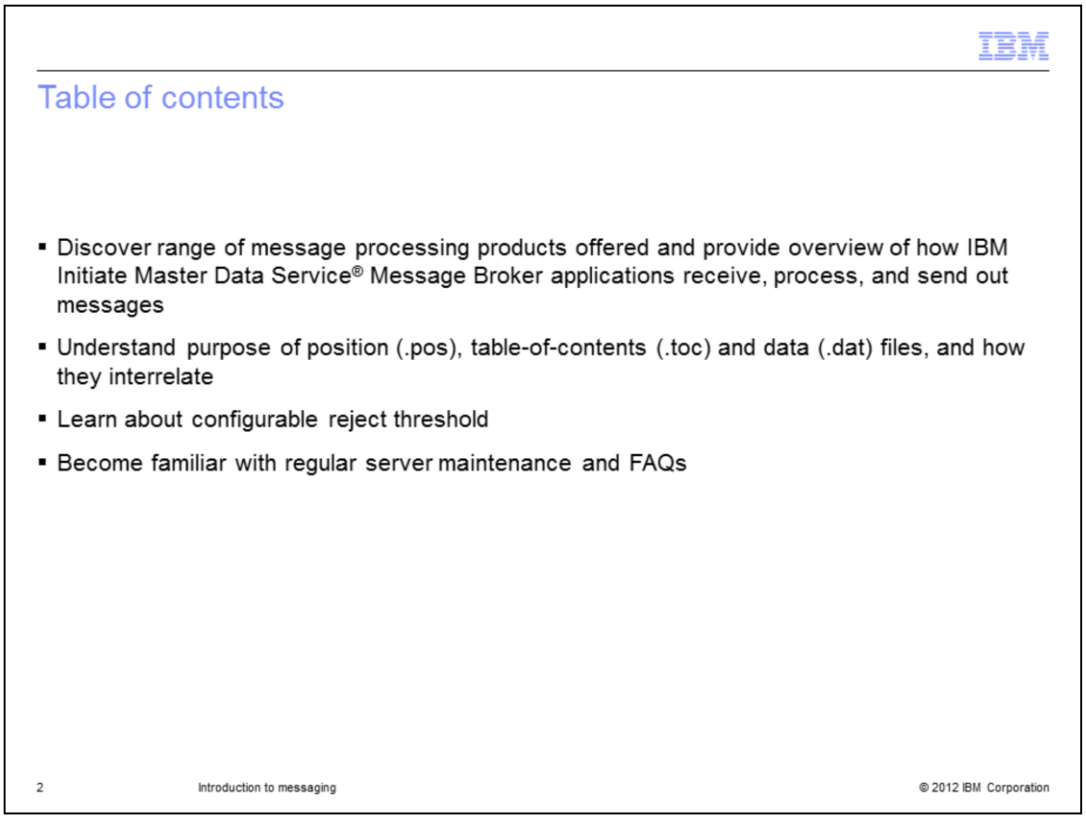
# IBM Initiate

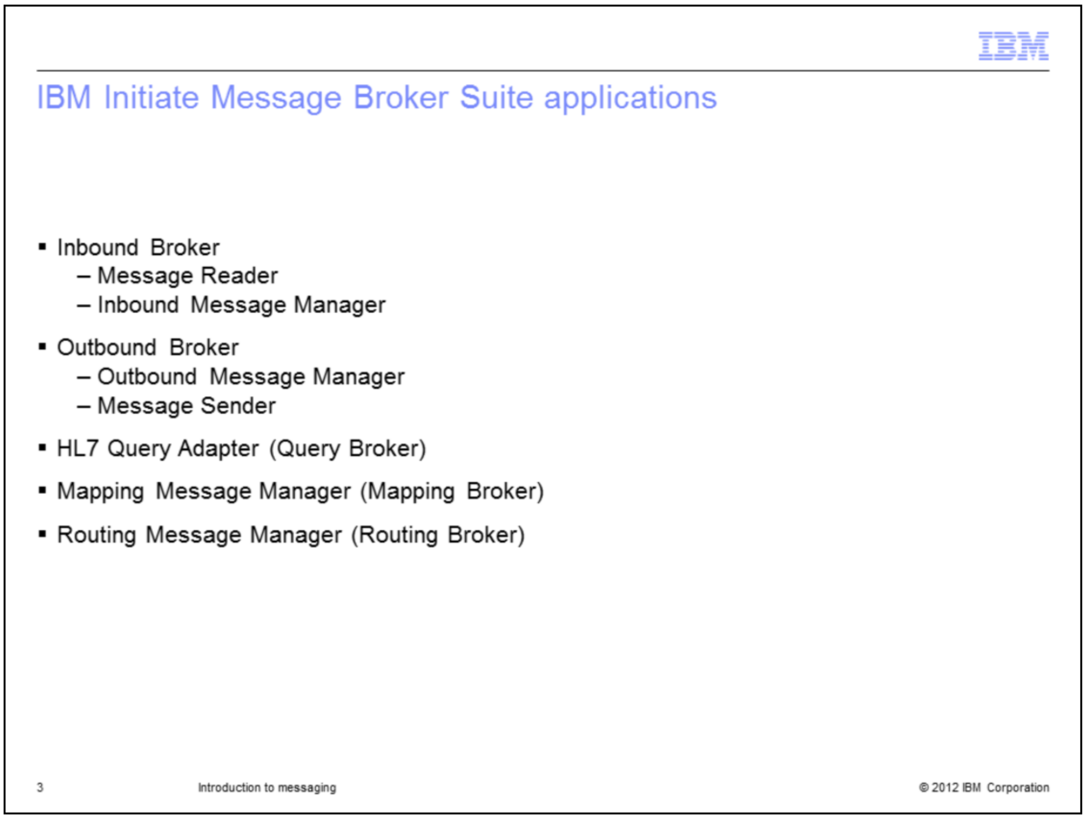
## Introduction to messaging

This is the “Introduction to Messaging” training presentation from IBM Initiate®.



## Table of contents

This training presentation begins with a brief overview of the different applications offered by the IBM Master Data Service Message Broker Suite. You will get a high level description of how the different IBM Initiate messaging applications communicate with other applications. Next, you are shown the purpose and are provided an example of how the .pos, .dat, and .toc files are used. The training then covers reject thresholds and regular server maintenance. Finally, frequently asked questions are discussed.



## IBM Initiate Message Broker Suite applications

- Inbound Broker
  - Message Reader
  - Inbound Message Manager
- Outbound Broker
  - Outbound Message Manager
  - Message Sender
- HL7 Query Adapter (Query Broker)
- Mapping Message Manager (Mapping Broker)
- Routing Message Manager (Routing Broker)

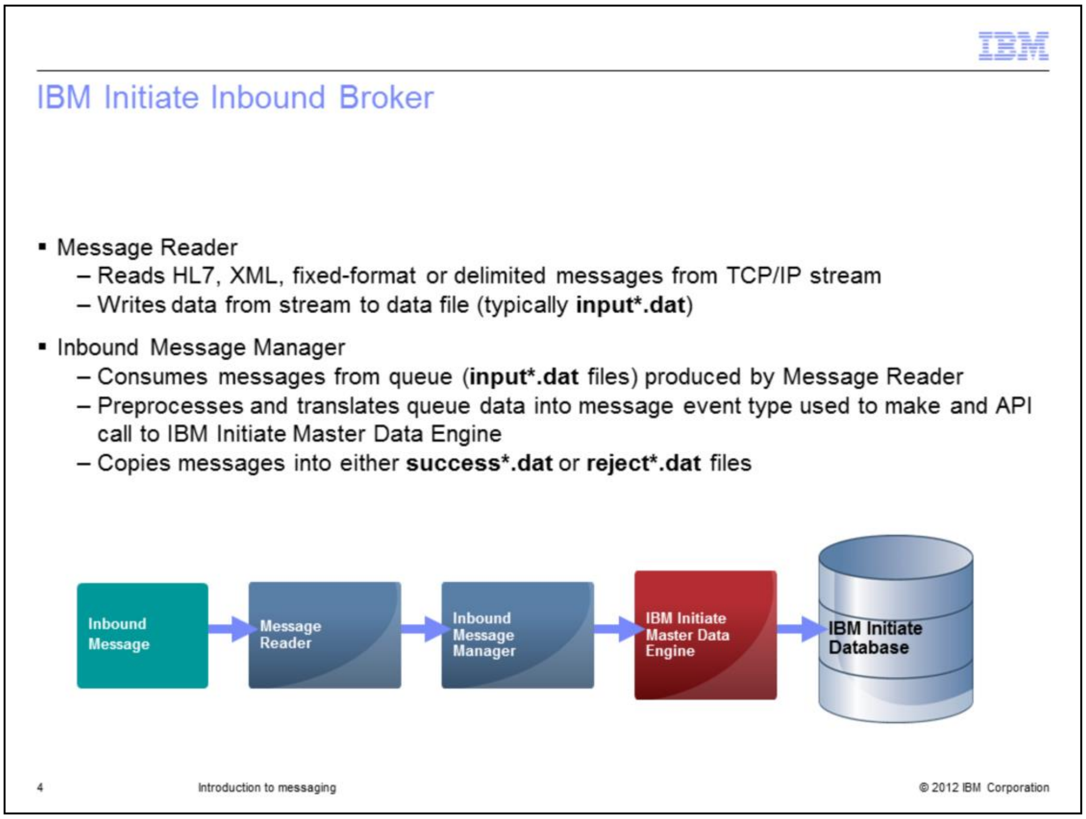
3 Introduction to messaging © 2012 IBM Corporation

There are five messaging applications to choose from in the IBM Initiate Master Data Service Message Broker Suite. There is the Inbound Broker, the Outbound Broker, the HL7 Query Adapter or Query Broker, the Mapping Message Manager or Mapping Broker, and the Routing Message Manager or Routing Broker.

The Inbound Broker and Outbound Broker applications both have two separate processes that handle the message-based transactions. The Inbound Broker has a Message Reader service and an Inbound Message Manager service.

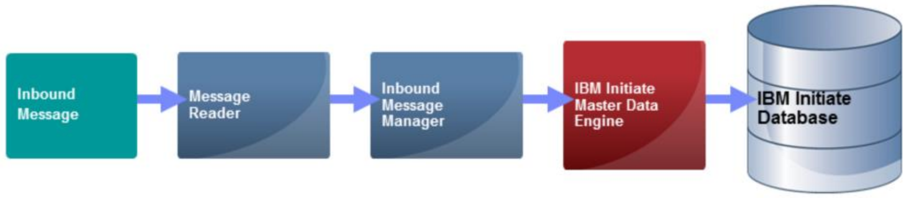
The Outbound Broker consists of an outbound message manager process and a message sender process.

There are more details on these broker applications later in this presentation.



## IBM Initiate Inbound Broker

- Message Reader
  - Reads HL7, XML, fixed-format or delimited messages from TCP/IP stream
  - Writes data from stream to data file (typically **input*.dat**)
- Inbound Message Manager
  - Consumes messages from queue (**input*.dat** files) produced by Message Reader
  - Preprocesses and translates queue data into message event type used to make and API call to IBM Initiate Master Data Engine
  - Copies messages into either **success*.dat** or **reject*.dat** files



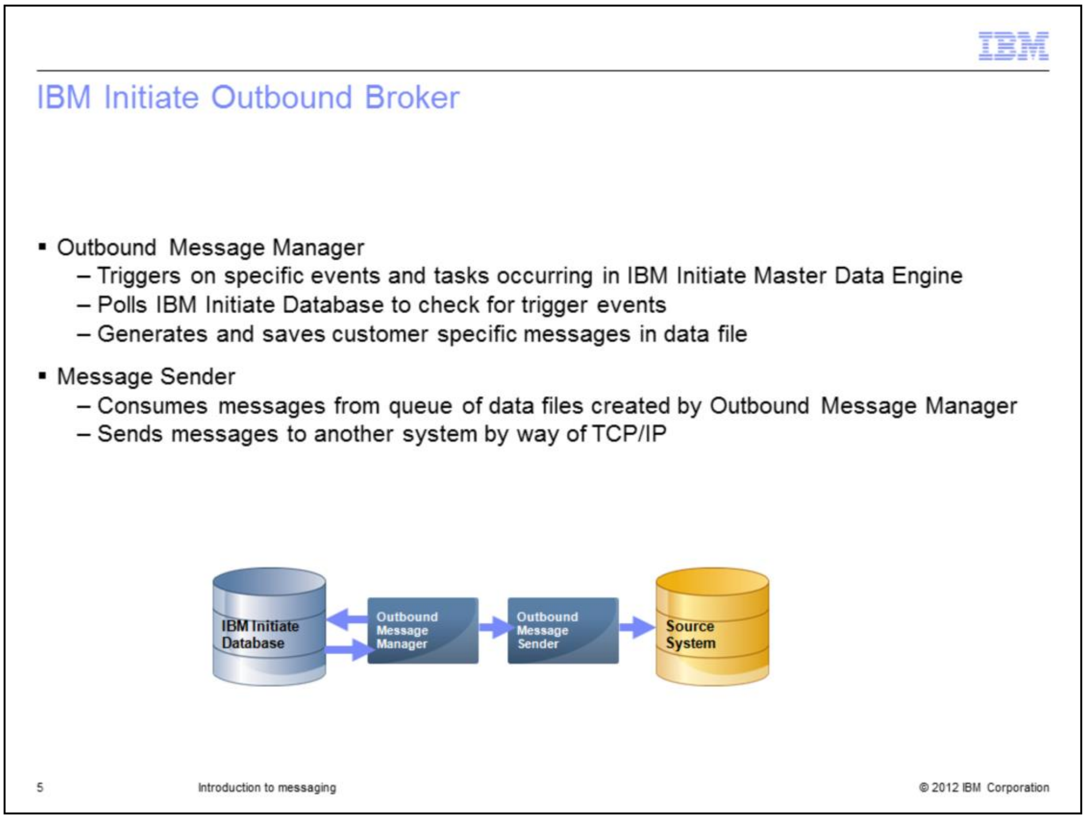
4 Introduction to messaging © 2012 IBM Corporation

As just previously mentioned, the Inbound Broker has a Message Reader and an Inbound Message Manager component. There are some important points about each of these components.

The message flow diagram displayed at the bottom of this slide, depicts an inbound message as it flows into and is received by the Message Reader. The Message Reader is configured to listen for inbound messages on a TCP/IP stream. It can be configured to accept HL7, XML, fixed-format, and delimited message types. It stores the inbound messages in data files. These data files are treated as a queue and the files are typically named “input .dat”.

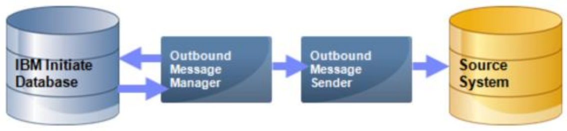
Next, the Inbound Message Manager uses the messages that were stored in the input queue by the Message Reader. It processes and modifies these messages into a message event interaction and then uses this to make the appropriate API call to send the message to the IBM Initiate Master Data Engine. The message is then copied by the Inbound Message Manager into a success or reject file queue based on the positive or negative response from the engine.

Depending on the message event interaction, the IBM Initiate Master Data Engine makes the appropriate changes in the IBM Initiate database. For a list of the possible message event types, see the documentation available through the IBM Support Portal.



## IBM Initiate Outbound Broker

- Outbound Message Manager
  - Triggers on specific events and tasks occurring in IBM Initiate Master Data Engine
  - Polls IBM Initiate Database to check for trigger events
  - Generates and saves customer specific messages in data file
- Message Sender
  - Consumes messages from queue of data files created by Outbound Message Manager
  - Sends messages to another system by way of TCP/IP



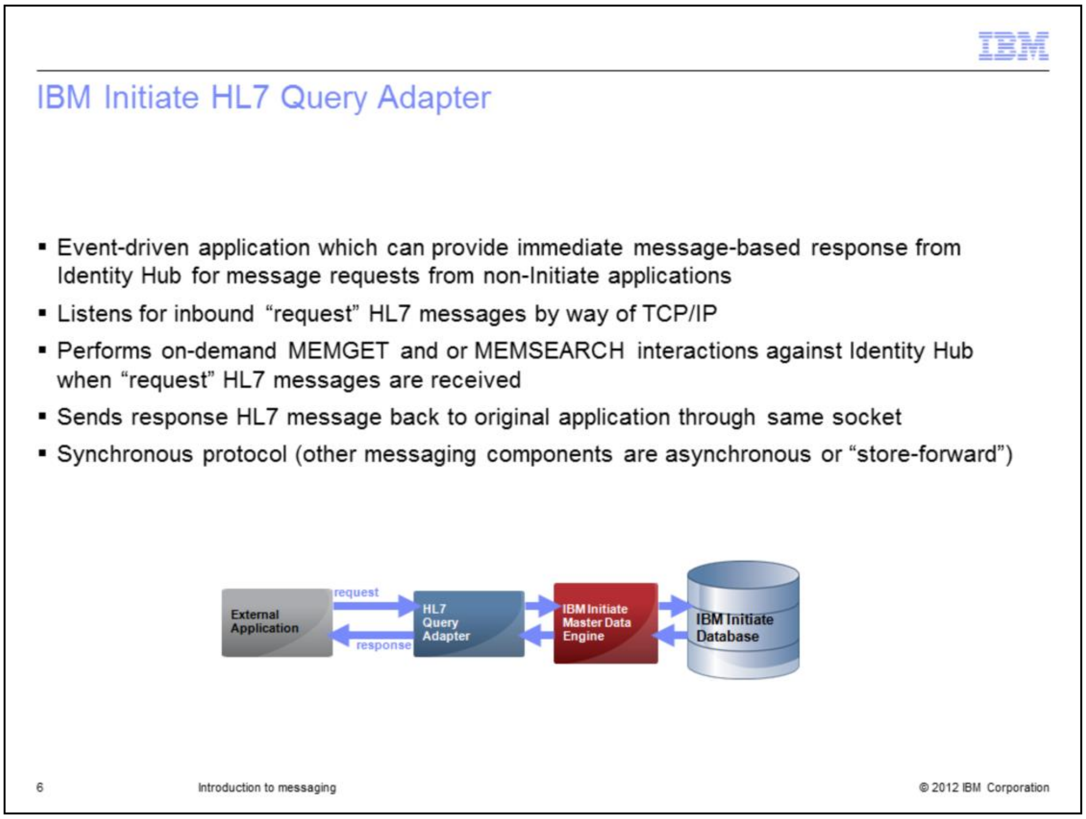
5 Introduction to messaging © 2012 IBM Corporation

On the outbound side, there are two Outbound Broker components; the Outbound Message Manager and the Message Sender.

In the message flow diagram displayed on this slide, you see the initial component is the Outbound Message Manager. The left arrow represents the periodic polls to the IBM Initiate Master Data Engine Database. If new events are detected, the data is sent to the Message Manager to be processed. The trigger events for the Outbound Message Manager are: “Has Shadow”, “PreMerge”, and an “EID Add” or “EID Update”.

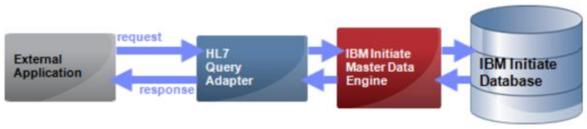
When a trigger event is found, the Outbound Message Manager builds a customer-specific message. The construction of the message is based on the broker configuration file. The message is written in data files placed in an outbound queue.

Next, the Outbound Message Sender picks up the messages from the outbound queue and sends them to an external source system by way of a TCP/IP connection.



## IBM Initiate HL7 Query Adapter

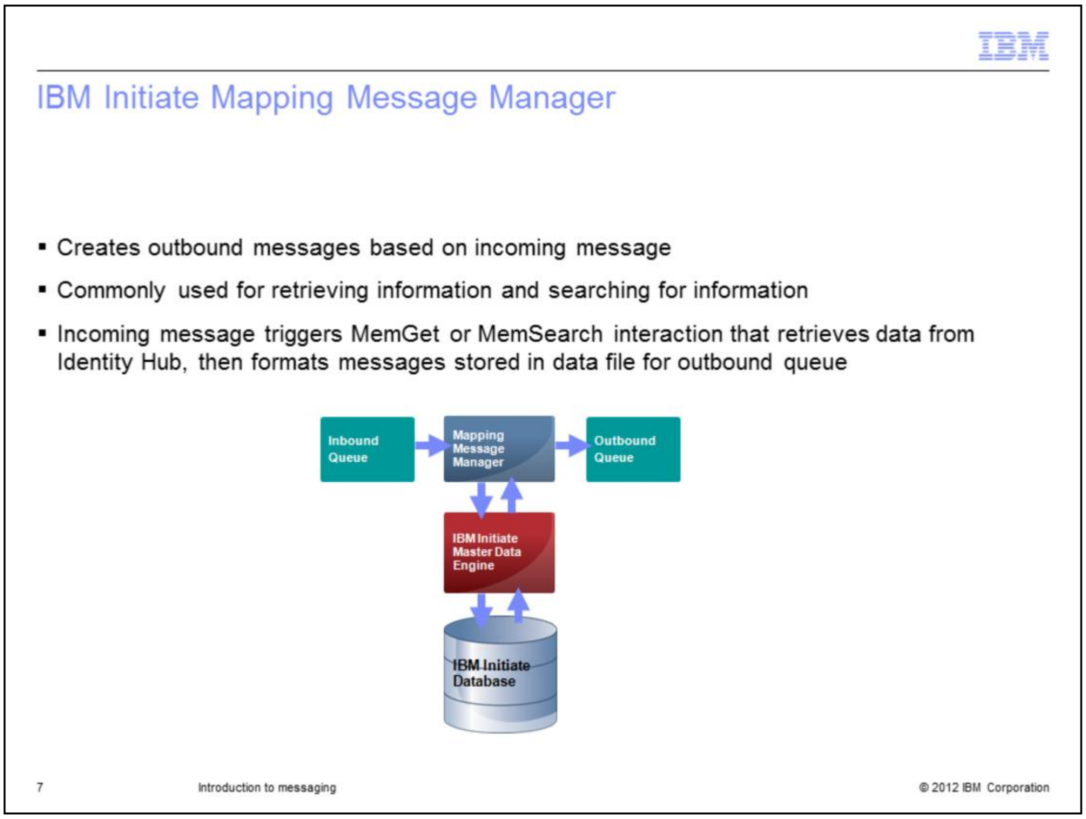
- Event-driven application which can provide immediate message-based response from Identity Hub for message requests from non-Initiate applications
- Listens for inbound “request” HL7 messages by way of TCP/IP
- Performs on-demand MEMGET and or MEMSEARCH interactions against Identity Hub when “request” HL7 messages are received
- Sends response HL7 message back to original application through same socket
- Synchronous protocol (other messaging components are asynchronous or “store-forward”)



HL7 stands for “Health Level Seven” and is a standard messaging protocol primarily used in the healthcare industry. The HL7 Query Adapter, also called the “Query Broker”, is a special read-only message application typically used by healthcare customers. It was developed to provide a way for customer applications to query the IBM Initiate system by way of HL7 messages. The HL7 Query Adapter listens for inbound HL7 message requests being sent to a specific port by way of a TCP/IP connection.

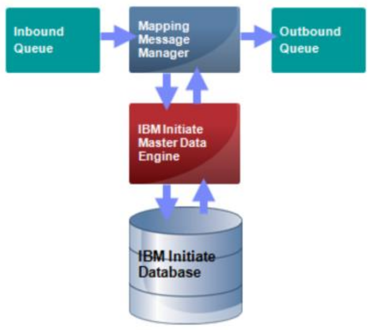
The message flow diagram displayed on this slide begins with the HL7 Query Adapter receiving an HL7 message request. Since this application is read-only, the request is either a MemGet or MemSearch interaction. The interaction request is then sent to the IBM Initiate Master Data Engine. The requested data is then retrieved by the engine and synchronously sent back to the HL7 Query Adapter. Then, the information is formatted by the HL7 Query Adapter and sent back to the originating system as a HL7 response message on the same socket connection the request message was received from.

It is important to note that the round trip interaction is performed on the same socket connection.



## IBM Initiate Mapping Message Manager

- Creates outbound messages based on incoming message
- Commonly used for retrieving information and searching for information
- Incoming message triggers MemGet or MemSearch interaction that retrieves data from Identity Hub, then formats messages stored in data file for outbound queue



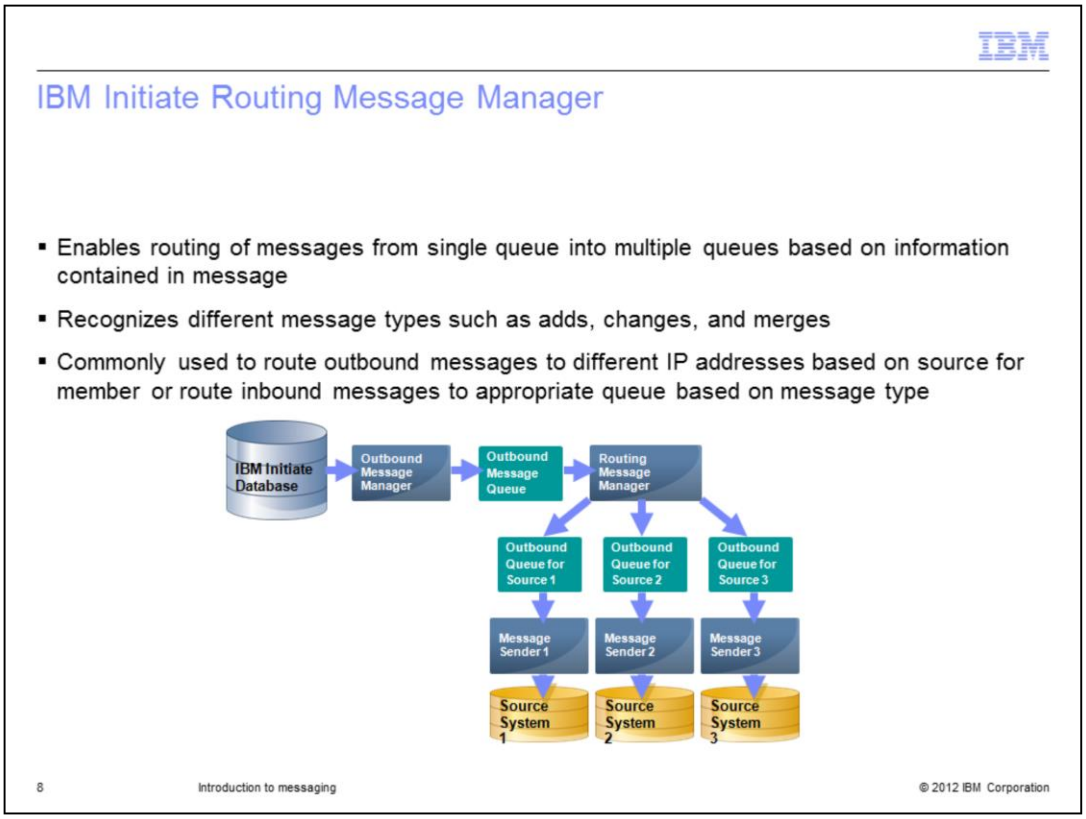
7 Introduction to messaging © 2012 IBM Corporation

The next broker application in line is the Mapping Message Manager which is also referred to as the “Mapping Broker”.

The Mapping Broker reads messages from an input queue, requests data from the hub, and then writes to an outbound message queue. A Mapping Broker instance is used for two basic purposes: retrieving information and searching for information. This information is then added to the incoming message and propagated to a downstream source system by way of the Outbound Broker.

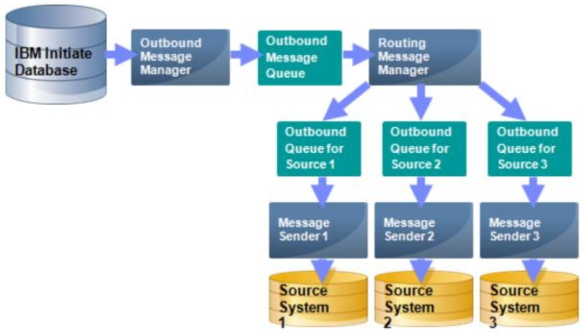
An example of a practical application of the Mapping Broker is when a Mapping Broker is used to examine the inbound input files sent in from one source system for address updates. The Mapping Broker then forwards the address update to a downstream source system. The Mapping Broker is triggered by the inbound message queue. When you install a Mapping Broker, you normally do not just use that application by itself. Typically, you have an Inbound Message Manager that is also processing the same inbound messages in parallel and sending them to the IBM Initiate Master Data Engine. Each time a message comes in, you can trigger the Mapping Message Manager to pull data from the IBM Initiate Master Data Engine using a MemGet or MemSearch interaction. Then, write this data from the engine into a data file for the outbound queue.

Basically, if you want to send messages to another source system based on an inbound message from another system, the Mapping Message Manager may be the solution you are looking for.



## IBM Initiate Routing Message Manager

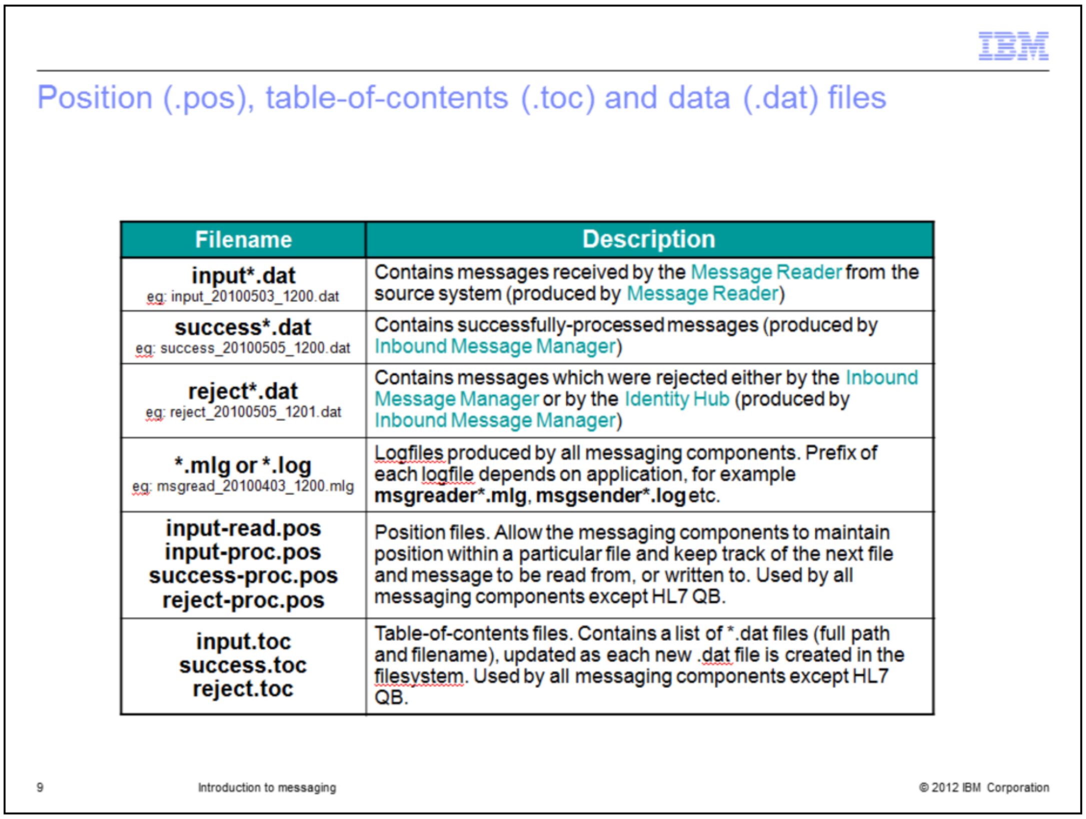
- Enables routing of messages from single queue into multiple queues based on information contained in message
- Recognizes different message types such as adds, changes, and merges
- Commonly used to route outbound messages to different IP addresses based on source for member or route inbound messages to appropriate queue based on message type



8 Introduction to messaging © 2012 IBM Corporation

The Routing Message Manager, also called the “Routing Broker”, allows you to take messages from one queue and duplicate them into other queues based on the information within the message and it’s configuration file.

An example of how the Routing Broker is used is when you need to send an outbound message to multiple source systems based on the source of the records being updated. This example is used to illustrate the message flow diagram displayed on this slide. The Routing Message Manager takes messages out of the outbound message queue and then, based on it’s configuration, copies this message to the appropriate outbound message queue or queues. Then, each of the outbound queues uses their own message sender to send the messages to their source system.



## Position (.pos), table-of-contents (.toc) and data (.dat) files

| Filename | Description |
|---|---|
| **input*.dat** eg: input_20100503_1200.dat | Contains messages received by the Message Reader from the source system (produced by Message Reader) |
| **success*.dat** eg: success_20100505_1200.dat | Contains successfully-processed messages (produced by Inbound Message Manager) |
| **reject*.dat** eg: reject_20100505_1201.dat | Contains messages which were rejected either by the Inbound Message Manager or by the Identity Hub (produced by Inbound Message Manager) |
| ***.mlg or *.log** eg: msgread_20100403_1200.mlg | Logfiles produced by all messaging components. Prefix of each logfile depends on application, for example **msgreader*.mlg**, **msgsender*.log** etc. |
| **input-read.pos** **input-proc.pos** **success-proc.pos** **reject-proc.pos** | Position files. Allow the messaging components to maintain position within a particular file and keep track of the next file and message to be read from, or written to. Used by all messaging components except HL7 QB. |
| **input.toc** **success.toc** **reject.toc** | Table-of-contents files. Contains a list of *.dat files (full path and filename), updated as each new .dat file is created in the filesystem. Used by all messaging components except HL7 QB. |

9 Introduction to messaging © 2012 IBM Corporation

Now the focus will shift to talking about the mechanism behind the inbound and outbound message process. The next slides describe the file naming conventions, purposes of the files, and how the position and table of contents files are used. The table displayed on this slide, lists the file naming conventions used for the default broker configurations and a brief description of these files.

The first three files, “input*.dat”, “success*.dat”, and “reject*.dat” are all data files. The asterisk is a place holder for the date-time stamp of when the file was created in a real system.

The input .dat file is basically the input queue. This file contains messages that were received by the Message Reader from the source system.

The success .dat contains the messages that were copied from the input .dat file after being successfully processed by the Inbound Broker. Alternatively, the reject .dat file contains messages that were rejected by the Inbound Broker. It is important to note that the messages in all the .dat files are exactly the same as they were received by the Message Reader.

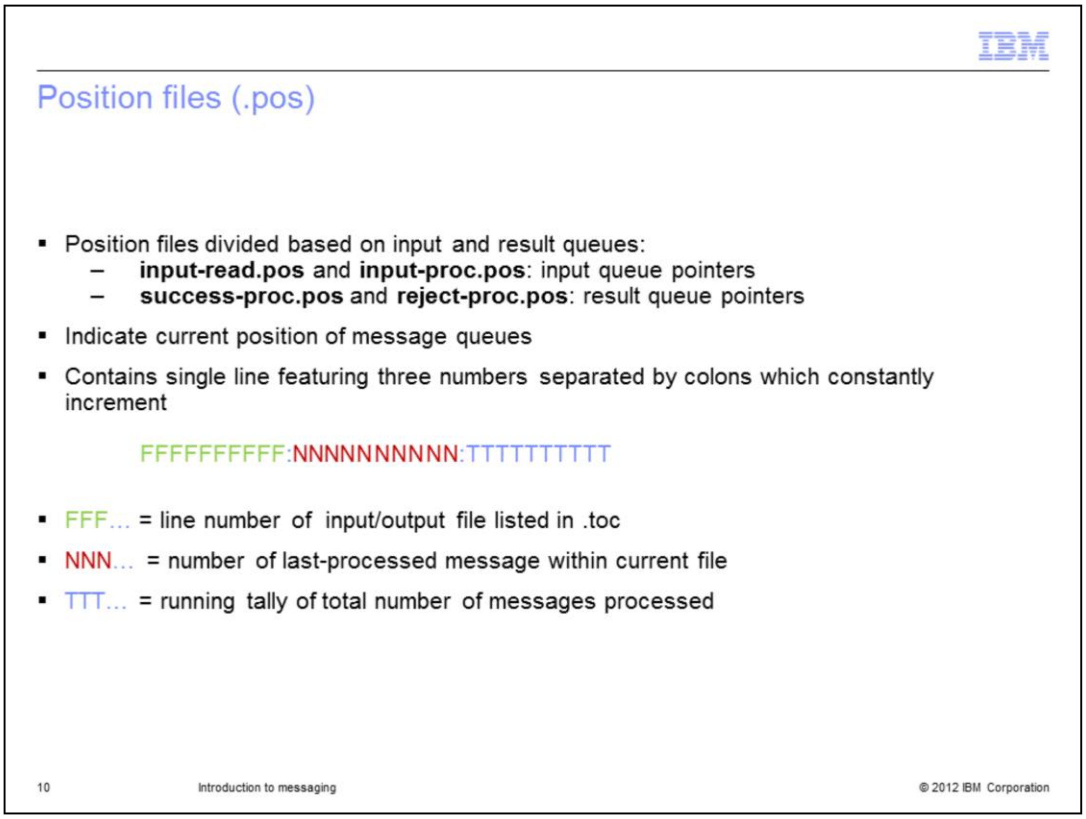
Ideally, you do not want to have any rejected messages. However, if you do find messages being rejected, look at the Broker log files to determine why a message was rejected. If there is no detail of any error related to the rejection in the broker log file, look at the engine log file as the message can be rejected by the Inbound Broker or the Engine.

The .mlg or .log files are created by every component. The first part of the file name corresponds to the type of component and date-time stamp of when that file was created.

The next files are the position files. All of the components use position files except HL7 Query Adapter. The position files keep track of what file and line position in the file an individual component was last working on. This allows the broker components to stay synchronized.

The last files in this table, are the input.toc, success.toc, and reject.toc. These are the table of contents files. These files contain a list of directories that point to the .dat file they represent. They are updated each time a new .dat file is created. Just like the position files, the HL7 Query Adapter is the only messaging component that does not use a table of content file.

Now that you are familiar with the different file types used by the brokers, you are now ready to jump into how they are used.



## Position files (.pos)

- Position files divided based on input and result queues:
  - **input-read.pos** and **input-proc.pos**: input queue pointers
  - **success-proc.pos** and **reject-proc.pos**: result queue pointers
- Indicate current position of message queues
- Contains single line featuring three numbers separated by colons which constantly increment

  FFFFFFFFFF:NNNNNNNNNN:TTTTTTTTTT

- FFF… = line number of input/output file listed in .toc
- NNN… = number of last-processed message within current file
- TTT… = running tally of total number of messages processed

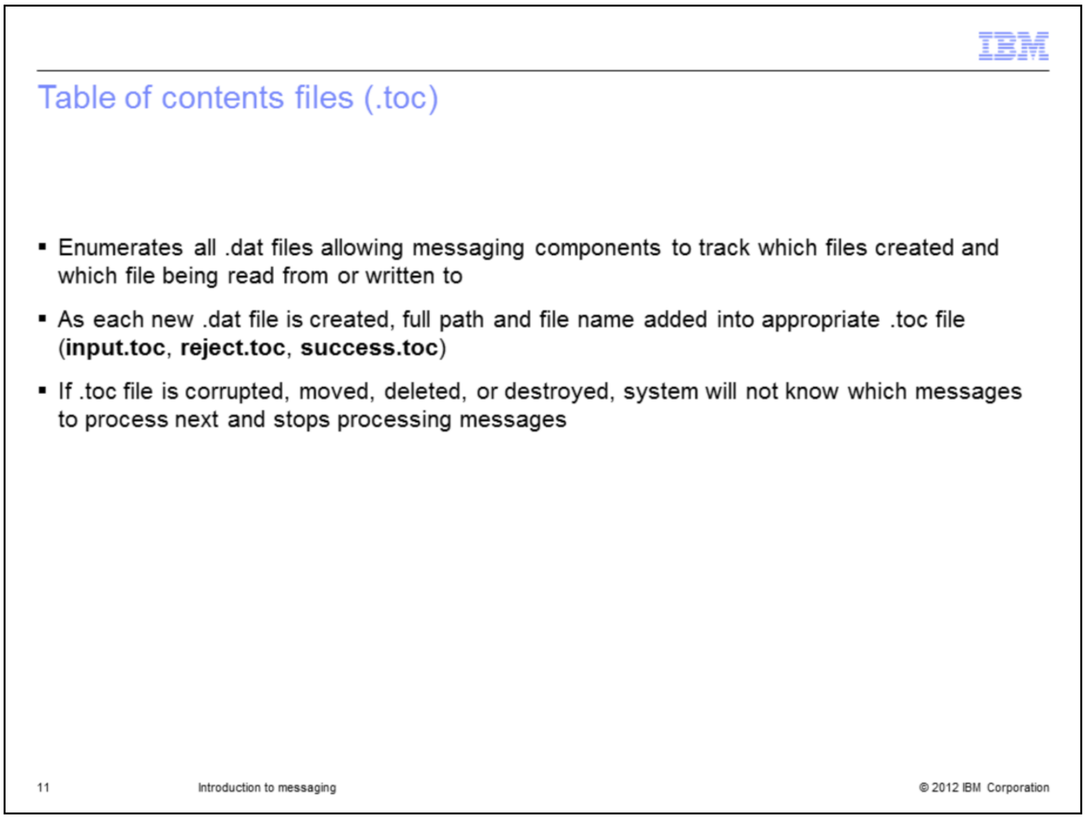
10 Introduction to messaging © 2012 IBM Corporation

The next five slides illustrate how messages are processed using the files mentioned in the previous slide. The first files examined are the position files or ".pos" files. The position files have two sets of files; one set operates on the input queue and the other is used for the result queue. The input-read.pos and input-proc.pos are used for tracking the input queue position. The success-proc.pos and reject-proc.pos files are used for the result queues. Each file is composed of a single line with three numbers separated by two colons.

The first numeric is a count of files and is represented by a string of "F"s. Specifically, it is the line number in the .toc file, which in turn points to the .dat file. Essentially, it is the file it was last working on.

The second numeric is a count of messages and is represented by the string of "N"s. This is the last message count in the file "F" that was processed within the current .dat file.

The last numeric is a running total of messages processed and is represented by the string of "T"s.



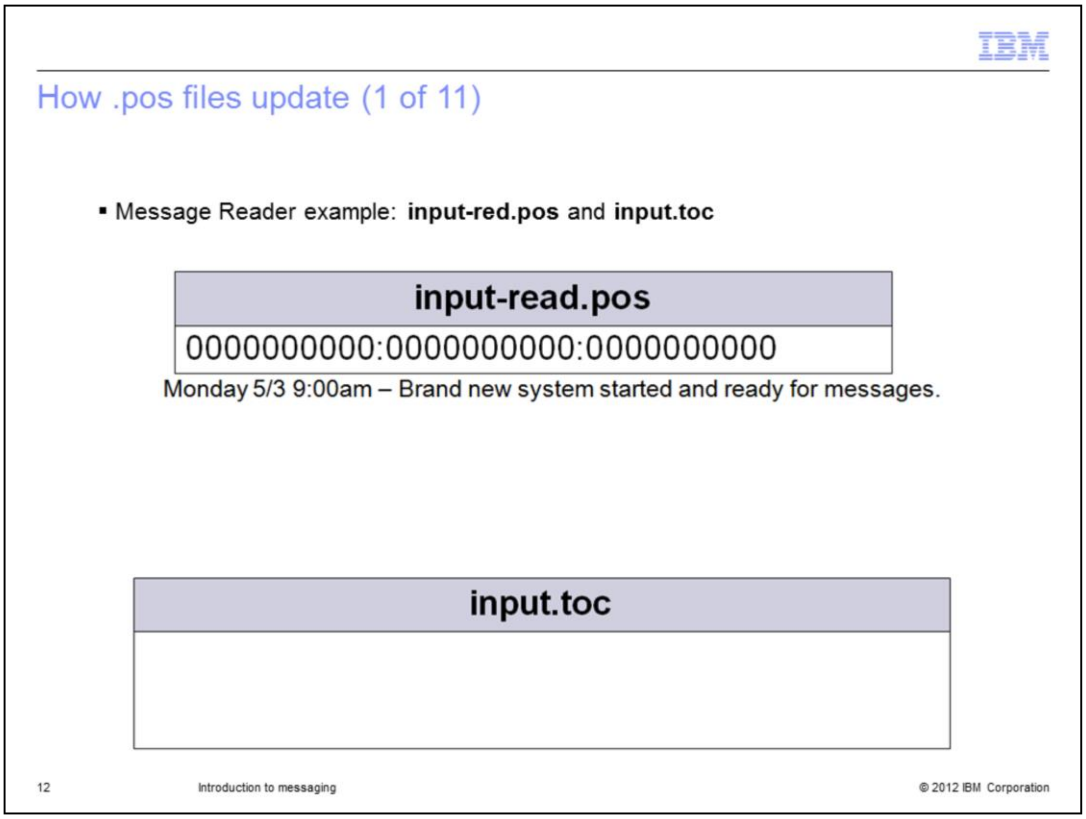
## Table of contents files (.toc)

- Enumerates all .dat files allowing messaging components to track which files created and which file being read from or written to
- As each new .dat file is created, full path and file name added into appropriate .toc file (**input.toc**, **reject.toc**, **success.toc**)
- If .toc file is corrupted, moved, deleted, or destroyed, system will not know which messages to process next and stops processing messages

11 Introduction to messaging © 2012 IBM Corporation

The table of contents file, or .toc file, enumerates all the .dat files for an input or output queue. With the help of the .pos file it keeps track of which files have already been created and processed, and which .dat files are currently being written to. When a new .dat file is created, its full path and file name is added to the end of the appropriate .toc file.

Be careful with the .pos and .toc files. If the files become corrupted, moved, deleted, or destroyed, the message component will not know which message to process next and will stop processing messages altogether.



## How .pos files update (1 of 11)

- Message Reader example: **input-red.pos** and **input.toc**

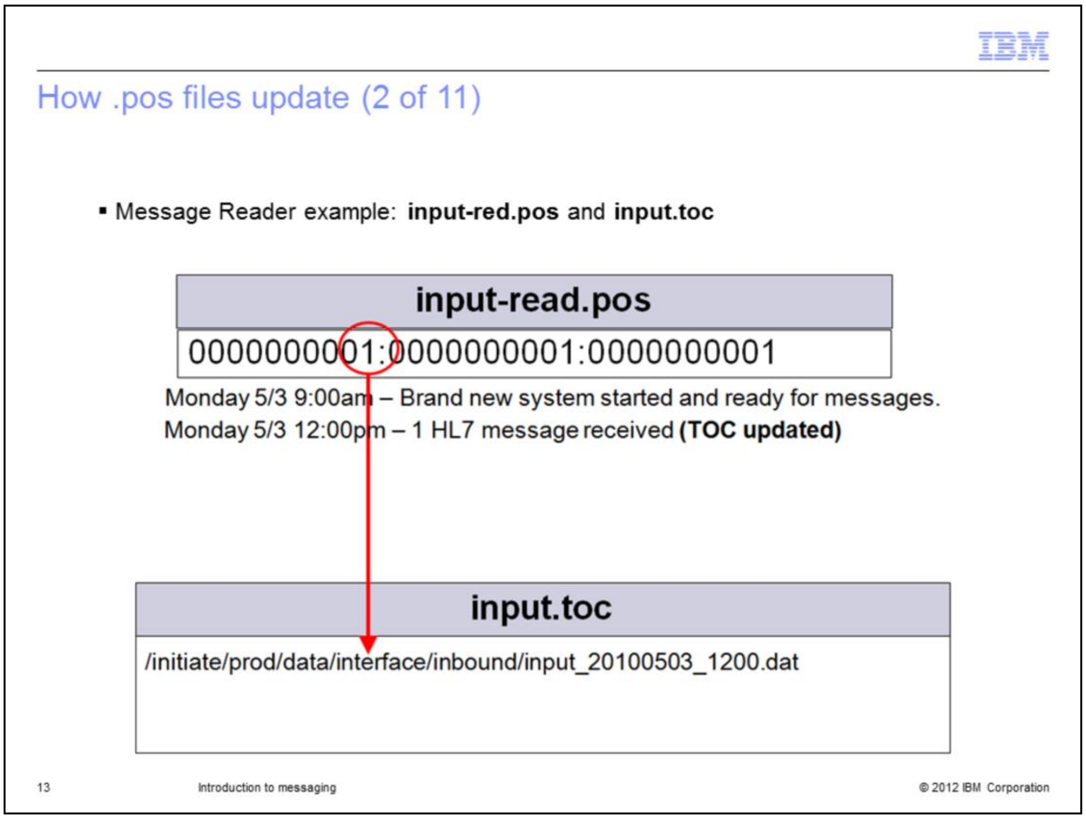
| input-read.pos |
|---|
| 0000000000:0000000000:0000000000 |

Monday 5/3 9:00am – Brand new system started and ready for messages.

| input.toc |
|---|
| |

12 Introduction to messaging © 2012 IBM Corporation

The next example goes though, step by step, how a message reader updates the .pos and .toc files. This is a typical inbound example. There is currently no .toc or .pos file and the message reader is turned on at 9:00 AM on Monday May 3rd for the very first time. The input.toc file and the input-read.pos files are created. The .toc file is initially blank and the .pos contains all zeros.



## How .pos files update (2 of 11)

- Message Reader example: **input-red.pos** and **input.toc**

**input-read.pos**

0000000001:0000000001:0000000001

Monday 5/3 9:00am – Brand new system started and ready for messages.

Monday 5/3 12:00pm – 1 HL7 message received **(TOC updated)**

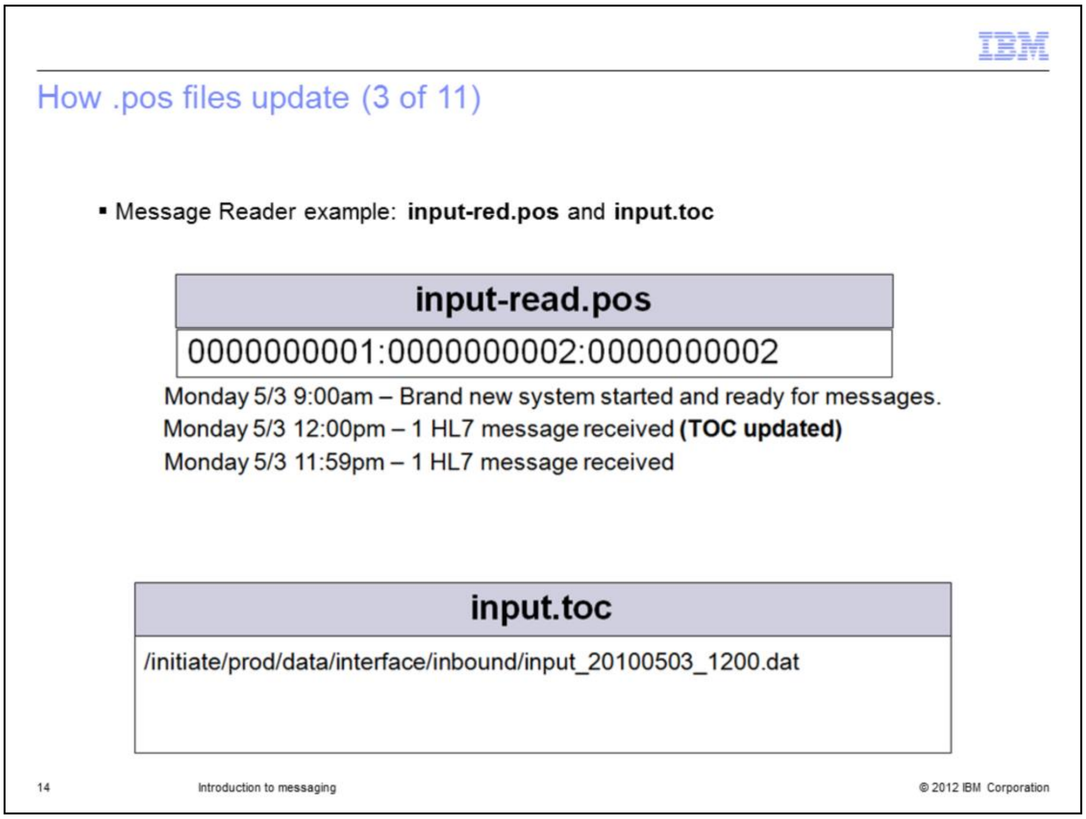
**input.toc**

/initiate/prod/data/interface/inbound/input_20100503_1200.dat

13 Introduction to messaging © 2012 IBM Corporation

The first HL7 formatted message is received on Monday May 3rd at 12:00 PM. The message reader now creates the input .dat file and adds the message to the end of the file.

The path to the input .dat file is then added to the end of the .toc file. Within the input-read.pos file, the first number is incremented from a zero to a one in reference to the position of the .dat file in the .toc. Next, the second number, in the .pos file, is incremented from a zero to a one in reference to the message number in the .dat file. The last number is incremented from a zero to a one, since one message has been received.



## How .pos files update (3 of 11)

- Message Reader example: **input-red.pos** and **input.toc**

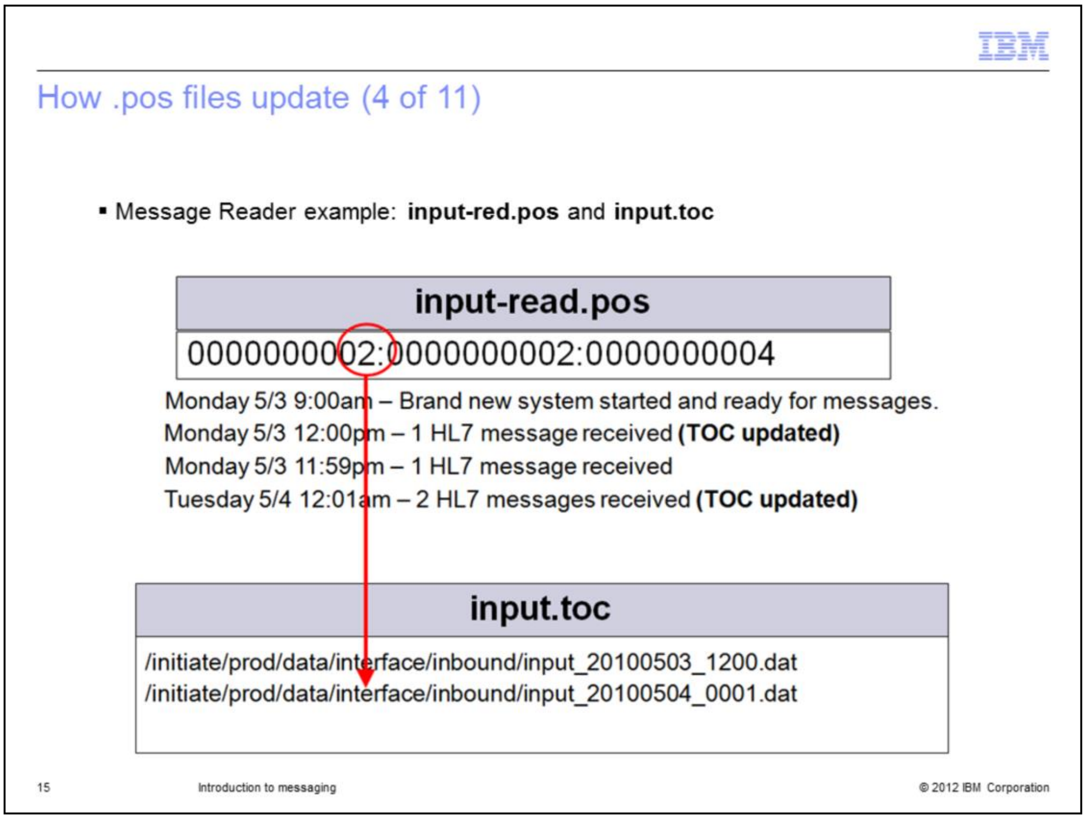
| input-read.pos |
|---|
| 0000000001:0000000002:0000000002 |

Monday 5/3 9:00am – Brand new system started and ready for messages.
Monday 5/3 12:00pm – 1 HL7 message received **(TOC updated)**
Monday 5/3 11:59pm – 1 HL7 message received

| input.toc |
|---|
| /initiate/prod/data/interface/inbound/input_20100503_1200.dat |

14 Introduction to messaging © 2012 IBM Corporation

Another HL7 formatted message is received on Monday May 3rd at 11:59 PM just before the end of the day. Since it is the same day since the last time the message reader created an input .dat file, it does not create a new .dat file. Since there was no new .dat file created, there is no need to update the input.toc file or the first number in the input-read.pos file. The message is added to the end of the .dat file and the second number in the .pos file is incremented from a one to a two indicating that there are two messages in the .dat file. The last number in the .pos file is also incremented to maintain a running total.



## How .pos files update (4 of 11)

- Message Reader example: **input-red.pos** and **input.toc**

**input-read.pos**

0000000002:0000000002:0000000004

Monday 5/3 9:00am – Brand new system started and ready for messages.
Monday 5/3 12:00pm – 1 HL7 message received **(TOC updated)**
Monday 5/3 11:59pm – 1 HL7 message received
Tuesday 5/4 12:01am – 2 HL7 messages received **(TOC updated)**

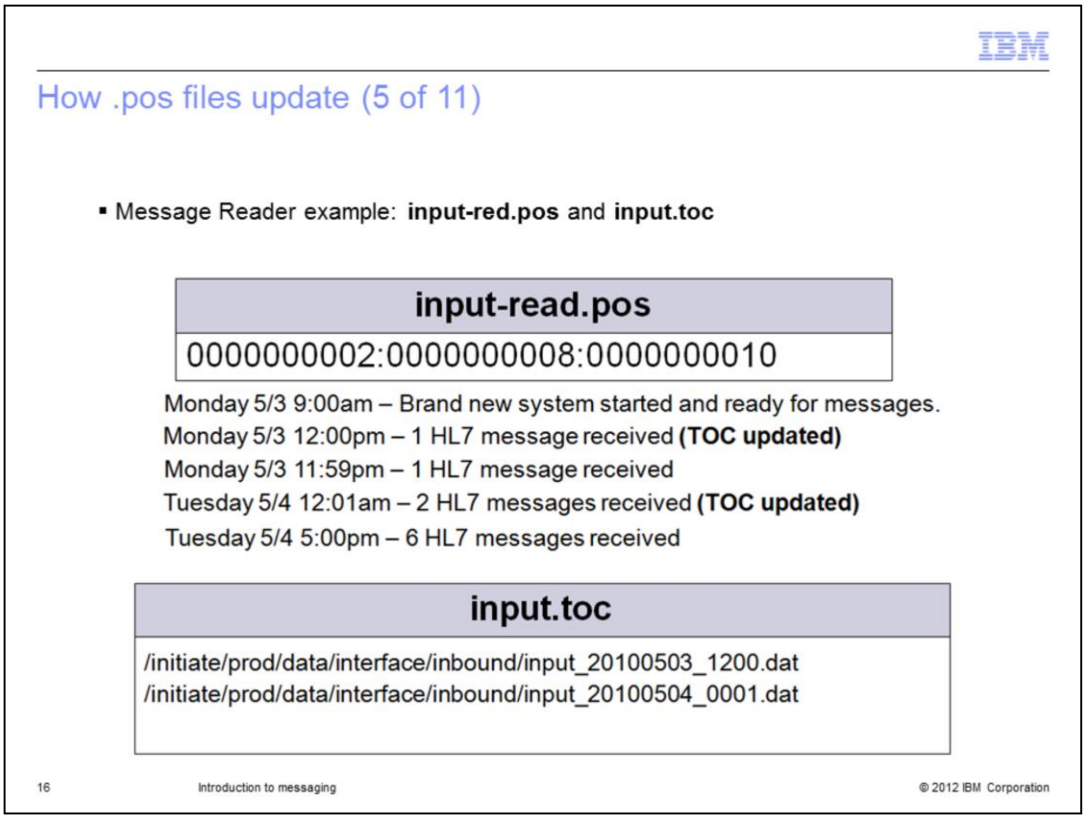
**input.toc**

/initiate/prod/data/interface/inbound/input_20100503_1200.dat
/initiate/prod/data/interface/inbound/input_20100504_0001.dat

15 Introduction to messaging © 2012 IBM Corporation

Two minutes after the last message was received two more HL7 messages are received. It is now a new day, Tuesday May 4th, and since it is a new day, a new input .dat file is created.

Because of the new .dat file creation, the path to the new file is added to the end of the input.toc file, the first numeric in the .pos file is incremented from a one to a two and the second number is reset back to zeros. The two messages are added to the end of the new .dat file, and the second numeric in the .pos is incremented by two. The last number in the .pos file is incremented by two (changes from a two to a four).


## How .pos files update (5 of 11)

- Message Reader example: **input-red.pos** and **input.toc**

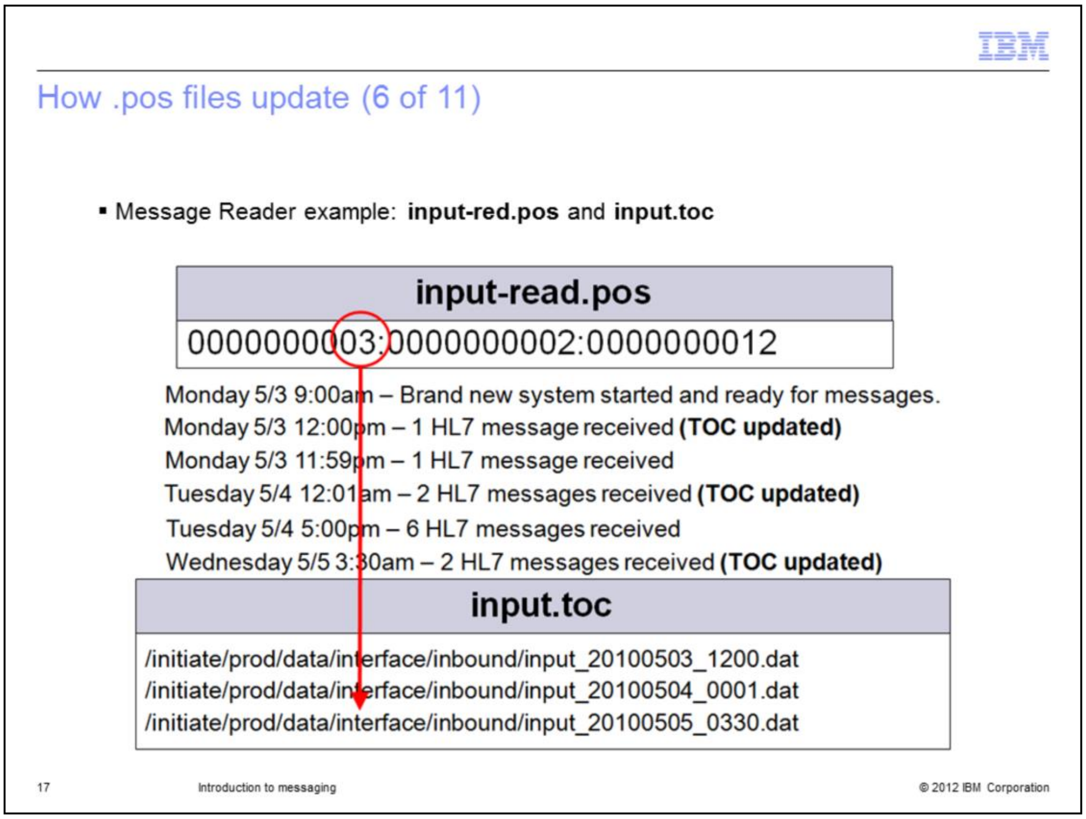
| **input-read.pos** |
|---|
| 0000000002:0000000008:0000000010 |

Monday 5/3 9:00am – Brand new system started and ready for messages.
Monday 5/3 12:00pm – 1 HL7 message received **(TOC updated)**
Monday 5/3 11:59pm – 1 HL7 message received
Tuesday 5/4 12:01am – 2 HL7 messages received **(TOC updated)**
Tuesday 5/4 5:00pm – 6 HL7 messages received

| **input.toc** |
|---|
| /initiate/prod/data/interface/inbound/input_20100503_1200.dat<br>/initiate/prod/data/interface/inbound/input_20100504_0001.dat |

16 Introduction to messaging © 2012 IBM Corporation

Later, on Tuesday May 4th, six more HL7 messages are received. It is the same day so no .dat file is created and no change is made to the first value in the .pos file. The six messages are added to the end of the existing .dat file from May 4th. The six is added to the second number in the .pos file making it eight. The last number in the .pos file is incremented by six and is now at ten.



## How .pos files update (6 of 11)

- Message Reader example: **input-red.pos** and **input.toc**

**input-read.pos**

0000000003:0000000002:0000000012

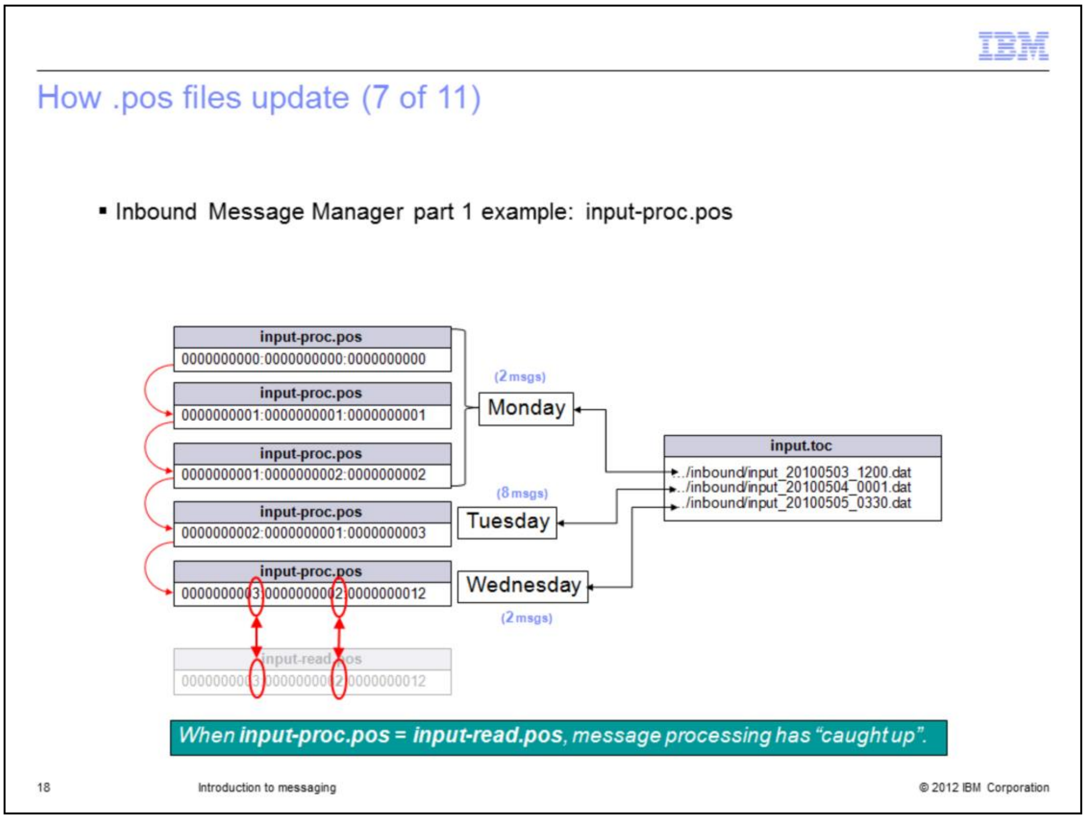
Monday 5/3 9:00am – Brand new system started and ready for messages.
Monday 5/3 12:00pm – 1 HL7 message received **(TOC updated)**
Monday 5/3 11:59pm – 1 HL7 message received
Tuesday 5/4 12:01am – 2 HL7 messages received **(TOC updated)**
Tuesday 5/4 5:00pm – 6 HL7 messages received
Wednesday 5/5 3:30am – 2 HL7 messages received **(TOC updated)**

**input.toc**

```
/initiate/prod/data/interface/inbound/input_20100503_1200.dat
/initiate/prod/data/interface/inbound/input_20100504_0001.dat
/initiate/prod/data/interface/inbound/input_20100505_0330.dat
```

17 Introduction to messaging © 2012 IBM Corporation

The next two HL7 messages are received on Wednesday, May 5th. Since it is a new day, a new .dat file is created. The two messages are added to the end of the .dat file and the second position in the .pos file is reset back to a zeros then incremented by two. A new .dat file was created so the path to the new file is added to the end of the .toc file and the first number in the .pos file is incremented by one to a three. The last number in the .pos file is incremented by two and is now 12.



## How .pos files update (7 of 11)

- Inbound Message Manager part 1 example: input-proc.pos

When ***input-proc.pos*** = ***input-read.pos***, *message processing has "caught up".*

18 Introduction to messaging © 2012 IBM Corporation

On this slide, you see how the Inbound Broker processes the inbound messages. On the next slide, you see the other part of the Inbound Broker where it deals with successful and rejected messages.

This will hopefully make it easier for you to understand how the messages in the Message Reader are consumed by the Inbound Broker and keeps this separate from the part that determines if a message is successful or not.

The first thing to do is turn on the Inbound Broker. Since it is going to process the files that were used as in the example of the Message Reader on the previous slide, the Inbound Broker is going to have to catch up. Look at the input.toc file and you see there are three .dat files in the .toc. If you open the first .dat file, you see it has two messages. The second .dat file has eight messages and the last .dat file has two messages.

Now when the Inbound Broker begins processing, it reads the first message from the first .dat file in the .toc. It then creates the input-proc.pos file and increments the three counters. The numbers are 1, 1, and 1. So this is the first .dat file in the .toc file being read, the first message in that .dat file, and, the first message processed by the Message Manager.

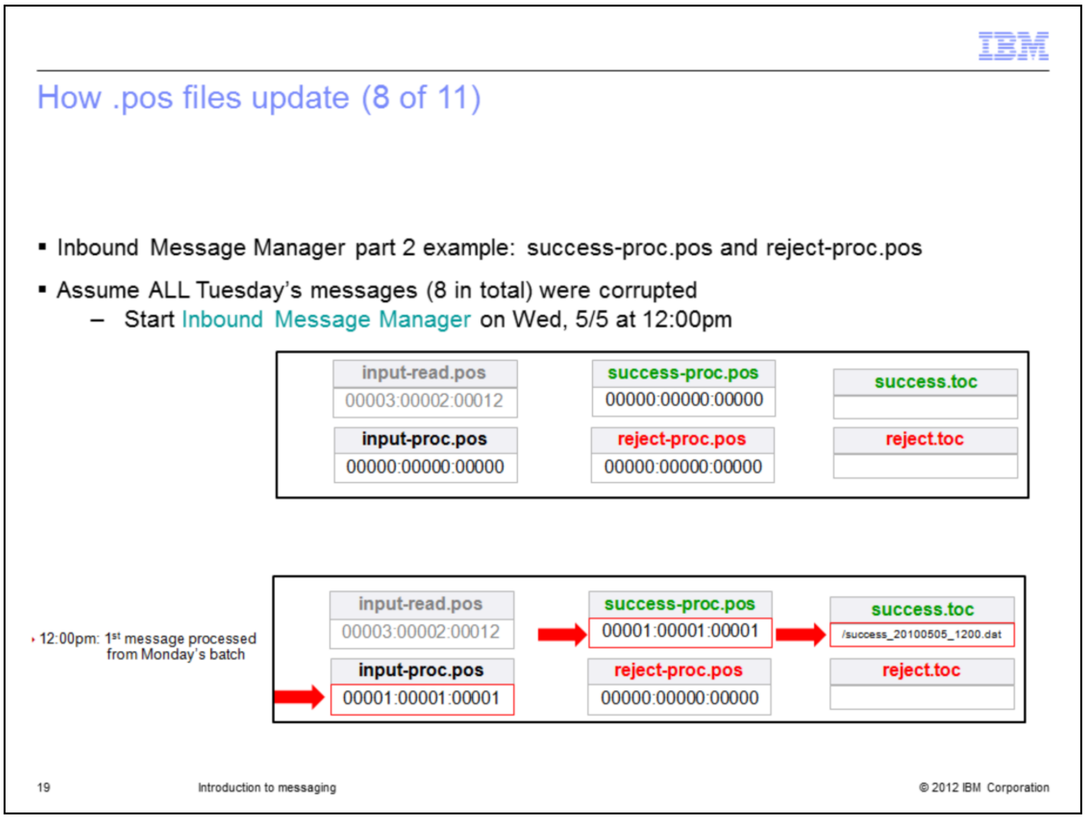
Next, the second message from the .dat file created on Monday May 3rd is processed. The new .pos file becomes 1, 2, and 2. This is still the first .dat file in the .toc. It is the second message in the .dat file and the second message to be processed.

The "end of file" is reached and there are no more files to process in the first .dat file so the Inbound Broker moves onto the second .dat file in the .toc. It then reads in the first message created on Tuesday May 4th from the .dat file. The .pos file is updated to 2, 1, and 3. This is the second .dat file in the .toc, the first message in the .dat file, and the 3rd message in total to be processed. The Inbound Broker then continues processing the .dat file just like it did the first .dat file. It keeps going until it reaches the "end of file" in the second .dat file.

Next, the last .dat file from Wednesday May 5th is read and the 2 messages are processed. The .pos file is now 3, 2, and 12. This is the 3rd .dat file in the .toc, the 2nd message in .dat file, and 12 message have been processed.

After the Inbound Broker has finished finish processing the last message in the .dat file you can see that the input-proc.pos file now matches the input-read.pos file. When the first two, left most, numbers of the input-proc.pos file equal the corresponding numbers in the input-read.pos file the Inbound Broker knows he has "caught up" to the Message Reader.

As you can see, the first two corresponding values in each file now match so you know all of the messages in the Message Reader have been processed by the Inbound Broker.



## How .pos files update (8 of 11)

- Inbound Message Manager part 2 example: success-proc.pos and reject-proc.pos
- Assume ALL Tuesday's messages (8 in total) were corrupted
  - Start Inbound Message Manager on Wed, 5/5 at 12:00pm

19 Introduction to messaging © 2012 IBM Corporation

This example is a continuation from the example used in part one of the previous slide. The next four slides describe the part of the Inbound Broker where it deals with successful and rejected messages. The "success" and "reject" queues work in much the same way the input queue does. They both have positioning (.pos) and table of contents (.toc) files.

In this example three assumptions are made. The first assumption is that all eight of the messages on Tuesday, May 4th were rejected. These messages may have been rejected for any number of reasons including missing a required data field like a social security number, but for this demonstration the reason is not important.

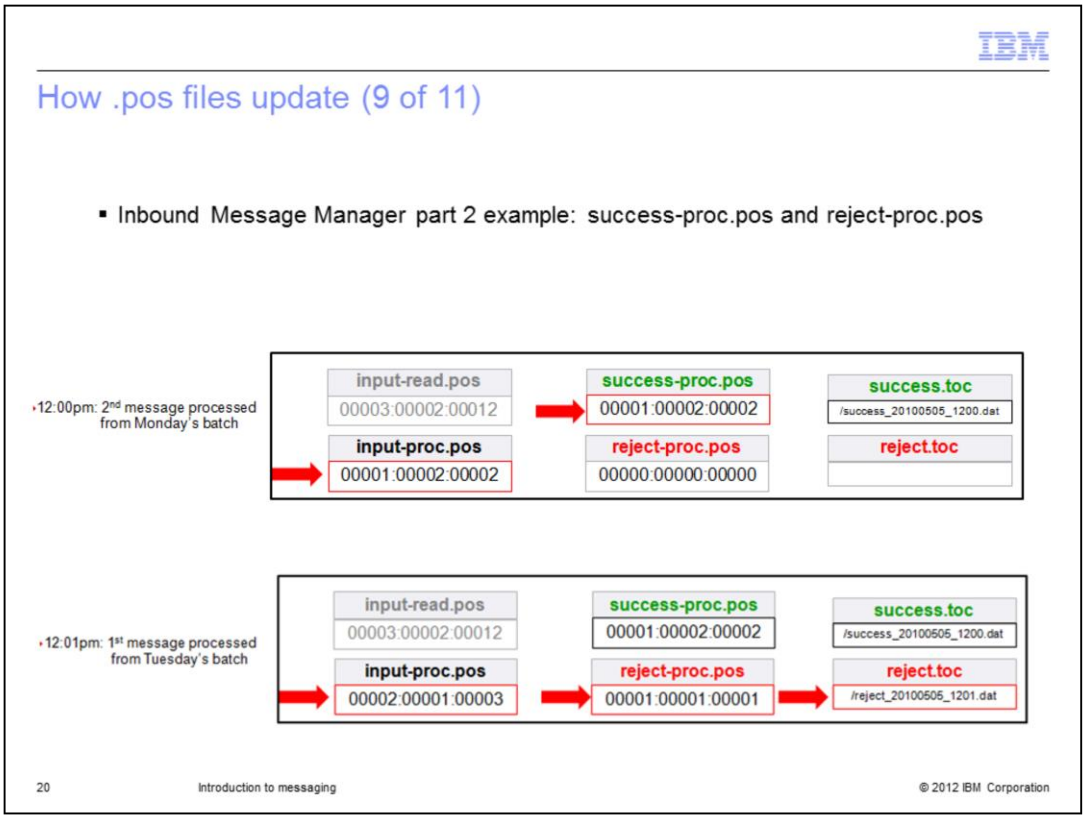
The second assumption is that the Inbound Broker is started at 12:00 PM on Wednesday, May 5th after all of the messages have been stored in the input queue by the Message Reader.

The third assumption is that this is the first time any messages have been processed by this Inbound Broker.

This example is started by processing the first message from Monday, May 3rd. The first action creates the input-proc.pos file and all three values are incremented as described in the previous slide to 1, 1, and 1. The message is determined to be successful at 12:00 PM. Therefore, the success-proc.pos, reject-proc.pos, success_20100505_1200.dat, success.toc, and reject.toc files are created.

Next, the same exact message that was read by the Message Reader is copied to the end of the success .dat file. The full path to the success .dat file is then appended to the end of the success.toc file and the values in the success-proc.pos are updated to 1, 1, and 1.

(Note the full path has been truncated in this example due to space constraints).



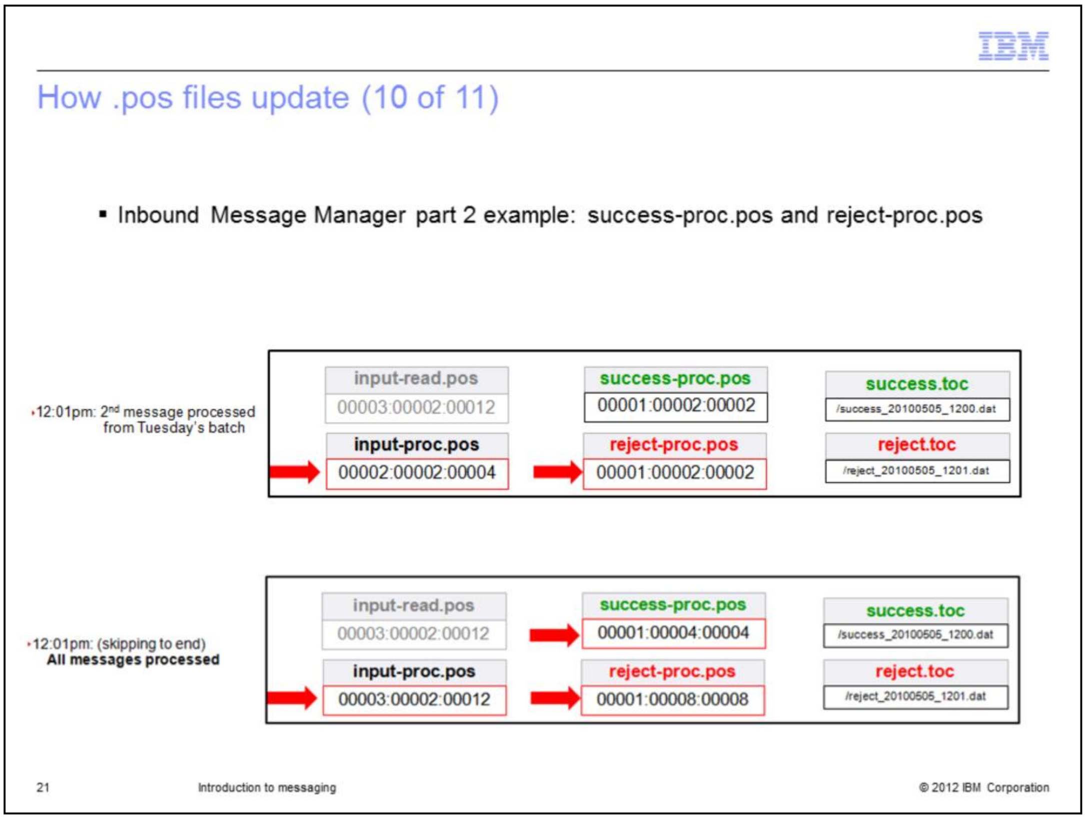
## How .pos files update (9 of 11)

- Inbound Message Manager part 2 example: success-proc.pos and reject-proc.pos

20 Introduction to messaging © 2012 IBM Corporation

The second message from Monday is now processed and is successful. The second and third numeric in the input-proc.pos are incremented by one and the new values are 1, 2, and 2. The message is copied to the end of the existing success .dat file and one is added to the last two numerics in the success-proc.pos file. The new values are 1, 2, and 2.

Now the time changes to 12:01 PM and the first message from Tuesday, May 4th is processed. The input-proc.pos file is updated to 2, 1, and 3. This message fails and is rejected. This causes the reject_20100505_1201.dat file to be created, this same message from input .dat file is copied into the reject .dat file. The path to the reject .dat file is added to the end of the reject.toc, and all three values in reject-proc.pos are incremented by one to 1, 1, and 1.

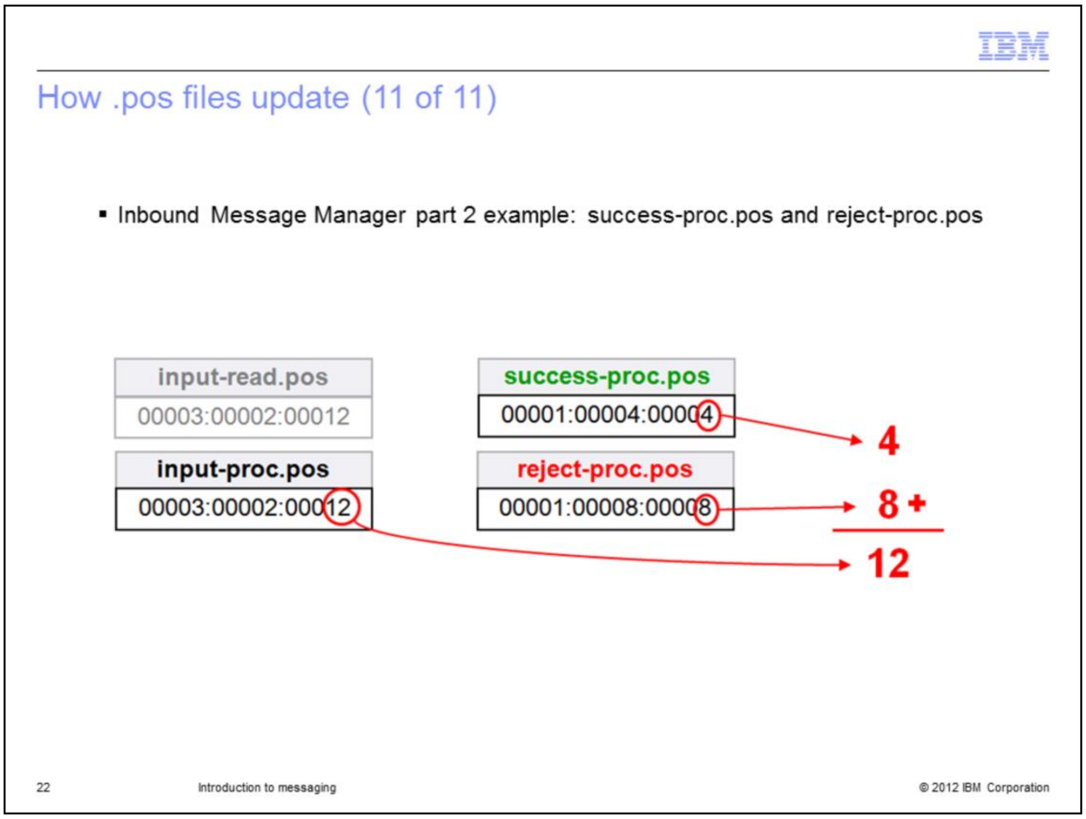


## How .pos files update (10 of 11)

- Inbound Message Manager part 2 example: success-proc.pos and reject-proc.pos

The second message from Tuesday is now processed and is rejected. The second and third values in input-proc.pos are incremented by one and the new values are 2, 2, and 4. The message is copied to the end of the existing reject .dat file and one is added to the last two values in the reject-proc.pos file. The new rejected values are 1, 2, and 2.

Since you are probably starting to get the idea of how this works, the presentation will jump to the end of processing all the remaining messages. There are six messages left over from Tuesday that are rejected and two from Wednesday that are processed successfully. The six messages from Tuesday are added to the end of the current reject .dat file. The two messages from Wednesday are added to the end of the existing success .dat file. The input-proc.pos file is now 3, 2, and 12 and matches the input-read.pos. The reject-proc.pos is updated with the six remaining messages from Tuesday and the final values are 1, 8, and 8. The last two digits for the success-proc.pos have been incremented by two and the final numbers are now 1, 4, and 4.

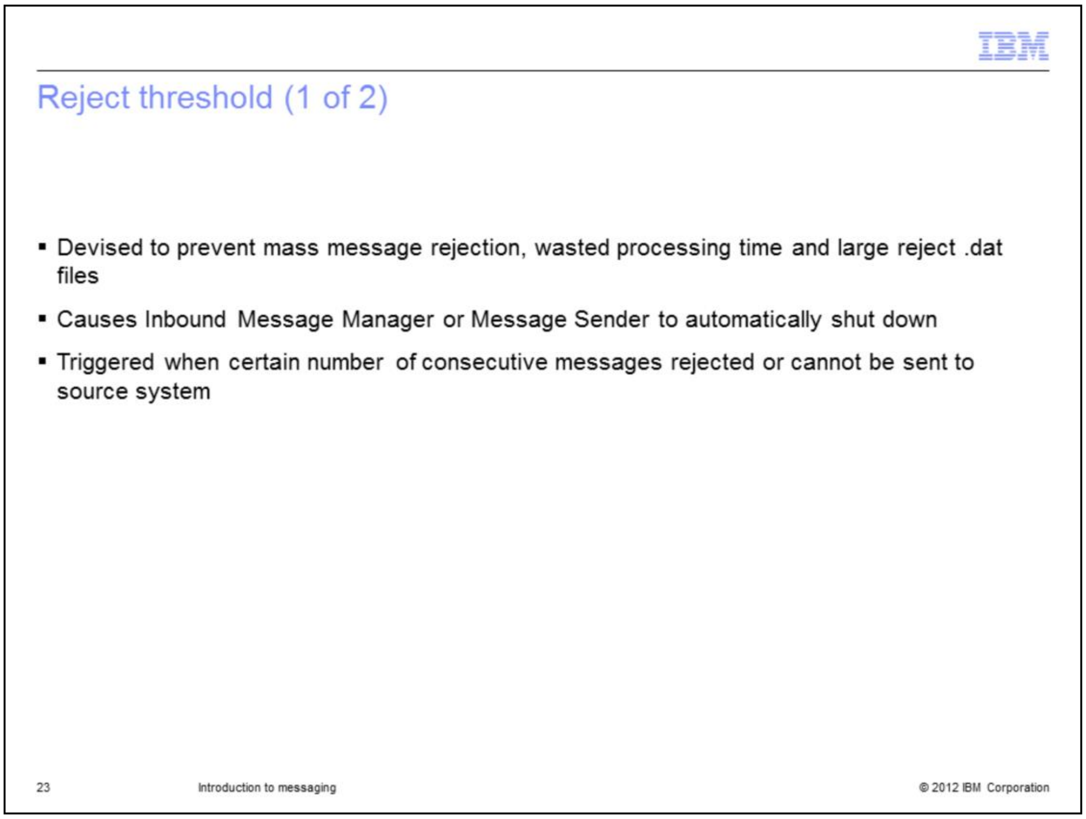


## How .pos files update (11 of 11)

- Inbound Message Manager part 2 example: success-proc.pos and reject-proc.pos

Now look at the last values in the success-proc.pos file and the reject-proc.pos file (the right most numeric value). Notice that if you add these numbers together, they equal the last number in the input-proc.pos file. The total number of successful messages plus the total number of rejected messages equals the total number of input messages. This eludes to the fact that all messages received by the Reader must end up in either the success queue or reject queue.

This concludes the detailed discussion of how the .dat, .pos, and .toc files are used when messages are processed through the message queues.



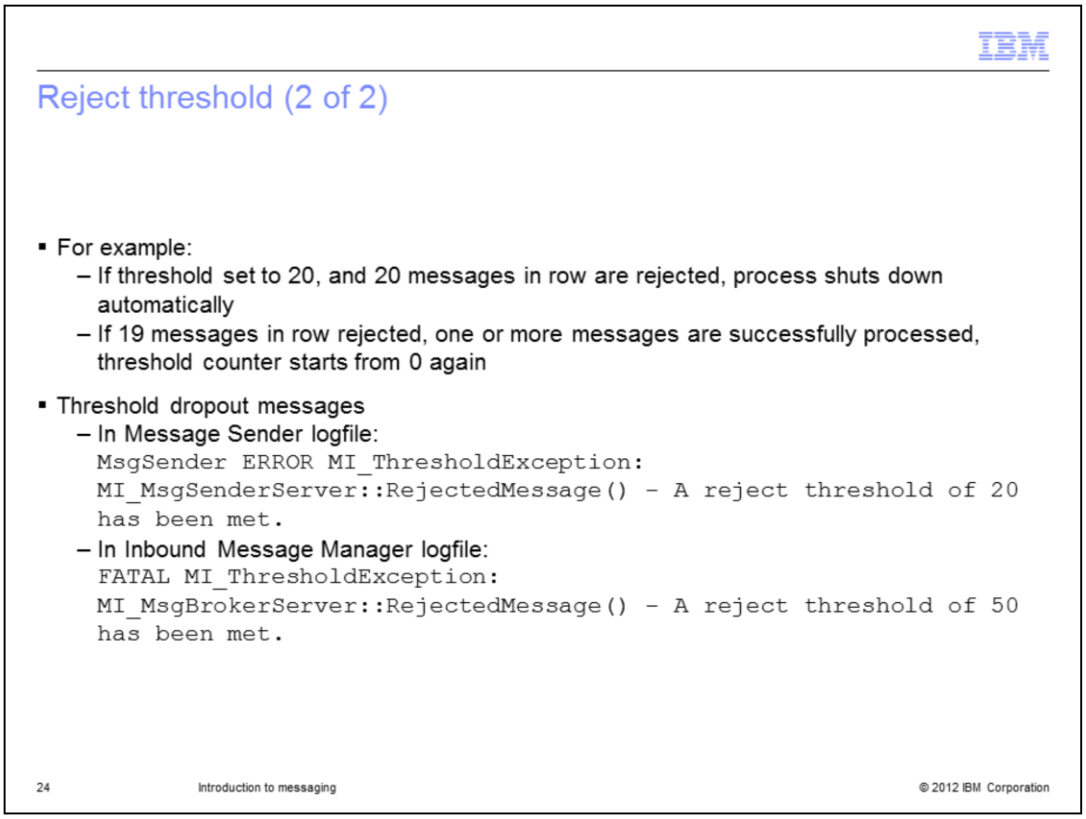
## Reject threshold (1 of 2)

- Devised to prevent mass message rejection, wasted processing time and large reject .dat files
- Causes Inbound Message Manager or Message Sender to automatically shut down
- Triggered when certain number of consecutive messages rejected or cannot be sent to source system

23 Introduction to messaging © 2012 IBM Corporation

The next topic discussed is the reject threshold. The reject threshold was created to prevent a large amount of bad data from being processed and to signal that there is a problem with the data coming in. This saves processing time and having to fix a large amount of messages that get rejected. It is configurable by way of the master configuration file (typically named services.ini or madman.ini).

It is used to determine how many consecutive failures are allowed by the Inbound Broker before it is automatically shutdown or how many messages cannot be sent by the Message Sender before it is shutdown. If a message is successful, the reject threshold counter is reset to zero.



## Reject threshold (2 of 2)

- For example:
  - If threshold set to 20, and 20 messages in row are rejected, process shuts down automatically
  - If 19 messages in row rejected, one or more messages are successfully processed, threshold counter starts from 0 again
- Threshold dropout messages
  - In Message Sender logfile:

```
MsgSender ERROR MI_ThresholdException:
MI_MsgSenderServer::RejectedMessage() - A reject threshold of 20
has been met.
```

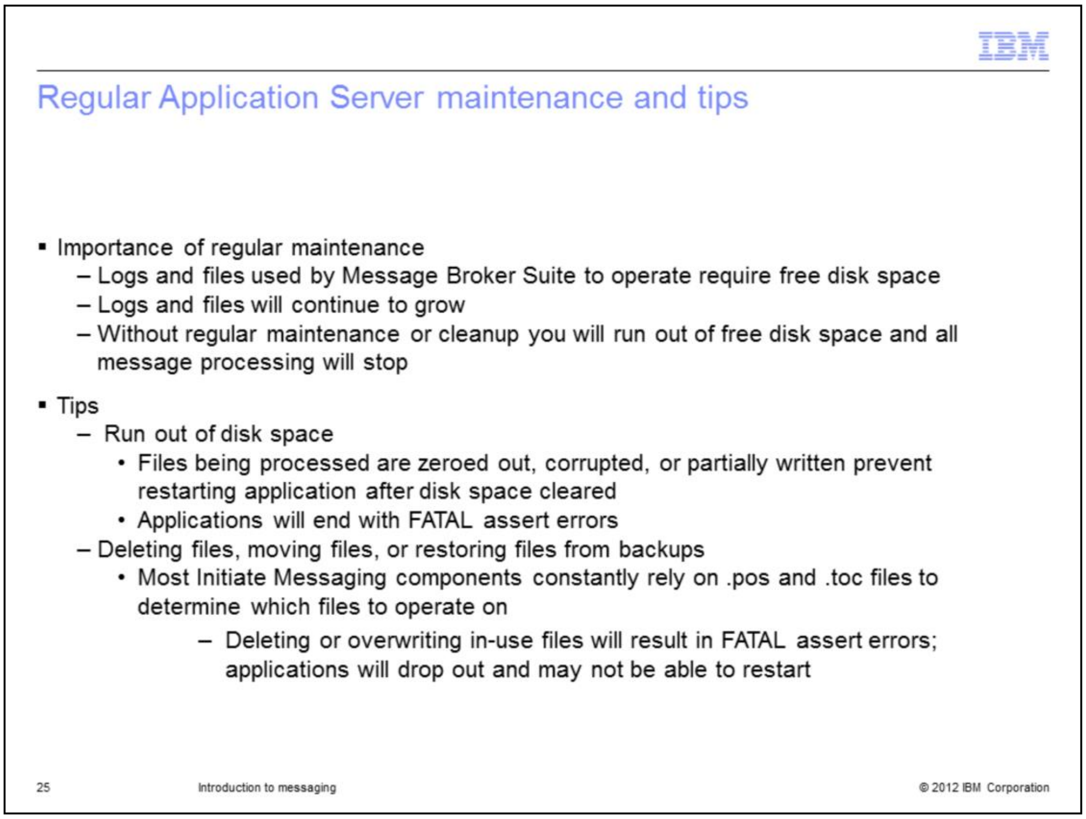
  - In Inbound Message Manager logfile:

```
FATAL MI_ThresholdException:
MI_MsgBrokerServer::RejectedMessage() - A reject threshold of 50
has been met.
```

24 Introduction to messaging © 2012 IBM Corporation

For example, if the reject threshold for the Inbound Broker is set to 20, and 20 consecutive messages are rejected, the process is shut down automatically. If 19 consecutive messages are rejected and the next message is successfully processed, the process will not shut down as the threshold counter is reset on the 20th successful message.

When you reach the reject threshold, there is an ERROR or FATAL message recorded in the broker log. Look at the examples from a Message Sender log and Inbound Broker. You can see the reject threshold for the Message Sender was configured for 20 and the Inbound Broker had its value set to a reject threshold of 50.



## Regular Application Server maintenance and tips

- Importance of regular maintenance
  - Logs and files used by Message Broker Suite to operate require free disk space
  - Logs and files will continue to grow
  - Without regular maintenance or cleanup you will run out of free disk space and all message processing will stop
- Tips
  - Run out of disk space
    - Files being processed are zeroed out, corrupted, or partially written prevent restarting application after disk space cleared
    - Applications will end with FATAL assert errors
  - Deleting files, moving files, or restoring files from backups
    - Most Initiate Messaging components constantly rely on .pos and .toc files to determine which files to operate on
      - Deleting or overwriting in-use files will result in FATAL assert errors; applications will drop out and may not be able to restart

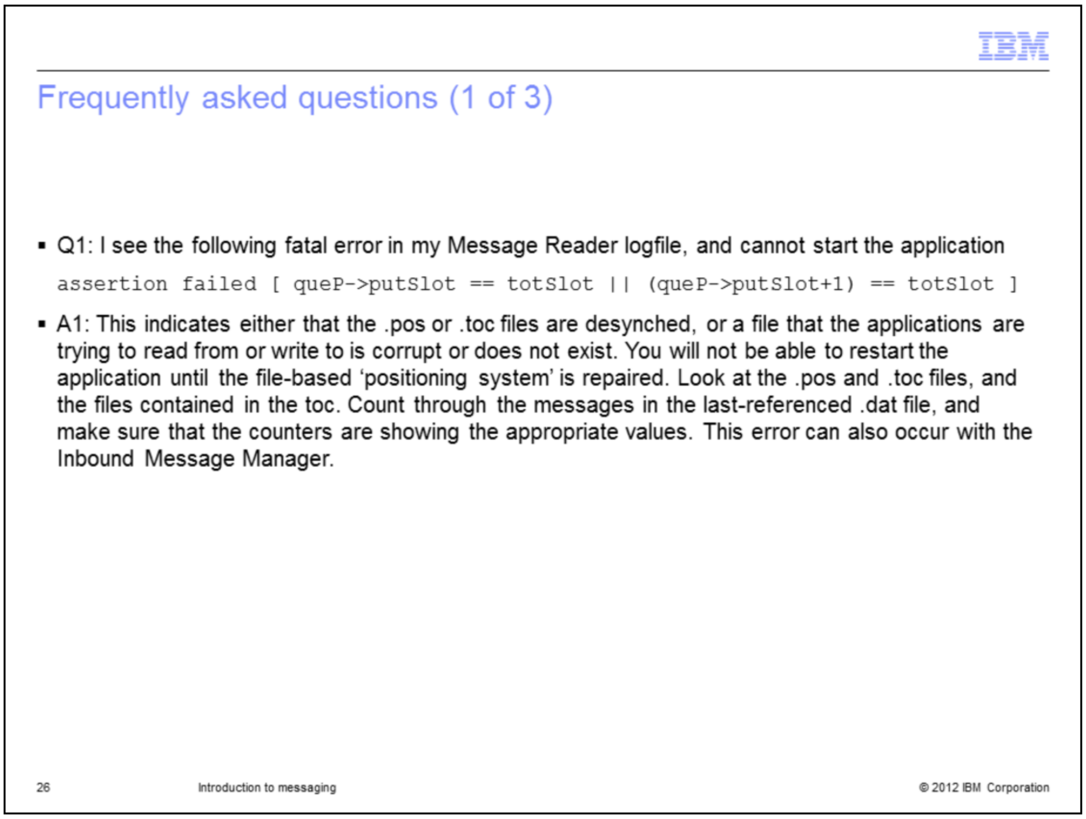
25 Introduction to messaging © 2012 IBM Corporation

The details on regular system maintenance best practices are covered in another presentation. Therefore, this slide only touches on some specifics related to the IBM Initiate Master Data Service Message Broker Suite.

Depending on your specific configuration with all of the messages being passed through all of the different brokers, any interruption in the message processing can become a critical issue for your business if you are not performing regular maintenance.

All of the log files and the files used by the message Brokers suite require free disk space to function. The amount of space these files use on your system will grow over time. If you do not periodically cleanup these files, you will eventually run out of disk space. If you run out of disk space, all of your message processing will come to a halt. Running out of disc space can cause some unusual behavior such as zero byte, corrupt or partially written files. All of which will keep the messaging applications from restarting, even after you have freed up space. The messaging applications will shutdown with FATAL assert errors. If this happens, contact the IBM Initiate Support group for assistance.

When performing maintenance, be careful when deleting, moving, or restoring files being used by the message brokers. Deleting or overwriting any of the files that are actively being used can cause unexpected consequences and prevent you from restarting the application.



## Frequently asked questions (1 of 3)

- Q1: I see the following fatal error in my Message Reader logfile, and cannot start the application

```
assertion failed [ queP->putSlot == totSlot || (queP->putSlot+1) == totSlot ]
```

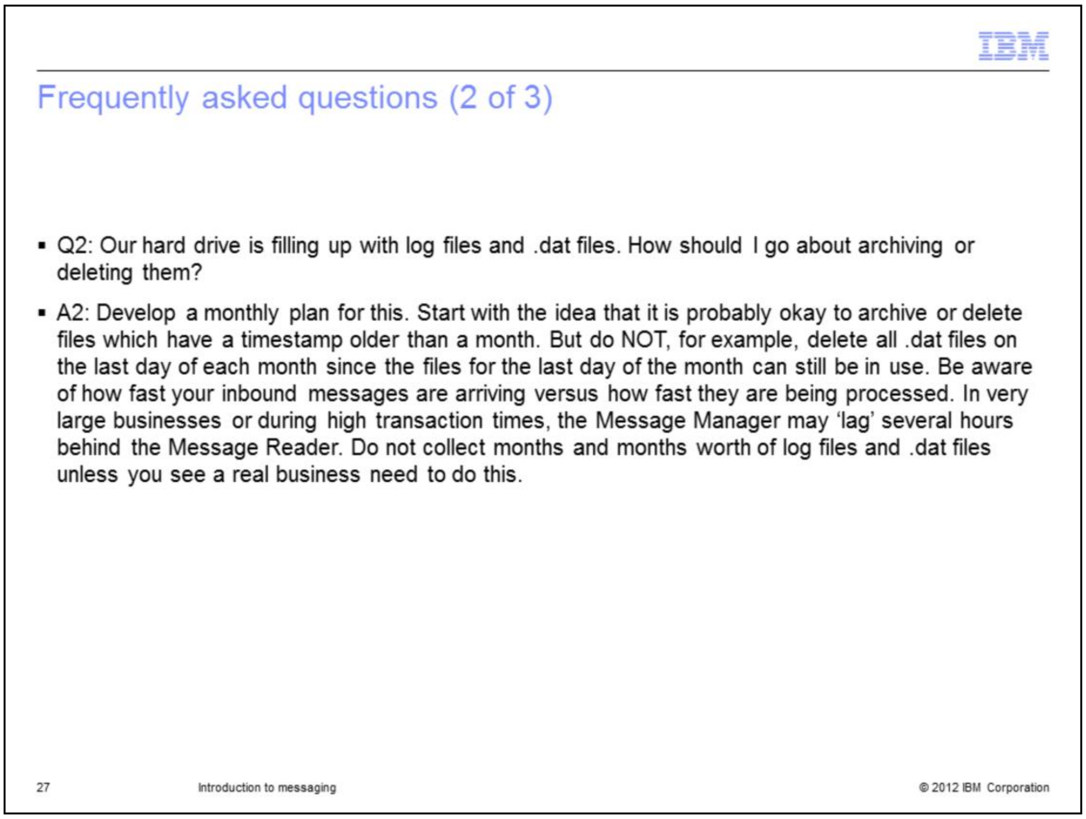
- A1: This indicates either that the .pos or .toc files are desynched, or a file that the applications are trying to read from or write to is corrupt or does not exist. You will not be able to restart the application until the file-based 'positioning system' is repaired. Look at the .pos and .toc files, and the files contained in the toc. Count through the messages in the last-referenced .dat file, and make sure that the counters are showing the appropriate values. This error can also occur with the Inbound Message Manager.

26 Introduction to messaging © 2012 IBM Corporation

The next three slides display some of the most commonly asked questions.

The first question is, what if you see a fatal error in your Message Reader logfile, which mentions a failed assertion, and cannot start the application. How do you start the application?

The answer is that this indicates either that the .pos or .toc files are desynched, or a file that the applications are trying to read from or write to is corrupt or does not exist. You will not be able to restart the application until the file-based 'positioning system' is repaired. To fix this, look at the .pos files, the .toc files, and the files contained in the toc. Count through the messages in the last-referenced .dat file, and make sure that the counters are showing the appropriate values. This error can also occur with the Inbound Message Manager.



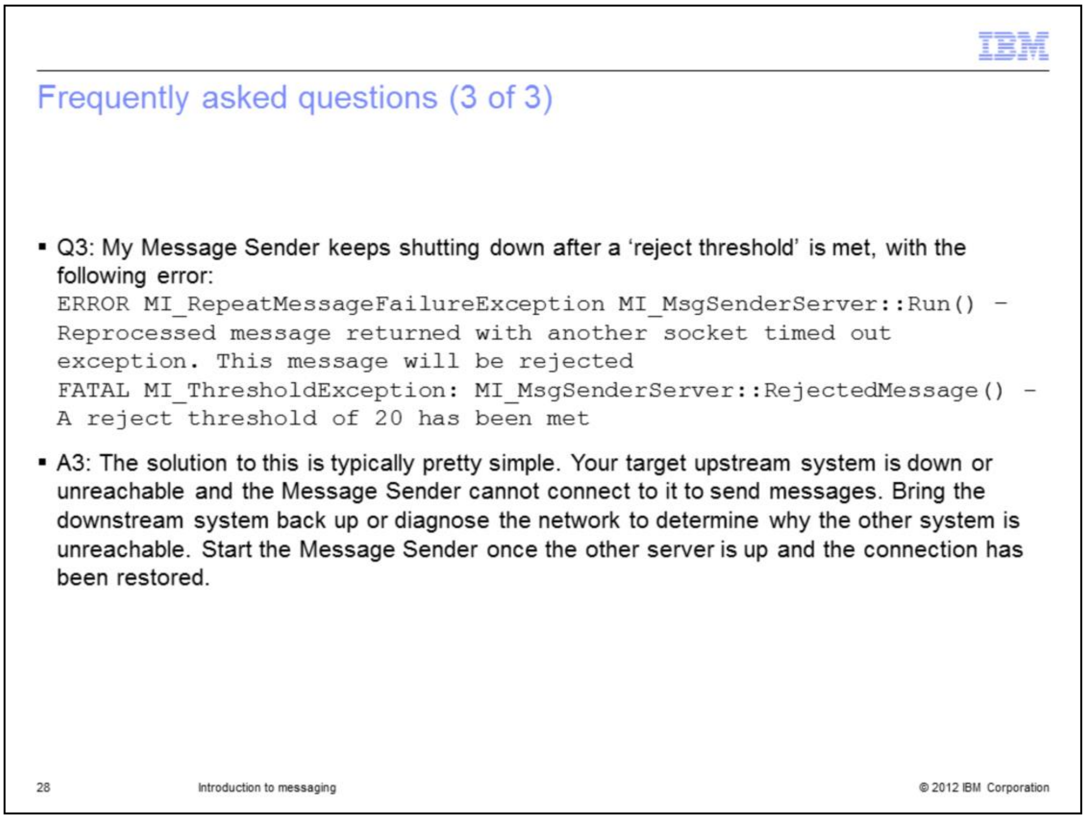
## Frequently asked questions (2 of 3)

- Q2: Our hard drive is filling up with log files and .dat files. How should I go about archiving or deleting them?
- A2: Develop a monthly plan for this. Start with the idea that it is probably okay to archive or delete files which have a timestamp older than a month. But do NOT, for example, delete all .dat files on the last day of each month since the files for the last day of the month can still be in use. Be aware of how fast your inbound messages are arriving versus how fast they are being processed. In very large businesses or during high transaction times, the Message Manager may 'lag' several hours behind the Message Reader. Do not collect months and months worth of log files and .dat files unless you see a real business need to do this.

27 Introduction to messaging © 2012 IBM Corporation

What if your hard drive is filling up with log files and .dat files. How should you go about archiving or deleting them?

The answer is to develop a monthly plan for archiving and deleting files. Start with the idea that it is probably okay to archive and or delete files which have a timestamp older than a month. But do not, for example, delete all .dat files on the last day of each month since the files for that last day of the month can still be in use. Be aware of how fast your inbound messages are arriving versus how fast they are being processed. If you delete a file out from “under” a component, that component will error out. In very large businesses or during high transaction times, the Message Manager may ‘lag’ several hours behind the Message Reader. Do not collect months and months worth of log files and .dat files unless you see a real business need to do so.



## Frequently asked questions (3 of 3)

- Q3: My Message Sender keeps shutting down after a ‘reject threshold’ is met, with the following error:

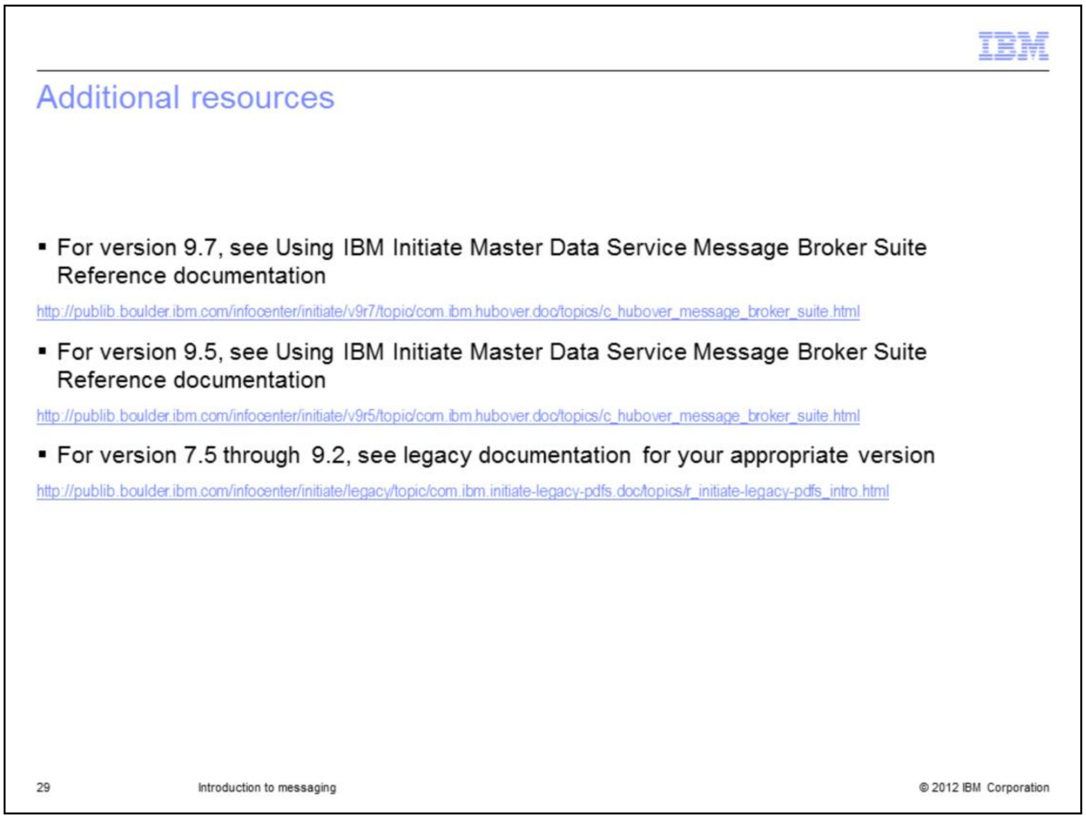
```
ERROR MI_RepeatMessageFailureException MI_MsgSenderServer::Run() –
Reprocessed message returned with another socket timed out
exception. This message will be rejected
FATAL MI_ThresholdException: MI_MsgSenderServer::RejectedMessage() –
A reject threshold of 20 has been met
```

- A3: The solution to this is typically pretty simple. Your target upstream system is down or unreachable and the Message Sender cannot connect to it to send messages. Bring the downstream system back up or diagnose the network to determine why the other system is unreachable. Start the Message Sender once the other server is up and the connection has been restored.

28 Introduction to messaging © 2012 IBM Corporation

The next question involves the Message Sender. Say your Message Sender keeps shutting down after a ‘reject threshold’ is met, with errors. How do you stop this?

The solution to this is typically pretty simple. Your target upstream system is down or unreachable and the Message Sender cannot connect to it to send messages. Bring that external system back up or diagnose the network to determine why the other system is unreachable. Start the Message Sender once the other server is up and the connection has been restored.



## Additional resources

- For version 9.7, see Using IBM Initiate Master Data Service Message Broker Suite Reference documentation

http://publib.boulder.ibm.com/infocenter/initiate/v9r7/topic/com.ibm.hubover.doc/topics/c_hubover_message_broker_suite.html

- For version 9.5, see Using IBM Initiate Master Data Service Message Broker Suite Reference documentation

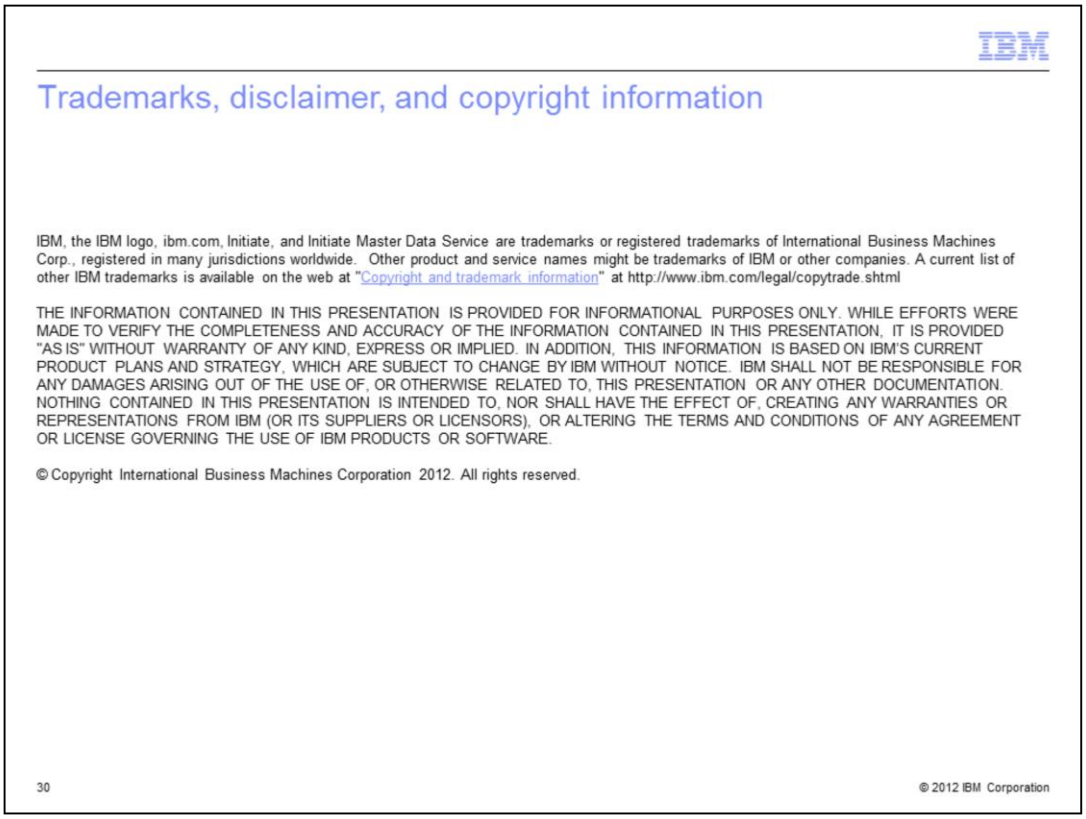
http://publib.boulder.ibm.com/infocenter/initiate/v9r5/topic/com.ibm.hubover.doc/topics/c_hubover_message_broker_suite.html

- For version 7.5 through 9.2, see legacy documentation for your appropriate version

http://publib.boulder.ibm.com/infocenter/initiate/legacy/topic/com.ibm.initiate-legacy-pdfs.doc/topics/r_initiate-legacy-pdfs_intro.html

29 Introduction to messaging © 2012 IBM Corporation

For additional information and details about the IBM Initiate Master Data Service Message Broker Suite, view the documentation on the IBM Support Portal website for the version you are using. This slide displays links to the current versions of the documentation.



## Trademarks, disclaimer, and copyright information