



IBM Software Group

Debugging Top Five IBM IIB/ WMB problems with memory on Windows[®]/UNIX[®]es

Amar Shah

samar@in.ibm.com

Vivek Grover

vgrover@us.ibm.com

IBM Integration Bus Level 3 & Level 2 Support

August 28, 2014



WebSphere[®] Support Technical Exchange



Agenda

- Problem 1: Message parsing and DataFlowEngine memory usage
- Problem 2: Memory growth with large messages/files
- Problem 3: Memory leak Vs Memory fragmentation
- Problem 4: Memory growth in JVM™ component
- Problem 5: Memory growth due to thread caching of objects



Problem 1

Message parsing and DataFlowEngine memory usage



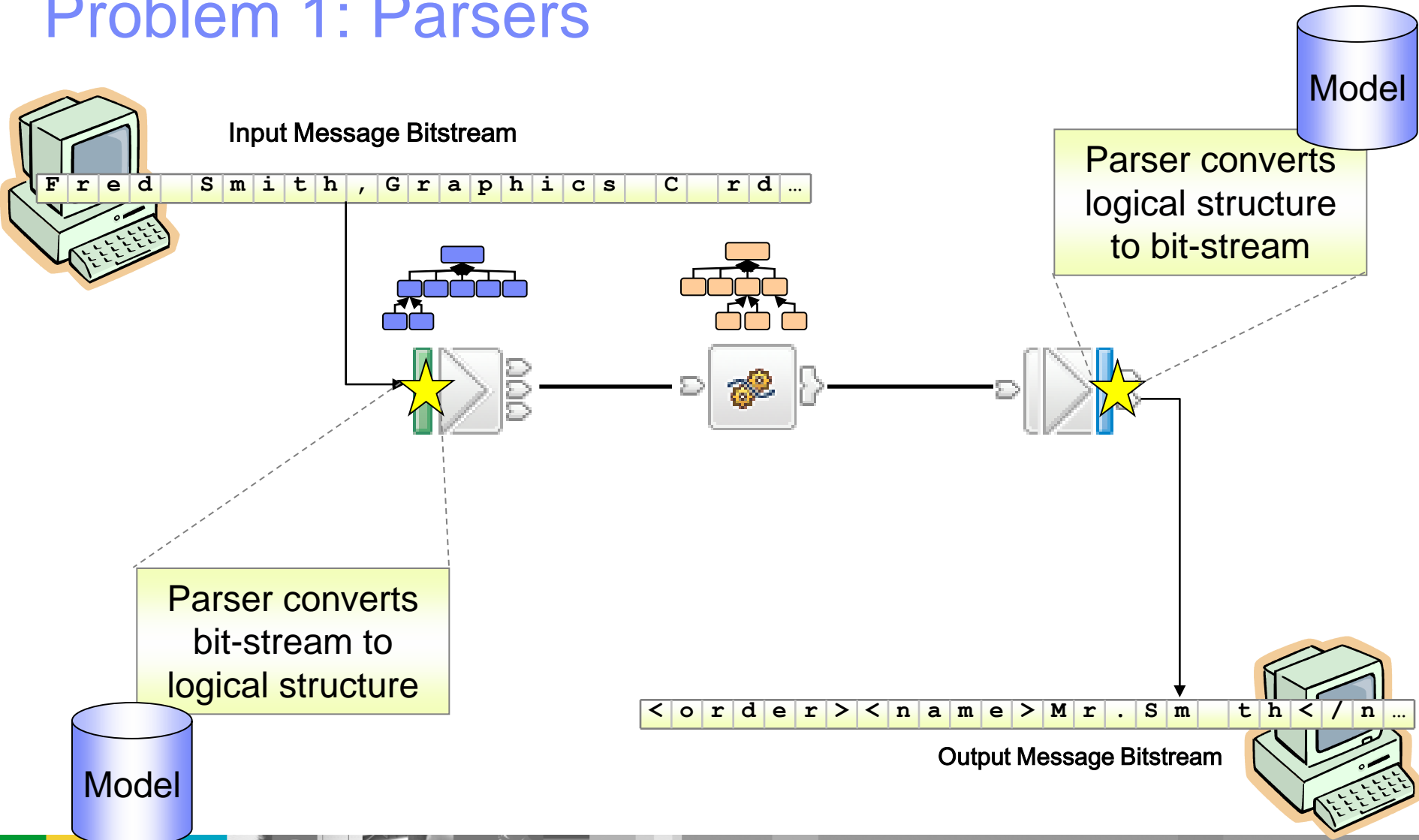
Problem 1: Parsing

- Receive message in one format, transform it and then output it in another format.
- A parser has 2 uses:
 - Parsing: Input message bitstream is converted into a logical structure called a message tree.
 - Serializing: A message tree is converted into output bitstream.
- Applied to message header and body
- Multiple Parsers available: XMLNSC, DFDL, BLOB, SOAP, MIME, JMSMap, JMSStream
- Message complexity, size varies significantly and so do memory and performance costs

JMS™



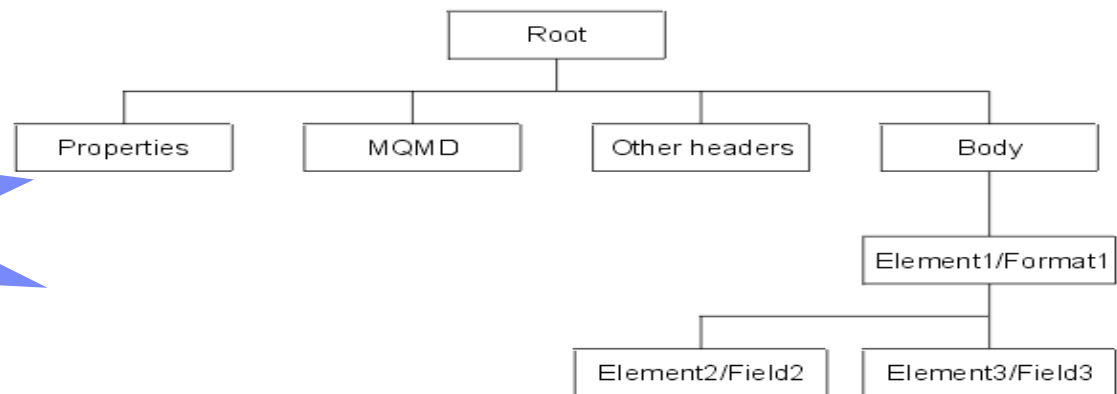
Problem 1: Parsers



Problem 1: Message tree facts

- Logical structure that is created as a result of parsing the message
- Contents are identical to the message
- Easier to manipulate in the Integration flow
- A new message tree is created for every Integration flow invocation
- Cleared down at Integration flow termination
- Tree representation is proportional to but typically larger than the input message

Message size and
complexity of logic can
cause memory usage
to grow



Problem 1: Parsing techniques

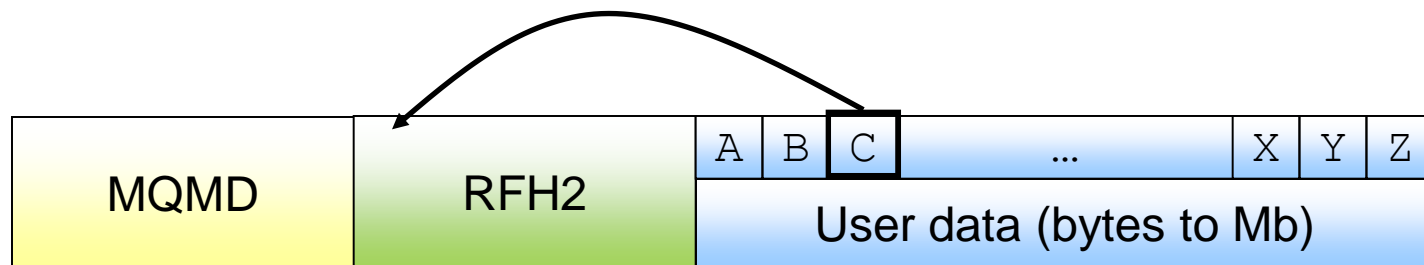
Techniques that can be used to keep memory usage to a minimum:

- If possible, use a compact parser - XMLNSC for XML and DFDL for non-XML data
 - Discards white space and comments in a message when populating the message tree
- Parse only selected portion instead of the whole message unless needed
 - Only the parsed message portion is populated in the message tree
- If whole of the message is required to be processed then parse the whole message on first reference to a field within the message.
 - To ensure a full parse specify:
Choose Parse Immediate or Parse Complete (Parse on Demand – default option)
- Avoid XML sub trees to be parsed by defining Opaque elements
 - sub tree will not be fully expanded and populated into the message tree



Problem 1: Parsing avoidance

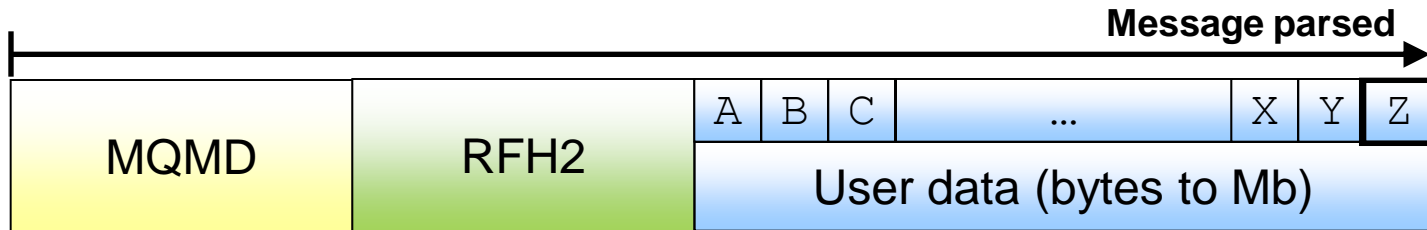
- If possible, avoid the need to parse at all
 - ▶ Promote/copy key data to MQMD, MQRFH2 or JMS Properties
 - May save having to parse the message body
 - Particularly useful for message routing that may only require reading the header info



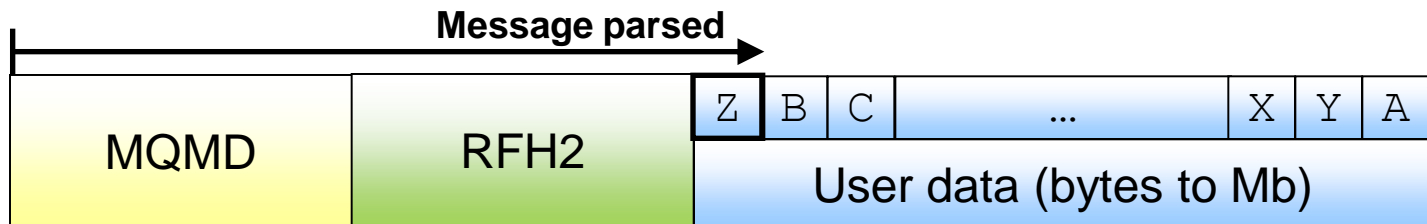
- No need to parse if you don't send
 - ▶ Send only what changed rather than the entire message

Problem 1: Partial parsing

- Typically, the Broker parses elements up to and including the required field
 - ▶ Elements that are already parsed are not reparsed
- If possible, put important elements nearer the front of the message body



VS.



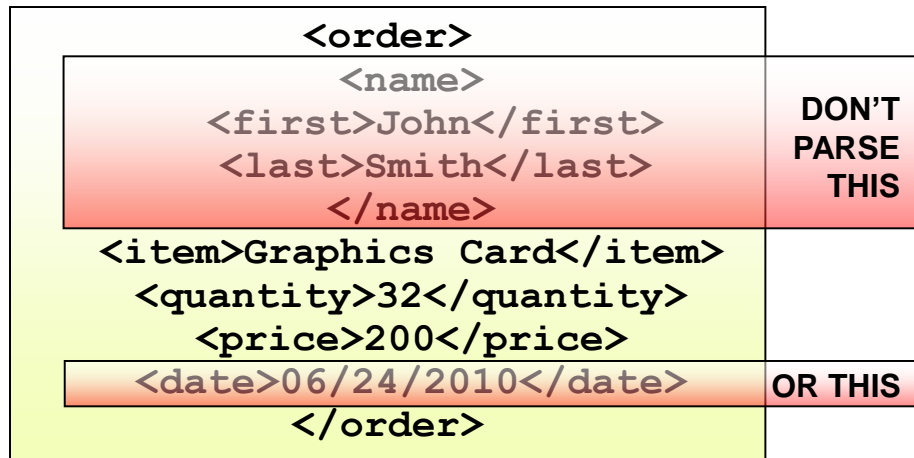
Example:

ESQL: InputRoot.Body.Z

Xpath: `/aaa[1]` if you just want the first instance of the search argument

Problem 1: Opaque parsing

- Allows entire XML sub tree to be placed in the message tree as a single element as an unparsed bitstream
- Reduces message tree size and parsing costs
- Cannot reference the sub tree in message flow processing
- Configure Opaque elements on input nodes ("Parser Options" tab)



For eg. XMLNS:

```

CREATE LASTCHILD OF OutputRoot
DOMAIN('XMLNS') PARSE (BitStream ENCODING
InputRoot.Properties.Encoding CCSID
InputRoot.Properties.CodedCharSetId FORMAT
'XMLNS_OPAQUE' TYPE 'Body');

```

XMLNSC:

MQ Input Node Properties - MQ Input

Description			
Basic	Parse timing <input type="text" value="On Demand"/>		
Input Message Parsing	<input type="checkbox"/> Use MQRFH2C compact parser for MQRFH2 header		
Parser Options	XMLNSC Parser Options		
Advanced	<input type="checkbox"/> Build tree using XML schema data types		
Validation	<input type="checkbox"/> Use XMLNSC compact parser for XMLNS domain		
Security	<input type="checkbox"/> Retain mixed content		
Instances	<input type="checkbox"/> Retain comments		
Monitoring	<input type="checkbox"/> Retain processing instructions		
	Opaque elements		
	<table border="1"> <thead> <tr> <th>Elements</th> </tr> </thead> <tbody> <tr> <td>//name</td> </tr> </tbody> </table>	Elements	//name
Elements			
//name			

Problem 1: Troubleshooting

- Capture resource statistics

```
<ResourceType name="Message parsers">
  <resourceIdentifier name="summary"
    Threads="1"
    ApproxMemKB="3"
    MaxReadKB="1"
    MaxWrittenKB="0"
    Fields="100"
    Reads="1"
    FailedReads="0"
    Writes="0"
    FailedWrites="0"/>

  <resourceIdentifier
    name="Flow1.XMLNSC"
    Threads="1"
    ApproxMemKB="3"
    MaxReadKB="1"
    MaxWrittenKB="0"
    Fields="100"
    Reads="1"
    FailedReads="0"
    Writes="0"
    FailedWrites="0"/>
</ResourceType>
```

Threads	The number of message flow threads that contributed to the statistics for a message flows parser type accumulation.
ApproxMemKB	The approximate amount of user data-related memory used for the named message flow parser type. It is not possible to calculate the exact amount of memory used by a parser.
MaxReadKB	Shows the largest bit stream parsed by the parser type for the named message flow.
MaxWrittenKB	Shows the largest bit stream written by the parser type for the named message flow.
Fields	Shows the number of message fields associated with the named message flow parser type. These fields are retained by the parser and are used for constructing the message trees.
Reads	The number of successful parses that were completed by the named message flow parser type.
FailedReads	The number of failed parses that occurred for the named message flow parser type.
Writes	The number of successful writes that were completed by the named message flow parser type.
FailedWrites	The number of failed writes that occurred for the named message flow parser type.

- List out active parsers and check for `approxMemKB`
`mqsireportproperties <broker> -e <EG> -o ComIbmParserManager -r`

Problem 2

Memory growth with large messages/files



Problem 2: Handling large messages

- Parsing large messages is costly
- May contain repetitive and processing structures that may cause excessive memory use

```
<OuterMessage>
  <InnerMessage>
    <MessageBody>....</MessageBody>
  </InnerMessage>
  ....
</OuterMessage>
```

--- Repeatable

- Fills up the thread stack – Native / Java™



Problem 2: Handling large messages

- To improve use large message processing techniques where possible:
 - Use partial parsing – Parsing on-demand (discussed in Problem 1)
 - Use the DELETE statement in ESQL to delete already processed portions of the incoming message
 - Use Parsed Record Sequence for record detection with stream based processing such as with files
 - Consider using [MQSI_FREE_MASTER_PARSERS](#), MQSI_THREAD_STACK_SIZE, jvmMaxHeapSize, jvmNativeStackSize with **CAUTION !!**
 - Use [Message Splitter pattern](#) and [Large Messaging sample](#)



Problem 2: Handling large messages

- Use the DELETE statement in ESQL to delete already processed portions of the incoming message
 - Copy the body of the input message as a bit stream
 - Create a modifiable copy of the unparsed input message that would reduce memory
 - (Avoid any inspection of the input message, which avoids the need to parse the message)
 - Use a loop and a reference variable to step through the message one record at a time
 - For each record, use the following processes:
 - Use normal transforms to build a corresponding output subtree in a second special folder.
 - Use the ASBITSTREAM function to generate a bit stream for the output subtree
 - (The generated bit stream is stored in a BitStream element that is placed in the position in the output subtree that corresponds to its required position in the final bit stream.)
 - Use DELETE statement to delete both the input and output record message trees
 - DETACH the special folders so that they do not appear in the output bit stream upon completion



Problem 2: Handling large messages

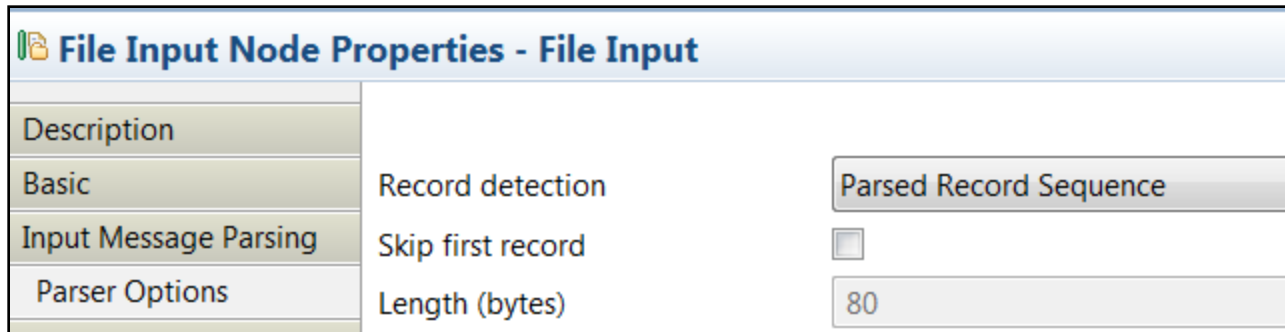
Example:

```
/* Create a reference variable to be used to traverse the input XML message*/  
DECLARE referenceToInputMsg REFERENCE TO rowCachedInputXML.XMLNSC;  
/*Acquire the first root element*/  
MOVE referenceToInputMsg FIRSTCHILD TYPE XMLNSC.Folder NAMESPACE *;  
/*Locate the first repeating structure*/  
MOVE referenceToInputMsg FIRSTCHILD NAMESPACE *;  
FIRST: LOOP  
SET Environment.PatternVariables.temp = FieldName(referenceToInputMsg);  
IF FIELDTYPE(referenceToInputMsg)= XMLNSC.Folder THEN LEAVE FIRST; END IF;  
MOVE referenceToInputMsg NEXTSIBLING;  
END LOOP FIRST;  
/*Delete the message tree allowing the memory to be re-used*/  
DELETE PREVIOUS SIBLING OF referenceToInputMsg;
```



Problem 2: Handling large Files

- User Parsed record sequence for files and packets
- Works only with XMLNSC, DFDL or MRM(CWF or TDS) parsers
- Set the Record Detection property on the Input node to *Parsed Record Sequence*.



The screenshot shows the 'File Input Node Properties - File Input' dialog box. It has a sidebar with tabs: Description, Basic, Input Message Parsing, and Parser Options. The 'Basic' tab is selected. In the 'Record detection' field, the value 'Parsed Record Sequence' is entered. The 'Skip first record' checkbox is unchecked. The 'Length (bytes)' field contains the value '80'.

File Input Node Properties - File Input		
Description		
Basic	Record detection	Parsed Record Sequence
Input Message Parsing	Skip first record	<input type="checkbox"/>
Parser Options	Length (bytes)	80

- With this technique the Input node will use the parser to determine the end of a logical record, which may be determined by length or a simple delimiter like <cr>
- When a logical record has been detected by the parser then it will be propagated through the message flow for processing in the usual way
- The Record Detection property controls how the file is split into sub-transactions

Problem 2: Handling large Files

For Example:

```
<Message>This is a </Message><Message>test  
  created</Message><Message>for this WSTE only</Message>
```

Message 1:

```
<Message>This is a </Message>
```

Message 2:

```
<Message>test created</Message>
```

Message 3:

```
<Message>for this WSTE only</Message>
```



Problem 2: Handling large messages

System/Environment variables:

- `MQSI_FREE_MASTER_PARSERS`
 - Clears the in-memory cache of the broker after every message
 - Should NOT be set in every environment
- `MQSI_THREAD_STACK_SIZE`
 - Increases the stack size for each thread (in bytes)

JVM parameter values:

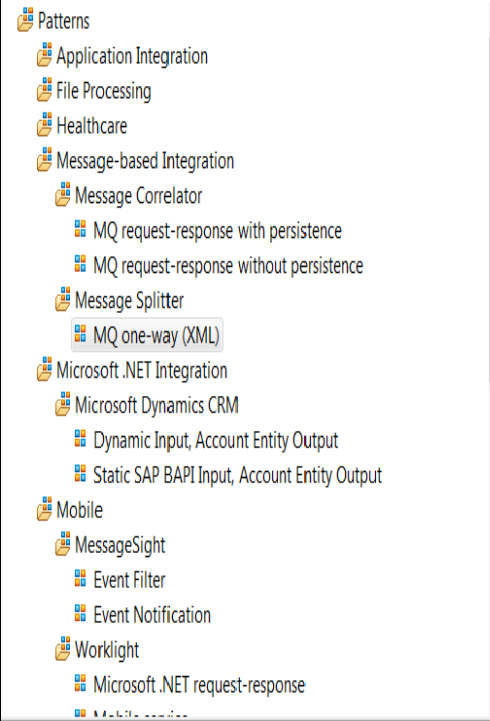
- `jvmMaxHeapSize`
 - maximum size of the storage available to the JVM (in bytes)
- `jvmNativeStackSize`
 - The maximum stack size for Java threads (in bytes)



Problem 2: Message splitter pattern

- Patterns provide top-down, parameterized connectivity of common use cases
- They help describe best practice for efficient message flows
- Write your own patterns too!

Patterns [Download...](#)



- Patterns
 - Application Integration
 - File Processing
 - Healthcare
 - Message-based Integration
 - Message Correlator
 - MQ request-response with persistence
 - MQ request-response without persistence
 - Message Splitter
 - MQ one-way (XML)
 - Microsoft .NET Integration
 - Microsoft Dynamics CRM
 - Dynamic Input, Account Entity Output
 - Static SAP BAPI Input, Account Entity Output
 - Mobile
 - MessageSight
 - Event Filter
 - Event Notification
 - Worklight
 - Microsoft .NET request-response

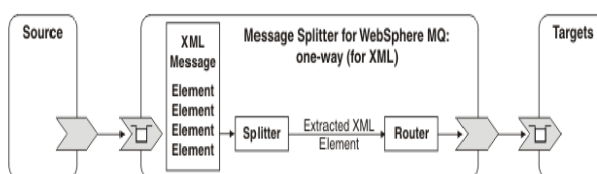
View Pattern Specification
View information about the selected pattern and then click the "Create New Instance" button or click [here](#) to start using a pattern.

Message Splitter for WebSphere MQ: one-way (for XML) pattern

Use the Message Splitter for WebSphere MQ: one-way (for XML) pattern to split a large XML message into smaller elements for processing by one or more targets, by using transactional flows and persistent WebSphere MQ messages.

Use this pattern when you have applications that store information about a number of business transactions and transmit this information to one or more target applications in batches. It can be used to handle large messages without excessive memory use.

This pattern can be used when either the source application or target applications cannot make or process near real-time service calls, or when batching of information is required for business reasons.



```
graph LR; Source[Source] --> XML[XML Message<br/>Element<br/>Element<br/>Element<br/>Element]; XML --> Splitter[Splitter]; Splitter -- "Extracted XML Element" --> Router[Router]; Router --> Targets[Targets];
```

Problem 3

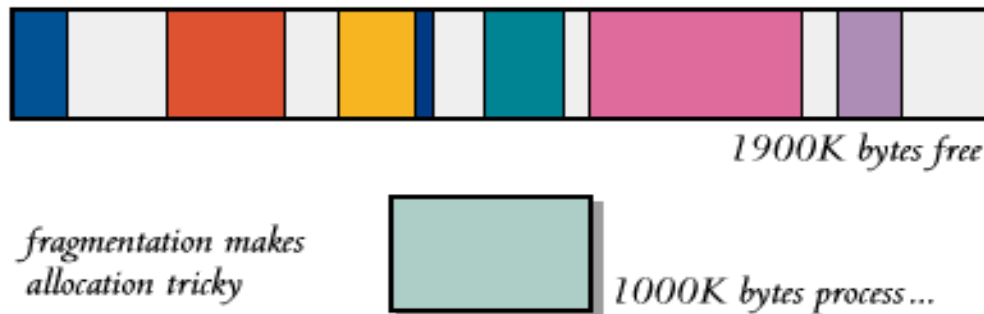
Memory Leak vs Fragmentation



Problem 3: Fragmentation vs Memory Leak

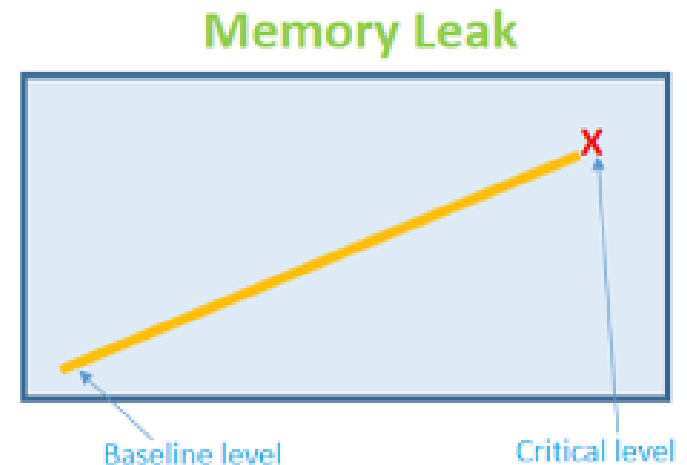
Definition : Fragmentation

Memory fragmentation occurs when a system contains memory that is technically free but that the system can't utilize.



Definition : Memory Leak

A memory leak is the gradual loss of available system memory when a program (an application or part of the operating system) repeatedly fails to return memory that it has obtained for temporary use.



Problem 3: How broker uses memory

- There are objects that are created within the Execution group that last the life of the execution group.
- Every execution group creates its own JVM
 - ▶ use to execute the internal administration threads
- When a message flow runs within the Execution group it runs as a thread within the DataFlowEngine process.
- When a thread is created an area of storage is reserved for that thread in which it can process its stack.



Problem 3: How broker uses memory

General principles :

- A DataFlowEngine process will grow in size to satisfy the storage requests of its threads which will primarily be message flows.
- The DataFlowEngine process will not decrease in size afterwards but will free storage back to its memory heap for re-use.
- A plateau point should be reached at which the memory heap has enough storage to satisfy all the requests from its threads, and this potentially could be a size where all threads are processing simultaneously and are processing their maximum sized message.



Is there a memory leak ?

- Two questions users typically have :
 - ▶ Should the Execution group have used this much storage ?
 - ▶ The execution group has not yet reached its plateau, and hence keeps growing .. why?".
- In either of these two cases the user suspects a memory leak
- Proving whether a memory leak is present or whether the growth is legitimate is difficult and there is no conclusive proof that can be obtained from a users environment.



Problem 3: Determining Fragmentation vs Memory Leak

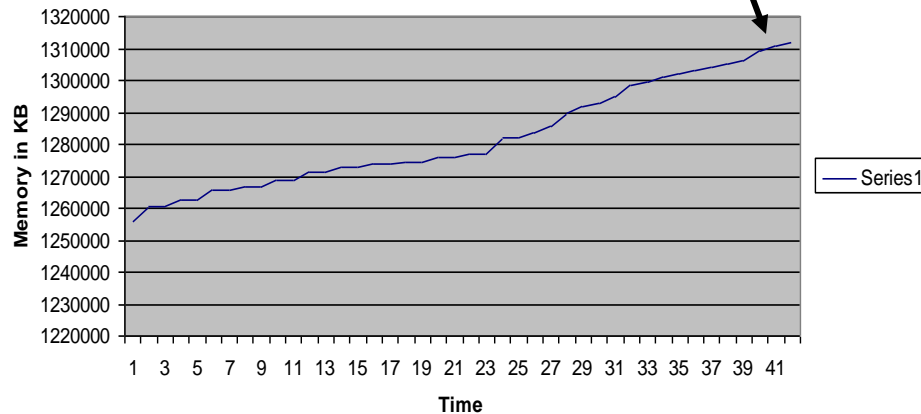
- Isolate the suspected message flow in a separate Execution Group.
- To identify a potential memory leak, the same input message should be repeatedly sent into the message flow.
- This should drive the same message flow paths each time.
 - ▶ Expect the DataFlowEngine process to plateau very quickly
- Monitor memory usage for DataFlowEngine over a period of time.
 - ▶ Simplest command for any Unix/Linux platform is 'ps -elf'.
- Plot a graph of memory usage. Check for any signs of plateau.
- This type of testing needs to be performed with a message flow running in its own execution group with no additional instances deployed.



Determining Memory Leak vs Fragmentation

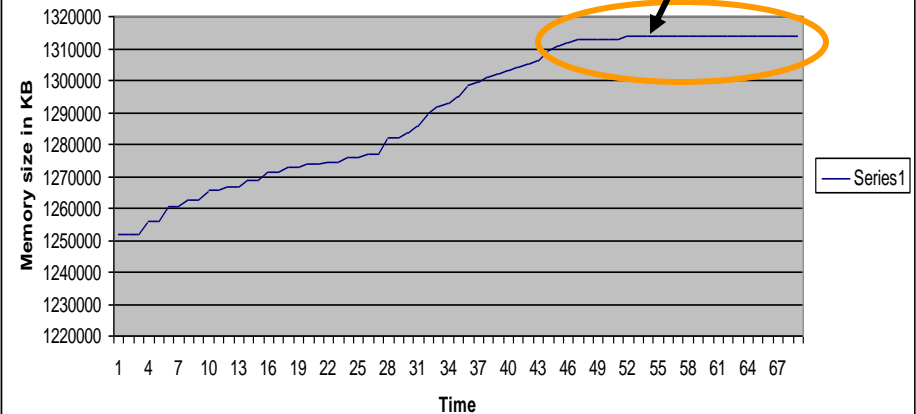
Constant rise

DataFlowEngine Memory Usage



Sign of Plateauing

DataFlowEngine memory usage



			PID	PPID				SIZE						BROKER		EG
242001	A	wmbuser	29556954	52166872	0	60	20 8287	1251992 *	13:07:44	-	7:23	DataFlowEngine	BRK80	2abe571c-4401-000	default	
242001	A	wmbuser	29556954	52166872	0	60	20 8287	1251992 *	13:07:44	-	7:23	DataFlowEngine	BRK80	2abe571c-4401-000	default	
242001	A	wmbuser	29556954	52166872	0	60	20 8287	1251992 *	13:07:44	-	7:23	DataFlowEngine	BRK80	2abe571c-4401-000	default	
242001	A	wmbuser	29556954	52166872	21	60	20 8287	1256072 *	13:07:44	-	7:26	DataFlowEngine	BRK80	2abe571c-4401-000	default	

Problem 3: What causes fragmentation ?

- Inefficient BLOB or CHAR processing.
- When dealing with BLOB or CHAR Scalar variables in ESQL
 - ▶ String manipulation or concatenation
 - ▶ These values need to be held in contiguous buffers in memory
- Additional Instances of message flows

Example 1 :

```
DECLARE c, d CHAR;

SET c = CAST(InputRoot.BLOB.BLOB AS CHAR CCSID
InputProperties.CodedCharSetId);

SET d = c;
DECLARE i INT 1;
WHILE (i <= 52) DO
    SET c = c || d;
    SET i = i + 1;
END WHILE;

SET OutputRoot.BLOB.BLOB = CAST(c AS BLOB CCSID
InputProperties.CodedCharSetId);
```

Example 2 :

```
CREATE FUNCTION changeData (IN oldBlob BLOB) RETURNS BLOB
BEGIN
    DECLARE i INT 1;
    DECLARE newBlob BLOB X'';
    DECLARE lenBlob INT LENGTH(oldBlob);
    WHILE i <= lenBlob DO
        DECLARE tempBlob BLOB SUBSTRING(oldBlob FROM i FOR 1);
        IF (tempBlob < X'5E' ) THEN
            SET newBlob = newBlob || X'40';
        ELSE
            SET newBlob = newBlob || tempBlob;
        END IF;
        SET i = i + 1;
    END WHILE;
    RETURN newBlob;
END;
```

Problem 3: Java String Concatenation

- To concatenate java.lang.String objects use StringBuffer class and append method rather than the + operator
 - ▶ + operator is expensive since it (internally) involves creating a new String object for each concatenation
- **Code such as**

```
keyforCache = hostSystem + CommonFunctions.separator+ sourceQueueValue +  
CommonFunctions.separator+ smiKey + CommonFunctions.separator+ newElement;
```

will perform better written as:

```
StringBuffer keyforCacheBuf = new StringBuffer();  
keyforCacheBuf.append(hostSystem);  
keyforCacheBuf.append(CommonFunctions.separator);  
keyforCacheBuf.append(sourceQueueValue);  
keyforCacheBuf.append(CommonFunctions.separator);  
keyforCacheBuf.append(smiKey);  
keyforCacheBuf.append(CommonFunctions.separator);  
keyforCacheBuf.append(newElement);  
keyforCache = keyforCacheBuf.toString();
```



Problem 3: Effect of additional instances

- Additional Instances property decides the number of threads that a message flow can use at run time to process messages.
 - ▶ Additional instances are not all launched from the start.
 - ▶ created and used based on message load on input side.
 - ▶ ***Start additional instances when the flow starts*** property associated with the message flow will launch all of the threads configured at once.
- Occupies its own storage for all run-time variables, bitstreams, message trees, database connections and interaction etc.
- Memory allocation patterns are more random with additional instances, which may lead to more fragmentation and thus likely to take more time reach the plateau.



Problem 4

Memory growth in JVM component



Problem 4: Java Heap in Message Broker

- Every Execution Group creates its own JVM.
- Heap use is very dependent on the facilities being used
 - ▶ JavaCompute including embedded user Java , XSLT
 - ▶ Message Broker nodes written in Java™ (SAP, TCP/IP,File)
- Other factors impacting Heap usage
 - ▶ Message size / SAP IDOC size
 - ▶ Message flow complexity
 - ▶ Use of non Message Broker Java
 - ▶ Number of instances of a message flow
- Very difficult to recommend one value
 - ▶ For optimal setting need to evaluate on a case by case basis
- Key indicators of a suitably sized heap
 - ▶ **GC Overhead** - percentage of time spent in stop-the-world GC vs. the time spent running the application
 - ▶ **GC Pause Times** - the length of the stop-the-world GC cycles



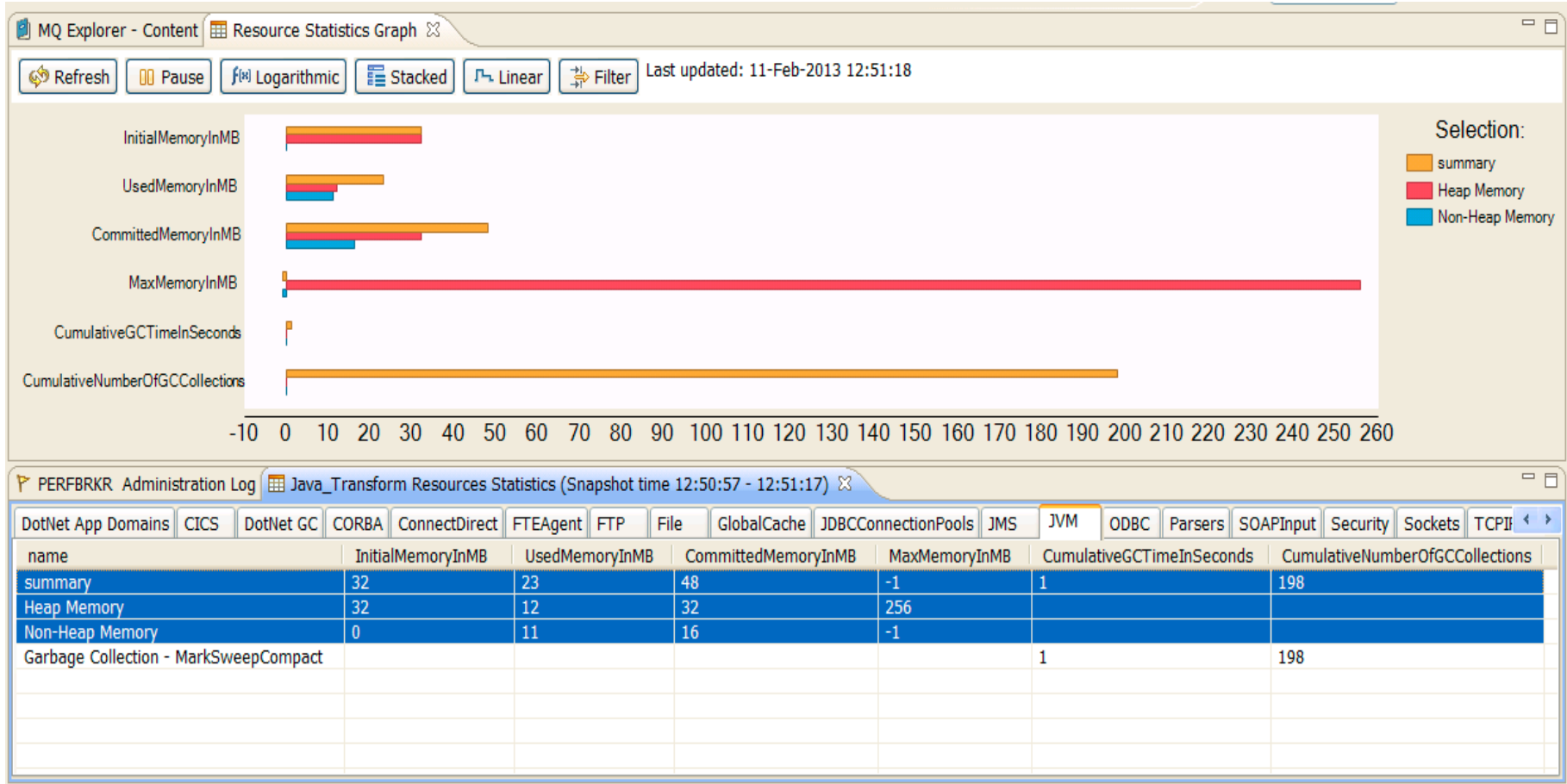
Problem 4: Tuning JVM Settings

- Target values will vary by situation
 - ▶ Batch – low (1%) GC overhead is the goal
 - ▶ Real-time – Low (<1 second) pause times is the goal
- Suggested approach for tuning JVM Heap size
 - ▶ Start with default JVM heap values (min heap=32MB, max heap=256MB)
 - ▶ Run a single instance of the message flow with largest message size
 - ▶ Observe GC overhead and Pause times using Resource Statistics
 - ▶ Increase number of additional instances
 - ▶ Observe GC overhead and Pause times using Resource Statistics
- Change JVM properties using command ***mqsichangeproperties***:
`mqsichangeproperties BRK9 -o ComIbmJVMMManager -e EG
-n jvmMaxHeapSize -v 536870912`



Problem 4: JVM Heap Usage Statistics

- Resource Statistics provide the following for each Execution Group



Problem 4: Troubleshooting JVM issues

- Users typically run into JVM OutofMemory errors due to:
 - ▶ Insufficient max heap size setting to manage the java workload or
 - ▶ There is a memory leak within Java™ components
- Use Garbage Collection analysis using Java™ Health Center to determine if there is a Java leak.
- Enable GC in broker using MQSIJVERBOSE=-verbose:gc



Problem 5

Memory growth due to thread caching of objects



Problem 5: Memory Growth due to caching of objects

- Caching of SQL statements
 - ▶ For performance reason, SQLPrepare statements are cached
 - ▶ Hardcoded SQL strings that differ slightly for each message causes SQLPrepare function to be always performed and cached .
- PASSTHRU statements without parameter markers
- In absence of best practices, the situation potentially leads to many unique SQL statements being cached.
 - ▶ Can lead to memory fragmentation
- Caching can be disabled using MQSI_EMPTY_DB_CACHE
 - ▶ This action might cause a slight performance degradation because every SQL statement is prepared.



Problem 5: Message Flow Thread Stack

- Message Flow requires storage to perform the instructions defined by the message flow nodes.
- A node typically uses 2 KB of the stack space.
 - ▶ For a default thread stack size typical message flow can therefore include 250 nodes on z/OS, 500 nodes on UNIX / Windows systems.

When do you need to increase stack size ?

- Nested or recursive processing can cause extensive usage of the stack.
- A message that contains a large number of repetitions or complex nesting.
- ESQL that calls the same procedure or function recursively
- When an operator ,for example, the concatenation operator, is used repeatedly in an ESQL statement.
- MQSI_THREAD_STACK_SIZE can help mitigate certain situations.
 - ▶ Applies to every thread that is created in a DataFlowEngine
 - ▶ A large value for stack size can lead to large memory requirement by DataFlowEngine



Problem 5: ESQL PASSTHRU Statement

■ Calling PASSTHRU

- ▶ Avoid the use of the PASSTHRU statement with a CALL statement to invoke a stored procedure.
- ▶ Use the "CREATE PROCEDURE ... EXTERNAL ..." and "CALL ..." commands instead

■ Host variables

- ▶ Important to use host variables so dynamic SQL statements can be re-used
- ▶ Host variables map a column value to a variable
- ▶ SQLPrepare is expensive so want to reuse where possible.

▶ Statement:

- `PASSTHRU('UPDATE SHAREPRICES AS SP SET Price=100 WHERE SP.COMPANY='IBM') ;`
- Can only be used when Price is 100 and Company is IBM.
- When Price or Company change need another statement, with another PREPARE

▶ Parameter markers allow Price and Company to change but still use same statement

- `PASSTHRU('UPDATE SHAREPRICES AS SP SET Price = ? WHERE SP.COMPANY = ?', InputRoot.XML.Message.Price, InputRoot.XML.Message.Company) ;`

References

[IBM Integration Bus \(IIB\) V9 Information Center](#)

[WebSphere Message Broker \(WMB\)/ IIB support page](#)

[Techniques to minimize memory usage with IIB](#)

[IIB FAQ for memory](#)

[Important facts about Java TM garbage collection](#)

[Resource statistics](#)



Connect with us!

1. Get notified on upcoming webcasts

Send an e-mail to wsehelp@us.ibm.com with subject line “wste subscribe” to get a list of mailing lists and to subscribe

2. Tell us what you want to learn

Send us suggestions for future topics or improvements about our webcasts to wsehelp@us.ibm.com



Questions and Answers



Additional WebSphere Product Resources

- Learn about upcoming WebSphere Support Technical Exchange webcasts, and access previously recorded presentations at:
http://www.ibm.com/software/websphere/support/supp_tech.html
- Discover the latest trends in WebSphere Technology and implementation, participate in technically-focused briefings, webcasts and podcasts at:
<http://www.ibm.com/developerworks/websphere/community/>
- Join the Global WebSphere Community:
<http://www.websphereusergroup.org>
- Access key product show-me demos and tutorials by visiting IBM® Education Assistant:
<http://www.ibm.com/software/info/education/assistant>
- View a webcast replay with step-by-step instructions for using the Service Request (SR) tool for submitting problems electronically:
<http://www.ibm.com/software/websphere/support/d2w.html>
- Sign up to receive weekly technical My Notifications emails:
<http://www.ibm.com/software/support/einfo.html>

