TechnDoc
WebSphere Message Broker / IBM Integration Bus

Parallel Processing (Aggregation)
(Message Flow Development)
Table of Contents
Introduction ................................................................................................................................. 3
Document Version ....................................................................................................................... 3
Product Naming History ............................................................................................................ 3
Product Component Terminology ............................................................................................... 3
Parallel Processing .................................................................................................................... 4
Programming Overview ........................................................................................................... 4
Message Broker Aggregation Implementation ............................................................................ 5
Queues ....................................................................................................................................... 5
Fan Out Message Flow ............................................................................................................... 6
Fan In Message Flow ................................................................................................................... 7
Error Handling ............................................................................................................................ 8
Solution Design Summary .......................................................................................................... 9
Message Broker Aggregation Nodes .......................................................................................... 10
AggregateControl Node ............................................................................................................ 10
AggregateRequest Node ............................................................................................................ 11
AggregateReply Node .............................................................................................................. 11
Best Practices ............................................................................................................................ 11
References .................................................................................................................................. 12

Table of Figures
Figure 1 - Parallel Programming ................................................................................................. 4
Figure 2 - Aggregation message processing .............................................................................. 5
Figure 3 - WMQ Aggregation Processing .................................................................................... 6
Figure 4 – AggregateControl Node – OutputLocalEnvironment ................................................. 7
Figure 5 – MQOutput Node – OutputLocalEnvironment ............................................................. 8
Figure 6 – AggregateRequest Node - OutputEnvironment ............................................................ 9
Figure 7 - Aggregation Message Tree .......................................................................................... 10
Introduction

Document Version
This document describes how to utilize parallel processing (Aggregation) within a Message Broker (WMB up to v8.x) or Integration Node (IIB v9.0 +) Message Flow. The document should, however, apply to most versions of these products. The contents of this document have been specifically verified on the following production versions:

- WebSphere Message Broker v7.0.0.2
- IBM Integration Bus v9.0.0.0

This documentation has been created and maintained by:

- Glen Brumbaugh

This documentation has been reviewed by:

- Glen Brumbaugh

This documentation was last updated:

- Version 1.0 February 2015

Product Naming History
The product currently known as IBM Integration Bus has been through a number of different product names during its several decade long evolution. The product was originally developed by the New Era of Networks (NEON) Corporation and was marketed and resold by IBM. IBM completely redesigned and rebuilt the product and released their own in-house developed product beginning with version 2.0. The product has had the following names and version numbers:

- MQSeries Integrator (MQSI) Version 1.0 – 2.0
- WebSphere MQSeries Integrator Version 2.1
- WebSphere Business Integration Message Broker (WBIMB) V5.0
- WebSphere Message Broker (WMB) Version 6.0 - 8.x
- IBM Integration Bus (IIB) Version 9.0 - 10.0

For the remainder of this document, the product will be referred to as “Message Broker”. This is both for historical reasons and to signify that this documentation applies to both the WMB and IIB product versions.

Product Component Terminology
With the Version 9.0 product rename (to IBM Integration Bus), several key product architectural components were given new names; while continuing to fill virtually the same role they had previously filled. This documentation will continue to refer to the “old” names because the steps documented here refer to both old and new product versions.

The old and corresponding new names are as follows:

- Message Broker ➔ Now called “Integration Node” (Beginning with v9.0)
- Execution Group ➔ Now called “Integration Server” (Beginning with v9.0)
- Message Flow ➔ Still called “Message Flow"
Parallel Processing

Programming Overview
Most application programming, whether in the C, COBOL, Java, or Message Broker programming languages, deals with sequential processing within a single process/thread. Very few programmers actually have experience with parallel processing, either within a multi-threaded environment or within a programming language that explicitly supports parallel processing. Parallel, or concurrent, programming has, however, been around for quite some time and has been implemented in a number of different programming languages.

Parallel processing (as illustrated in Figure 1) involves two separate operations. The “Fork” operation splits the main process thread into multiple parallel sub-threads. These child threads begin execution immediately. Once all child threads have been launched the parent thread suspends its processing until all the child threads have successfully completed. Each child thread will terminate with a “Join” operation. Once all child threads have terminated with a “Join”, the parent thread will resume processing.

Naturally, unexpected errors can occur during the whole of this process. The primary error associated with parallel processing is the case of a child thread that never completes with a “Join” operation, leaving the “Fork” waiting. This may be due to either a design/programming error or the death of the child thread. This error case is usually handled by resuming the parent thread after a certain amount has passed. This is typically accomplished by using a “timeout” parameter.

To summarize:
- Parallel programming uses the “Fork” and “Join” programming constructs.
- Parent threads issue the “Fork” operation, create child threads, and are then suspended.
- Child threads created by the “Fork” operation terminate with a “Join” operation.
- The parent thread resumes either when all child thread complete or a timeout is reached.
- The failure condition of not all child threads successfully completing must be handled.

**Message Broker Aggregation Implementation**

Since Message Broker is a messaging based technology, it uses messaging terminology to describe its parallel processing. To begin with, the term “Parallel processing” is not used. Instead, Message Broker defines the concept of “Aggregation”. What in parallel programming languages is called a “Fork”, Message Broker terms a “Fan Out”. What parallel programming language calls a “Join”, Message Broker terms a “Fan In”. The data trees from the Fan Out messages are “aggregated” during the “Fan In” process.

Message Broker supports parallel processing through a defined set of nodes built into the Message Broker toolkit. These nodes are normally implemented across two separate Message Flows. The first flow is the “Fan Out” flow. The second Message Flow is the “Fan In” flow. In between these two flows, the “Fanned Out” messages will perform the necessary business processing and generate “Reply” messages that will be aggregated in the “Fan In” flow.

Message Broker uses three different nodes to implement “Aggregation”. The two separate flows and the associated three nodes are referenced in Figure 2 and Figure 3. The three Message Broker Toolkit nodes are:

- AggregateControl Node
- AggregateRequest Node
- AggregateReply Node

---

**Queues**

Message Broker aggregation internally uses WMQ queues to support aggregation processing. The queues used are:

- SYSTEM.BROKER.AGGR.CONTROL
- SYSTEM.BROKER.AGGR.REPLY
- SYSTEM.BROKER.AGGR.REQUEST
- SYSTEM.BROKER.AGGR.TIMEOUT
- SYSTEM.BROKER.AGGR.UNKNOWN
For isolation and performance, it is possible to configure an Aggregation to use alternate queues. This is done through an **Aggregation Configurable Service**. In this case, the Configurable Service creates a “prefix” that is used to identify differently named queues. The resulting queue names are:

- **SYSTEM.BROKER.AGGR.\prefix.xxx** (Where “\prefix” = “CONTROL”, “REPLY”, etc.)

For more information, the References Section contains a Knowledge Center link to this topic.

---

**Fan Out Message Flow**

The “Fan Out” Message Flow begins with a request message from a user. This request is processed by an Input node. This input node begins a transaction (Logical Unit of Work) that will not be committed until the **AggregateRequest** nodes.

The actual “Fan Out” (Fork) process itself begins in the **AggregateControl** node. This node allows multiple nodes to be connected to its “Out” terminal. These multiple “Out” connections define the scope of the “Fan Out” (parallel processing). The node also defines the **Aggregation Name** as a Node property, so that data from different aggregations can be kept separated in the queues. At runtime, this Node defines the **Aggregation ID**, which enables the separation of data.
from multiple instances of the same aggregation. Both of these fields are stored in the OutputLocalEnvironment tree (See Figure 4).

The parallel “Fan Out” nodes must each generate a “Request” message that will be processed outside of the “Fan Out” message flow. Each of these “Request” messages will, in turn, generate a “Reply” message that will be input into the “Fan In” Message Flow. When these “Request” messages are generated, typically by an MQOutput node, the node updates the OutputLocalEnvironment element, adding a WrittenDestination element. This is illustrated in Figure 5.

Each “Fan Out” (Child) process ends with an AggregateRequest node. The AggregateRequest node provides a name (called a “Folder”) for each of instance of a child process. These “Folder” names will be aggregated under a ComIbmAggregateReplyBody domain during the “Fan In” process. This node also checks to see if it is the last Fan Out message flow. If it is, this node commits the transaction. This commits consumes the original request message and releases the request messages from each of the Fan Out flows. This node also updates the OutputEnvironment element. This structure is illustrated in Figure 6. The Environment element fields updated include:

- Aggregation Name (Element name; from AggregateControl node property)
- Folder Name (Element name; from AggregateRequest node property)
- Reply Identifier (Matches value set in Output node)
- Reply Protocol (Typically “MQ”)

Fan In Message Flow
The actual “Fan In” process itself begins with an Input node to process the “Reply” messages generated in response to the “Fan Out” request messages. The Input node begins a transaction (Logical Unit of Work) that will not be committed until the final Output node.

The “Fan In” (Join) process itself begins and ends with the AggregateReply node. This node consumes the “Reply” messages that are generated as a result of the multiple “Request” messages generated during the “Fan Out” processing. The node aggregates all of these “Reply” messages into the ComIbmAggregateReplyBody element. Under that element, each “Reply” is stored under a named “Folder”. The name of that folder is the name assigned by the
corresponding `AggregateRequest` node. The `ComlbmAggregateReplyBody` structure is illustrated in Figure 7.

Figure 5 – MQOutput Node – OutputLocalEnvironment

**Error Handling**
There are a number of different types of errors that may occur in an aggregation Message Flow. These errors include:

- A “Reply” message that cannot be associated with a request (“Unknown”).
- A late (if there is a specified timeout) “Reply” message (“Timeout”).
- A dropped “Reply” message, causing either a timeout or the aggregation to hang.
- A downstream error (after AggregationReply node) causing an exception to be thrown.

All of the errors must be handled in the following “Fan In” Message Flow node terminals:

- Input Node.Catch terminal (e.g. MQInput node)
- AggregateReply.Catch terminal
If these errors are not “caught” in those two nodes, then default Message Broker error handling will be performed. In the References Section, see the link regarding Error Handling for additional information.

**Solution Design Summary**

The design of an Aggregation solution thus has the following properties:

- The solution has two separate Message Flows: “Fan Out” and “Fan In”.
- Each “Fan Out” Message Flow:
  - Begins with a single Input node.
  - The “Fan Out” process begins with an AggregateControl node.
  - Each “Fan Out” parallel flow has an Output node generating a “Request” message.
  - Each “Fan Out” parallel flow Output node is followed by an AggregateRequest node.
- The “Request” messages generated in the “Fan Out” parallel flows:
  - Must ultimately result in a “Reply” message being generated.
  - All “Reply” messages must be routed to the same queue.
  - The “Reply” queue is used as the Input queue for the “Fan In” Message Flow.

![Figure 6 – AggregateRequest Node - OutputEnvironment](image-url)
• Each “Fan In” Message Flow:
  o Begins with a single Input node.
  o All “Reply” messages must ultimately be routed to this Input node.
  o The AggregateReply node performs the “Fan In” function.
    ▪ “Reply” messages aggregated into a ComIbmAggregateReplyBody structure.

![Message Broker Aggregation Nodes Diagram]

**Message Broker Aggregation Nodes**

**AggregateControl Node**

- The AggregateControl node is used in the “Fan Out” message flow.
- This node must be preceded by an Input node.
- This node must precede the AggregateRequest node.
- The input node defines the beginning of a transaction.
- The AggregateControl node must participate in the transaction.
- This node begins the “Fan Out” process.
• “Fan Out” parallel flows are defined by multiple outputs from the “Out” terminal.
• This node names this particular “Aggregation”.
• Updates the OutputLocalEnvironment tree.
  o Adds a ComIbmAggregateControlNode child element.
  o Adds ComIbmAggregateControlNode child fields.
    ▪ aggregateName
    ▪ aggregateId
• All parallel flows are passed identical tree structures.

AggregateRequest Node
• The AggregateRequest node is used in the “Fan Out” message flow.
• This node must be preceded by the AggregateControl node.
• This node must be preceded by an Output node.
  o All Output messages must ultimately result in a “Reply” message.
  o All “Reply” messages must ultimately be routed to the same queue.
• The AggregateRequest node defines the end of a transaction.
  o Transaction committed when the last parallel AggregateRequest node is processed.
• This node names the “Folder” associated with this parallel “Request” message flow.

AggregateReply Node
• The AggregateRequest node is used in the “Fan In” message flow.
• This node must be preceded by an Input node.
  o “Reply” messages resulting from “Fan Out” parallel flows route to this Input node.
• The AggregateRequest node builds the ComIbmAggregateReplyBody structure.
• The ComIbmAggregateReplyBody has a “Folder” element for each “Reply” message.
  o The name of the “Folder” element is assigned in the AggregateRequest node.
  o The “Folder element corresponds to the “OutputRoot” of the “Reply” messages.
• The “UnknownMessageTimeout” property
  o Delays processing a “Reply” message for which there is no control information

Best Practices

• MQInput Node (“Fan Out”): Transaction mode MUST be set to “Yes”. Subsequent nodes MUST use a setting of “Automatic”. This ensures that the input messages are processed under a syncpoint and only committed when the last message is processed by the AggregateRequest node. This accomplish two essential things:

  (1) Data integrity is ensured. Input “Request” messages are rolled back onto the input queue if the transaction is rolled back.

  (2) No “Request” messages are released until the end of the transaction. This ensures that the control information required by the AggregateReply node is supplied before any of the “Reply” messages arrive.

• AggregateControl Node: Do not use the Control terminal. This terminal is “deprecated” and remains only to support earlier product versions. The use of this terminal has no benefit and
can cause performance degradation. In the References Section, see the link regarding Thread Starvation.

- **MQInput Node (“Fan In”):** Transaction mode MUST be set to “Yes”. Subsequent nodes MUST use a setting of “Automatic”. This ensures that the input messages are processed under a syncpoint and only committed after the final “Reply” message is written, for example, by an MQOutput node. This accomplish two essential things:

  1. Data integrity is ensured. Input “Reply” messages are rolled back onto the input queue if the transaction is rolled back.
  2. Since exceptions may be thrown to either the Input node or the AggregateReply node, this ensures a consistent behavior across different types of exceptions.

**References**

- IBM – WebSphere Message Broker – Knowledge center (v7.0, 8.0)

- IBM – IBM Integration Bus – Knowledge center (v9.0, 10.0)

- IBM – IIB Knowledge center – Aggregation – Associating “Fan Out” and “Fan In” Message Flows

- IBM – IIB Knowledge center – Aggregation – Creating the aggregation “Fan Out” flow

- IBM – IIB Knowledge center – Aggregation – Creating the aggregation “Fan In” flow

- IBM – IIB Knowledge center – Aggregation – Handling exceptions in aggregation flows

- IBM – IIB Knowledge center – Aggregation – Setting timeout values for aggregation
- IBM – IIB Knowledge center – Aggregation – Avoiding thread starvation in “Fan In” flows

- IBM – IIB Knowledge center – Aggregation – Configuring queues used for Aggregation
  http://www-01.ibm.com/support/knowledgecenter/?lang=en#/SSKM8N_8.0.0/com.ibm.etools.mft.doc/bc28130.htm?cp=SSKM8N_8.0.0%2F1-7-7-0-3-0

- IBM – IIB Knowledge center – Samples - Aggregation

- IBM – developerWorks – Extending Aggregation

- IBM – developerWorks – Dynamic Message Aggregation