

# **MQSeries Integrator V2 Aggregator Plug-In Version 1.8**

11<sup>th</sup> September 2002

Mike Brady  
Senior Consulting IT Specialist  
IBM  
Australia

[mjbrady@au1.ibm.com](mailto:mjbrady@au1.ibm.com)

**Property of IBM**

Take Note!

Before using this report be sure to read the general information under "Notices".

**Tenth Edition, September 2002**

This edition applies to Version 1.8 of *MQSeries Integrator V2 – Aggregator Plug-In* and to all subsequent releases and modifications unless otherwise indicated in new editions.

© **Copyright International Business Machines Corporation 2001**. All rights reserved. Note to US Government Users -- Documentation related to restricted rights -- Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule contract with IBM Corp.

---

## Table of Contents

MQSeries Integrator V2 Aggregator Plug-In .....	i
Notices .....	iv
Trademarks and service marks .....	iv
Summary of Amendments .....	v
Preface .....	vii
Chapter 1. Overview .....	1
Chapter 2. Installation .....	6
SupportPac contents .....	6
Prerequisites .....	10
Supported platforms .....	10
Installing the plug-in nodes on NT .....	10
Installing the plug-in nodes on AIX .....	11
Installing the plug-in nodes on Solaris .....	11
Installing the plug-in nodes on HP-UX .....	11
Integrating the plug-in node into the Windows Control Center .....	12
Installation verification .....	12
Chapter 3. Nodes Reference .....	13
SpAggregateCreate Node .....	13
SpAggregateCreate node properties .....	14
Configuring the SpAggregateCreate node .....	14
SpAggregateReply Node .....	15
SpAggregateReply node properties .....	16
Configuring the SpAggregateReply node .....	18
Tracing the aggregator plug-in nodes .....	19

---

## Notices

The following paragraph does not apply in any country where such provisions are inconsistent with local law.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent program that does not infringe any of the intellectual property rights may be used instead of the IBM product.

Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS-IS. The use of the information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item has been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

## Trademarks and service marks

The following terms, used in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

- AIX
- IBM
- MQSeries
- MQSeries Integrator
- MQSI

The following terms are trademarks of other companies:

HP-UX	Hewlett Packard Corporation
Windows NT	Microsoft Corporation
Solaris	Sun Corporation

---

## Summary of Amendments

Date	Changes
6 <sup>th</sup> March 2001	Initial release
1 <sup>st</sup> May 2001	<ul style="list-style-type: none"><li>• AIX version released</li><li>• remove SingleReplyMessage property and reject aggregate messages containing same CorrelId but different MsgId values</li><li>• add AggregatedReplyDestination property</li><li>• send timer messages to queue specified by ReplyToQueue property. Remove dependency on SYSTEM.BROKER.TIMER.QUEUE</li><li>• encapsulate SpARI input queue (specified by ReplyToQueue property) within SpAggregateReply composite node</li><li>• remove 'Configurable Parameters' folder and replace with 'Basic' and 'Advanced' folders</li><li>• fix timer thread errors during re-deployment</li><li>• add support for multiple deployed aggregates running on the same broker</li><li>• improve recognition of timer messages through unique PutAppName value</li></ul>
11 <sup>th</sup> June 2001	<ul style="list-style-type: none"><li>• added recovery functions</li><li>• improved format of log messages</li><li>• fixed deployment failure caused by error in Unicode conversion routines</li><li>• set PutDate and PutTime values in timer messages</li><li>• Documented the restriction that all nodes of an aggregate must exist within the same Message Flow</li><li>• Added logic to police this restriction</li><li>• Include separator string in timed out reply messages</li><li>• check that all messages belonging to the same aggregate have the same MsgId value</li></ul>

- |                                 |  |
|---------------------------------|--|
| 26 <sup>th</sup> June 2001      | <ul style="list-style-type: none"><li>• Changes for compatibility with new deployment logic introduced by MQSeries Integrator V2.0.2</li><li>• Added support for aggregated reply message in XML format</li><li>• Added AggregatedReplyDataType property</li><li>• Added AggregatedReplyLabelName property</li></ul> |
| 11 <sup>th</sup> July 2001      | <ul style="list-style-type: none"><li>• Solaris version released</li></ul>   |
| 2 <sup>nd</sup> August 2001     | <ul style="list-style-type: none"><li>• Fixed memory leak which manifested itself when data type was XML or Recoverable was set to true</li><li>• Fixed deployment bug which prevented new deployment while outstanding aggregates existed</li></ul>   |
| 29 <sup>th</sup> August 2001    | <ul style="list-style-type: none"><li>• HP-UX version released</li><li>• Fixed exception handling error</li><li>• Removed DTD information from XML member messages before aggregation</li></ul>  |
| 5 <sup>th</sup> November 2001   | <ul style="list-style-type: none"><li>• Added error handling for messages with missing MsgId and/or CorrelId</li></ul>   |
| 21 <sup>st</sup> November 2001  | <ul style="list-style-type: none"><li>• Set Format to MQFMT_STRING for XML reply messages.</li></ul>   |
| 11 <sup>th</sup> September 2002 | <ul style="list-style-type: none"><li>• Add preprocessor directives to allow source to build correctly against v2.1, v2.0.2 and v2.0.1 releases</li><li>• Shipped v2.0.1 and post v2.0.1 binaries</li></ul>  |

---

## Preface

Many customers are involved in building new front-end applications that provide a unified view spanning multiple backend legacy applications. Typically such an implementation requires a layer of code that takes a single request from a front-end application, for example a web browser application, and turns this into multiple requests, one for each of the backend systems involved. This is commonly referred to as request 'fan out'. Usually this same code is required to wait for and consolidate the replies from each of the backend systems, building a single response for the front-end application. This stage is commonly referred to as 'fan in' or 'aggregation'.

With the introduction of a message broker as an intermediary between the front-end application and the legacy systems, the co-ordination of the reply messages is often made more difficult by the need to direct all replies to the broker prior to consolidating them. Typically this involves modifying the ReplyToQ details in the outbound request so that the legacy system sends all reply messages back to a queue owned by the broker, and not the original reply queue specified by the front-end application.

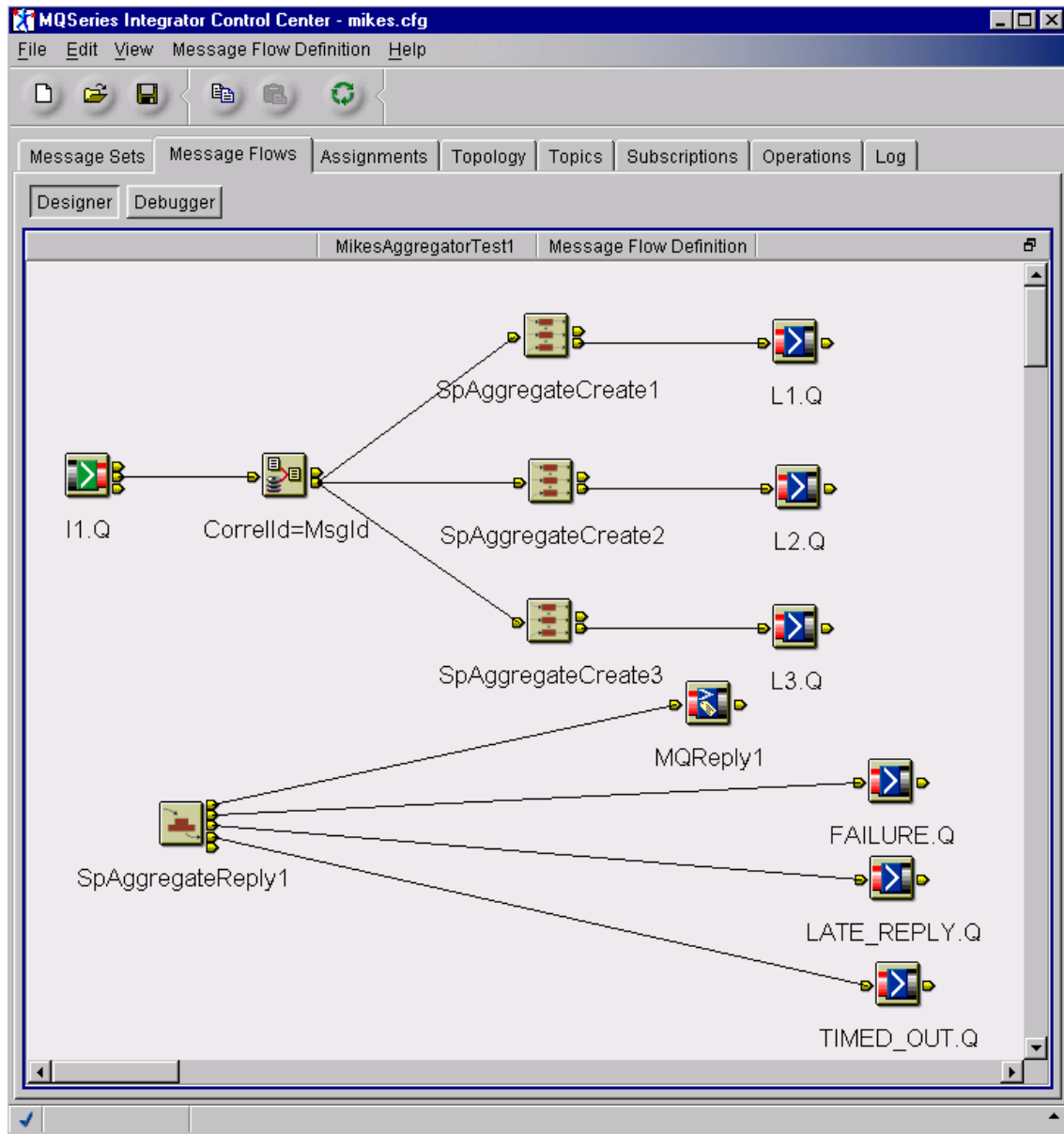
The MQSI V2 Aggregator function provides a mechanism for coordinating the delivery of reply messages so that all expected replies are delivered as a single message if replies are received within the allotted time, or not at all if the specified timeout period has expired. This function is implemented using a pair of cooperating nodes; the SpAggregateCreate node and SpAggregateReply node.

It is intended that the function provided by this SupportPac will be incorporated into a future release of the MQSeries Integrator product but this design is not yet finalized and the final implementation may therefore differ in some details from that presented here.

## Chapter 1. Overview

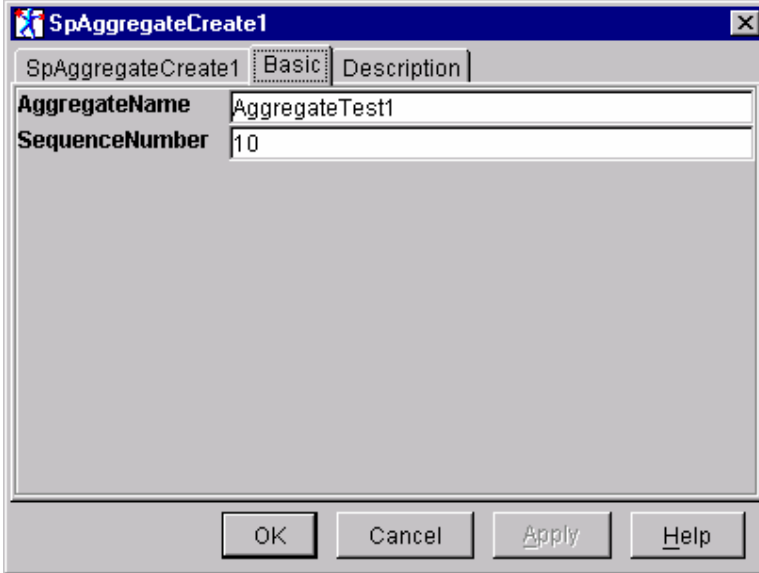
Previous versions of the aggregator function were memory only implementations. This meant that all interim message data and state data was held in in-memory structures and was lost if the Execution Group stopped whilst any aggregate instances were still active. This version of the aggregator function introduces a recovery facility that enables active aggregate instances to be recovered once the Execution Group is restarted.

The aggregator function works by monitoring each message on the outbound message flow and detecting messages that belong to the same aggregate instance. For this to happen, each path in a message flow that generates a request message for a legacy system must include an SpAggregateCreate node. The following message flow shows an input message being replicated to 3 legacy queues (L1.Q, L2.Q and L3.Q). An SpAggregateCreate node is present on each outbound flow.





The settings for the SpAggregateCreate1 node are shown below.

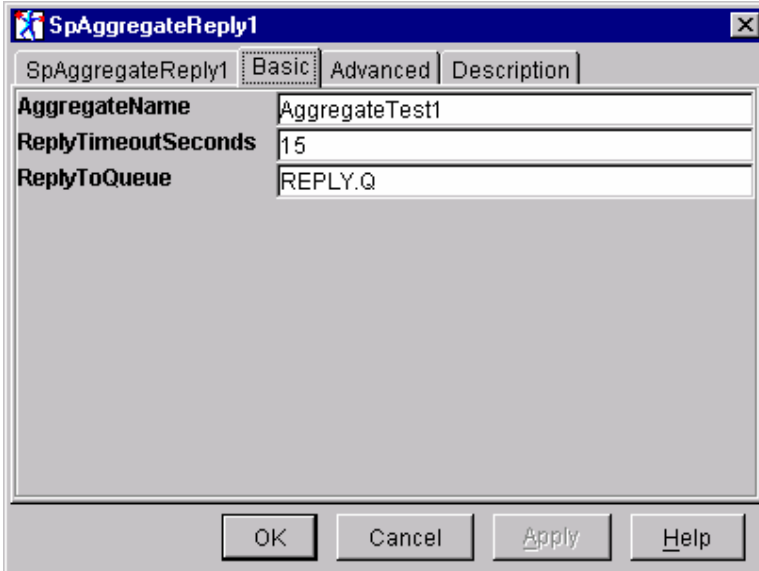


The screenshot shows a dialog box titled "SpAggregateCreate1" with a close button (X) in the top right corner. The dialog has two tabs: "Basic" (selected) and "Description". The "Basic" tab contains two input fields: "AggregateName" with the value "AggregateTest1" and "SequenceNumber" with the value "10". At the bottom of the dialog are four buttons: "OK", "Cancel", "Apply", and "Help".

Field	Value
AggregateName	AggregateTest1
SequenceNumber	10

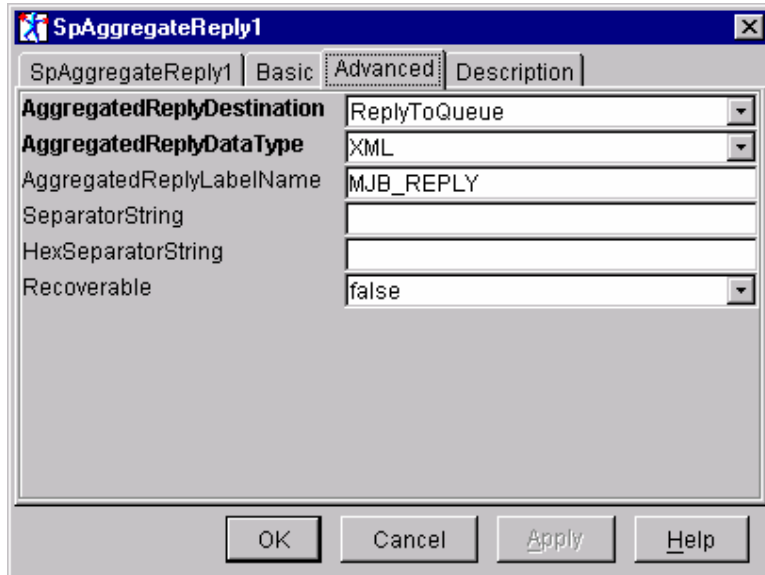
The settings for the SpAggregateCreate2 and SpAggregateCreate3 nodes is the same, except that SequenceNumber is set to 20 and 30 respectively.

The settings for the SpAggregateReply1 node basic and advanced attributes are shown below.

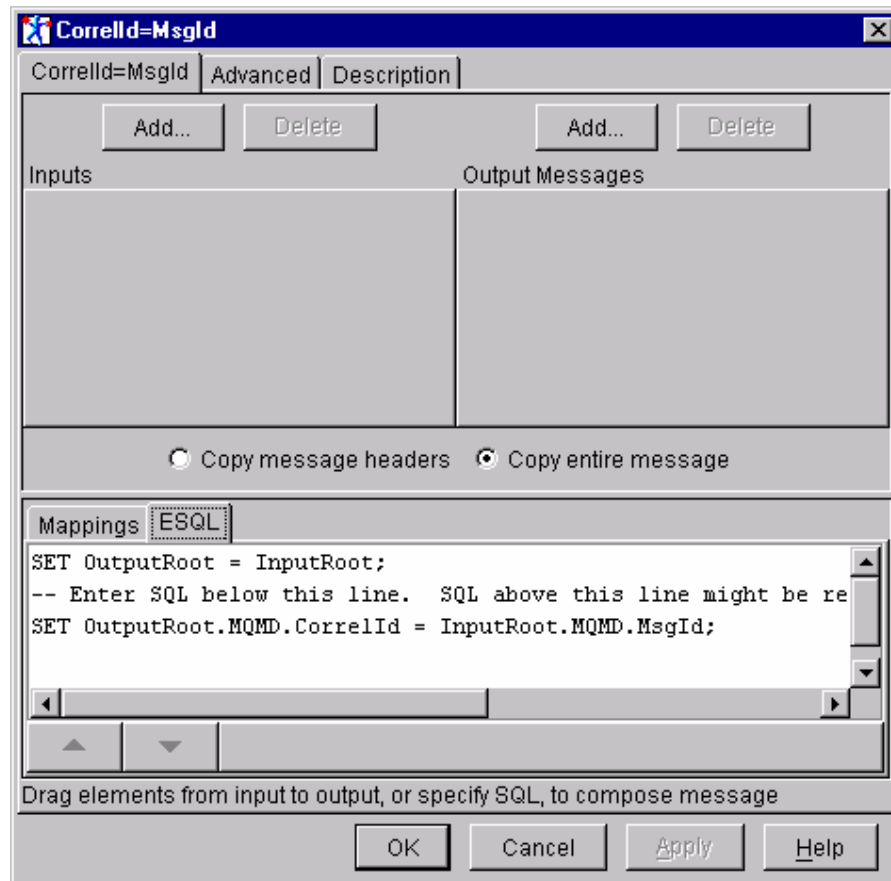


The screenshot shows a dialog box titled "SpAggregateReply1" with a close button (X) in the top right corner. The dialog has three tabs: "Basic" (selected), "Advanced", and "Description". The "Basic" tab contains three input fields: "AggregateName" with the value "AggregateTest1", "ReplyTimeoutSeconds" with the value "15", and "ReplyToQueue" with the value "REPLY.Q". At the bottom of the dialog are four buttons: "OK", "Cancel", "Apply", and "Help".

Field	Value
AggregateName	AggregateTest1
ReplyTimeoutSeconds	15
ReplyToQueue	REPLY.Q



The compute node shown in the flow is being used to set the CorrelationId to a unique value. It does this by copying the MsgId value, however this is shown only as an example of one method of setting the correlation id. Of course, you could also set the CorrelationId in the application that generates the request. For example purposes, the contents of the compute node are shown below.



A valid deployment of the aggregator function must include at **least** one SpAggregateCreate node and at **most** one SpAggregateReply node that belong to the same deployed aggregate. SpAggregateCreate and SpAggregateReply nodes are related via the AggregateName attribute which exists as a property of both types of node. In the message flow configuration shown previously, the three SpAggregateCreate nodes together with the SpAggregateReply node are deployed as part of the same Message Flow. To avoid initialisation race conditions between Message Flows in the same Execution Group, all nodes belonging to the same aggregate should be deployed in the same Message Group.

It should also be noted that more than one deployed aggregate might exist within the set of message flows deployed in an Execution Group. For example, another message flow similar to that shown earlier could be deployed and providing that the AggregateName attribute of its SpAggregateCreate nodes and its SpAggregateReply node were different, it would be treated as a separate deployed aggregate.

**NB:** Each deployed aggregate must use a different reply queue (as specified by the ReplyToQueue property). Failure to comply is likely to result in reply messages (and timer messages) being processed by the wrong SpAggregateReply node. Furthermore, a message flow containing aggregate nodes cannot be deployed to more than one Execution Group. If such a message flow needs to be deployed to more than one Execution Group, then the suggested solution is deploy a copy of the message flow to the alternate Execution Group after setting its ReplyToQueue property to a unique queue name.

The Correlld field of the message header is used to carry an aggregate identifier. All messages that are part of the same aggregate must have a common, unique value in their Correlld field. Generating this value is outside of the scope of the aggregator function. Whenever a SpAggregateCreate node encounters a message with a Correlld value not previously seen, it assumes this is the first message of a new aggregate instance and processes the message accordingly, creating a new aggregate instance.

Once a new aggregate instance has been created, subsequent messages containing the same Correlld will be added to the existing aggregate instance. Each message must also contain the same Msgld value otherwise the SpAggregateCreate node throws an exception. During initialisation of the plug in, a separate alarm thread is created to provide a timer function for the plug-in. For each deployed aggregate, a configurable attribute, ReplyTimeoutSeconds allows an aggregate timeout value in seconds to be specified. Whenever a SpAggregateCreate node processes a message, a timer is started with the specified value. For example, if ReplyTimeoutSeconds is set to 30, then a 30 second timer will be started each time an aggregate request message is processed by a SpAggregateCreate node. Apart from the initial message, all messages in an aggregate instance cause the existing aggregate instance timer to be cancelled and a new timer to be started for the instance. This means that the aggregate timer effectively starts from the moment that the **last** message in an aggregate instance is processed by a SpAggregateCreate node.

And how do we know which message is the last in an aggregate instance? Well, the straightforward answer is that we have no way of knowing which message is the last. What we do know is that SpAggregateCreate nodes have processed one or more request messages before the SpAggregateReply node has received any corresponding reply messages. To avoid race conditions, the recommended configuration for using the Aggregator function is to make the message flow outputs transactional. In this way, all messages belonging to an aggregate instance are guaranteed to have passed through the SpAggregateCreate node **before** any are processed by a legacy application. As a SpAggregateCreate node processes each message in an aggregate instance, a counter (the member count) is incremented. Thus at any time after creation, an aggregate instance will contain 1 or more members, represented by this counter. The counter can also be viewed as the number of outstanding replies.

The SpAggregateReply node keeps a corresponding tally of reply messages, and as each reply is received, it is correlated with its original request message and the corresponding aggregate instance member count is decremented. If the member count reaches zero, i.e. no outstanding replies, then the aggregate is deemed to be complete and is processed accordingly. The SpAggregateReply node generates a single output message consisting of all the reply messages concatenated together.

Depending on the setting of the configuration option `AggregatedReplyDataType`, the message is built either as a BLOB data type or as an XML message. In either case, the `CorrelId` field of the output message(s) is set to the original `MsgId`. Completed aggregates are propagated to the out terminal. Once the aggregate instance has been processed in this way, it is removed from memory. Should any subsequent request messages exist with the same `CorrelId`, they will be processed by a `SpAggregateCreate` node and treated as a separate aggregate instance.

Whenever the alarm thread is woken, it runs down its list of alarm elements looking for any which have timed out. Since the elements are held in time order, the next alarm to expire will be at the top of the list. If the alarm thread does not detect any timed out elements, it simply resets itself to wake up at the expiry time of the alarm at the top of the list.

Whenever it finds an element that has expired, it builds a simple MQ message containing the aggregate identifier and puts the message on the queue specified by the aggregate's `ReplyToQueue` property. It also changes the state of the aggregate instance by moving it from the chain of active aggregate instances to the chain of timed out aggregate instances. In this context, an expired aggregate means an aggregate instance that still has one or more outstanding replies after its time out interval has expired.

The `SpAggregateReply` node is in fact a composite node consisting of an `MQInput` node (reading from the `ReplyToQueue` queue ) and an `SpARI` node. The actual reply processing is carried out by the `SpARI` (`SpAggregateReply Internal`) node, although this is hidden within the `SpAggregateReply` composite.

As well as receiving the reply messages from the legacy systems, the `SpAggregateReply` node also receives timer messages sent to the same input queue. These are used to pass special messages to the broker thereby providing the stimulus for processing timed out aggregate instances. These messages enable to reply node to identify the aggregate that has timed out. The `SpAggregateReply` node removes the aggregate instance from the timed out chain and builds an aggregate message consisting of all the reply messages already received. The incomplete aggregate is then propagated to the `timedOut` terminal. The aggregate instance is then removed from memory. Any reply messages subsequently received for this aggregate will be propagated without change to the `lateReplies` terminal.

---

## Chapter 2. Installation

---

### SupportPac contents

The supplied zip file should be unzipped in a temporary directory. The following files and sub-directories will be created.

/source

- Makefile.NT
- Makefile.AIX
- Makefile.Solaris
- Makefile.HPUX
- aggregator.c
- destination.c
- plugin.c
- reply.c
- request.c
- alarm.c
- retcodes.c
- utilities.c
- log.c
- aggregator.h
- aggregator\_retcodes.h
- aggregator\_constants.h
- plugin.h
- unicode.h
- threads.h
- uuid.h
- log.h
- rc.h
- trace\_publ.h
- trace\_defs.h
- trace\_func.h
- trace\_data.h
- MQAggDriver.c
- MQAggLegacy.c
- samples.mak.NT
- samples.mak.AIX
- samples.mak.Solaris
- samples.mak.HPUX

/NT/help

MessageProcessingNodeType\_SpAggregateCreate.htm

MessageProcessingNodeType\_SpAggregateReply.htm

/NT/bin

aggregator.lil

v201/aggregator.lil

aggDriver.exe

aggLegacy.exe

/NT/messages

MQSIV2\_aggregator.msg

MQSIV2\_aggregator\_msg.h

MQSIV2\_aggregator.dll

Makefile

/NT/objects

traceinit.obj

trace.obj

error.obj

uuid.obj

unicode.obj

threads.obj

/NT/config

MJBAggregateCreate

MJBAggregateCreate.wdp

MJBAggregateReply

MJBAggregateReply.wdp

MJBARI

MJBARI.wdp

/NT/images

SpAggregateCreate.gif

SpAggregateCreate30.gif

SpAggregateCreate42.gif

SpAggregateCreate58.gif

SpAggregateCreate84.gif

SpAggregateReply.gif

SpAggregateReply30.gif

SpAggregateReply42.gif

SpAggregateReply58.gif

SpAggregateReply84.gif



/AIX

/AIX/bin

aggregator.lil  
v201/aggregator.lil  
aggDriver  
aggLegacy

/AIX/messages

MQSIV2\_aggregator.cat  
MQSIV2\_aggregator.msg  
MQSIV2\_aggregator\_msg.h

/AIX/objects

traceinit.o  
trace.o  
error.o  
uuid.o  
threads.o  
unicode.o

/Solaris

/Solaris/bin

aggregator.lil  
v201/aggregator.lil  
aggDriver  
aggLegacy

/Solaris/messages

MQSIV2\_aggregator.cat  
MQSIV2\_aggregator.msg  
MQSIV2\_aggregator\_msg.h

/Solaris/objects

traceinit.o  
trace.o  
error.o  
uuid.o  
threads.o  
unicode.o

/HPUX

/HPUX/bin



```
    aggregator.lil
    v201/aggregator.lil
    aggDriver
    aggLegacy
/ HPUX /messages

    MQSIV2_aggregator.cat
    MQSIV2_aggregator.msg
    MQSIV2_aggregator_msg.h
/ HPUX /objects

    traceinit.o
    trace.o
    error.o
    uuid.o
    threads.o
    unicode.o

license2.txt

ia72.pdf
```

---

## Prerequisites

This SupportPac provides a Plug-in node to be used with the IBM MQSeries Integrator for Windows NT and 2000- V2.0.1, IBM MQSeries Integrator for AIX - V2.0.1, IBM MQSeries Integrator for Solaris - V2.0.1 and IBM MQSeries Integrator for HP-UX V2.0.2 and above. For normal use, there are no other pre-requisite products other than those required by MQSeries Integrator V2.1 itself.

---

## Supported platforms

This SupportPac has been developed for and tested on

- Microsoft Windows NT environment
- IBM AIX environment
- Sun Solaris environment
- HP-UX 11 environment

---

## Installing the plug-in nodes on NT

1. Unzip the packaged files into a temporary directory.
2. Copy the message catalogue file **NT\messages\MQSIV2\_aggregator.dll** to a directory of your choice, e.g. <MQSI root>\messages.

3. Add an entry for the message catalogue to the registry. Use regedit to add an entry to the registry under...

```
HKEY_LOCAL_MACHINE
  SYSTEM
    CurrentControlSet
      Services
        EventLog
          Application
```

Create a new entry with the following details

```
MQSIV2_aggregator
  (default)          (value not set)
  EventMessageFile  <fully qualified name of MQSIV2_aggregator.dll>
  TypesSupported    0x00000007 (7)
```

4. Copy the file **NT\bin\aggregator.lil** to the MQSeries Integrator bin directory, e.g. <MQSI\_root>\bin.
5. Create the local queue SYSTEM.BROKER.AGGREGATOR.RECOVERY.QUEUE if you are going to deploy message flows containing recoverable aggregate configurations.
6. Restart the broker and check for plug-in initialisation messages in Event log

---

### Installing the plug-in nodes on AIX

1. Unzip the packaged files into a temporary directory.
2. Copy the message catalogue file **AIX\messages\MQSIV2\_aggregator.cat** to a directory specified by the NLSPATH setting, e.g. <MQSI\_root>/messages.
3. Copy the file **AIX\bin\aggregator.lil** to the MQSeries Integrator lil directory (<MQSI\_root>/lil).
4. Create the local queue SYSTEM.BROKER.AGGREGATOR.RECOVERY.QUEUE if you are going to deploy message flows containing recoverable aggregate configurations.
5. Restart the broker and check for plug-in initialisation messages in the syslog.

---

### Installing the plug-in nodes on Solaris

1. Unzip the packaged files into a temporary directory.
2. Copy the message catalogue file **Solaris\messages\MQSIV2\_aggregator.cat** to a directory specified by the NLSPATH setting, e.g. /usr/lib/locale/<locale>/LC\_MESSAGES where <locale> is the locale under which the machine is running or 'C' if none is set.
3. Copy the file **Solaris\bin\aggregator.lil** to the MQSeries Integrator lil directory (<MQSI\_root>/lil).
4. Create the local queue SYSTEM.BROKER.AGGREGATOR.RECOVERY.QUEUE if you are going to deploy message flows containing recoverable aggregate configurations.
5. Restart the broker and check for plug-in initialisation messages in the syslog.

---

### Installing the plug-in nodes on HP-UX

1. Unzip the packaged files into a temporary directory.

2. Copy the message catalogue file **HPUX\messages\ MQSIV2\_aggregator.cat** to a directory specified by the NLSPATH setting, e.g. /usr/lib/locale/<locale>/LC\_MESSAGES where <locale> is the locale under which the machine is running or 'C' if none is set.
3. Copy the file **HPUX\bin\agggregator.lil** to the MQSeries Integrator lil directory (<MQSI\_root>/lil).
4. Create the local queue SYSTEM.BROKER.AGGREGATOR.RECOVERY.QUEUE if you are going to deploy message flows containing recoverable aggregate configurations.
5. Restart the broker and check for plug-in initialisation messages in the syslog.

---

## Integrating the plug-in node into the Windows Control Center

1. Unzip the packaged files into a temporary directory.
2. Change to the **NT\images** directory and copy its contents to <MQSI\_root>\Tool\images
3. Change to the **NT\config** directory and copy its contents to <MQSI\_root>\tool\repository\private\<machine name>\<Queue Manager name>\MessageProcessingNodeType
4. Change to the **NT\help** directory and copy its contents to <MQSI\_root>\tool\help\com\isv
5. Start the MQSeries Integrator Control Center and display the MessageFlows panel. Right click on IBM Primitives and select **Add to Workspace**, then **Message Flow**. Select the SpAggregateCreate, SpAggregateReply and SpARI nodes from the displayed list and add them to the palette.
6. Check in all new node types
7. Remove (do not Delete) the ARI node from the workspace (this is an optional step)

---

## Installation verification

Create a message flow that is similar to the one shown in the Overview section.

To simulate the backend legacy applications, run the aggLegacy program. By default, this will process messages arriving on the legacy queues L1.Q, L2.Q and L3.Q. You can specify your own queue manager and queues by invoking the application in the following way

```
aggLegacy <queueManagerName> <queueName 1> <queueName 2> ...<queueName n>
```

Up to 20 queue names may be specified.

Add messages to the input queue I1.Q by running the aggDriver program. This program takes a single parameter that specifies the number of input messages to be generated. Aggregates that complete in time will be written to the O1.Q while those that are not processed in the allotted time will be written to the TIMED\_OUT.Q. Reply messages arriving after the allotted time has expired will be written to the LATE\_REPLY.Q.

Both source for both the aggDriver and aggLegacy programs is supplied along with an appropriate makefile for the target Operating System, e.g. samples.mak.NT to enable you to easily rebuild them.

---

## Chapter 3. Nodes Reference

---

### SpAggregateCreate Node

An Aggregate Create node takes a message as part of a group of messages being ‘fanned out’ to multiple queues. This grouping is not to be confused with the MQSeries group capability.

It works in conjunction with a **SpAggregateReply** node, storing certain information in the aggregate control blocks to allow the SpAggregateReply node to aggregate (combine) the replies from the applications into one reply back to a requesting application.

For each message sent to a back end application, there must be one SpAggregateCreate node before the MQOutput node.

The multiple SpAggregateCreate nodes in a flow are matched with a single SpAggregateReply node.

The **first** message in a group will cause a SpAggregateCreate node to store the following:

- An aggregate identifier - retrieved from the message correlation id which is required for this purpose
- The time to wait for the replies - obtained from the associated SpAggregateReply node

Every message in a group will cause an SpAggregateCreate node to store the following:

- An aggregate counter - incremented for every message in the group
- The message id - this could be the message id of the initial request message or a generated message id from a previous Compute node. All messages in the group must have the same message ids
- The requesting application Reply To Queue
- The requesting application Reply To Queue Manager
- A message sequence number for ordering the replies - obtained from a property on the SpAggregateCreate node. Each SpAggregateCreate node in the flow can have the same or different sequence numbers. If the sequence numbers are the same, this indicates that the SpAggregateReply node does not have to be concerned with ordering the messages in the aggregated reply. Furthermore, if the output format for the aggregated message is XML, it will not be possible to generate unique tag names for each of the replies.

Additionally, each message that flows through an SpAggregateCreate node will set/reset and initiate the timer. This allows for any number of messages in a group.

If the aggregate is configured to be recoverable (SpAggregateReply node property ‘recoverable=true’), the SpAggregateCreate processing will write a transactional ‘aggregate member’ log message to the SYSTEM.BROKER.AGGREGATE.RECOVERY.QUEUE.

Before leaving the SpAggregateCreate node, the Reply To Queue is automatically set to the name that is specified in the SpAggregateReply node property.

Messages flow in to the SpAggregateCreate node via the **in** terminal and normally out via the **out** terminal. If the node detects an error, the message will be routed to the **failure** terminal.

NB: If the message flow contains recoverable aggregates, the message flow inputs and outputs must be defined to be recoverable, thereby ensuring that the transactional log messages are committed as part of the message flow unit-of-work.

<b>SpAggregateCreate node terminals</b>	
<b>Terminal</b>	<b>Description</b>
In	The input terminal that accepts a message for processing by the node
Out	The output terminal to which the message is normally routed
Failure	The output terminal to which the message is routed if there is an error during the node processing

### ***SpAggregateCreate node properties***

These properties are displayed when you right click an SpAggregateCreate node entry in the Message Flow Types pane, and click **Properties**. The values displayed are the default properties for this instance of the node. They cannot be edited when displayed from the Message Flow Types pane.

#### **AggregateName**

Every set of SpAggregateCreate nodes in the flow and the associated SpAggregateReply node should have the same identifier

#### **SequenceNumber**

Identifier stored by the SpAggregateCreate node and used by the SpAggregateReply node to build the reply message. For BLOB format reply messages, this value is used to determine the relative position of each reply message in the aggregated reply, thereby allowing programs to easily map a BLOB message. If the order of the replies is irrelevant, this number can be the same on all SpAggregateCreate nodes in the flow.

For XML format reply messages, this value is used to build an XML tag name to contain the reply message.

### ***Configuring the SpAggregateCreate node***

For a description of the properties of the SpAggregateCreate node and their possible values, see “Aggregate Create node properties” above.

To configure an SpAggregateCreate node:

1. In the Message Flow Definition pane, right click the symbol of the SpAggregateCreate node you want to configure and click **Properties**, then click **Basic**. The **SpAggregateCreate node** dialog is displayed.
2. In the **SpAggregateCreate node** dialog, type values for those properties that you want to set.
3. If you want to provide a description of this instance of the SpAggregateCreate node (which is recommended if you want other Control Center users to be able to make use of it), click the **Description** tab of the **SpAggregateCreate node** dialog. Type a short description, or a long description, or both.
4. Click **OK** to finish configuring this SpAggregateCreate node.

---

## SpAggregateReply Node

An SpAggregateReply node is the corresponding node to the SpAggregateCreate node(s).

It aggregates (combines) replies from back end applications into one reply back to a requesting application using the details stored in the database by the SpAggregateCreate node and additional parameters set in its own properties.

Every reply message that passes into the SpAggregateReply node will cause it to perform the following:

- Locate the correct set of stored details using the correlation id of the reply message (this assumes that the back end application has used the message id of the incoming message to set the correlation id of the reply message)
- Decrement the aggregate counter
- Set the Reply To Queue from the one stored (since the replies are being combined into one, all Reply To Queue names should be the same)
- Set the Reply To Queue Manager from the one stored (since the replies are being combined into one, all Reply To Queue Manager names should be the same)
- For BLOB type aggregated reply messages, determine the position of this reply in the combined reply based on the message sequence number (optional)
- For BLOB type aggregated reply messages, separate each reply with either a character or hexadecimal string (optional)
- For XML type aggregated reply messages, wrap each reply with a tag the name of which is derived from the AggregatedReplyLabelName property concatenated with the reply sequence number
- For XML type aggregated reply messages, create a top-level tag using the AggregatedReplyLabelName value
- Set the MsgId and CorrelId of the aggregated message so that it is appropriate for processing by an MQReply node or an MQOutput node with its 'Advanced.Destination Mode' property set to replyToQueue (optional)
- Set the destination list of the aggregated message to contain the original ReplyToQ and ReplyToQMgr values (optional)
- write an 'aggregate reply' log message to the SYSTEM.BROKER.AGGREGATE.RECOVERY.QUEUE if the aggregate is configured to be recoverable.

As each message to the back end applications has passed through the SpAggregateCreate nodes, the timer is set to the value indicated in the SpAggregateReply node property. If all replies are received by the SpAggregateReply node via the **in** terminal within the time, the aggregated reply is routed out via the **out** terminal. If the timer pops before all replies are received, the replies so far are aggregated and routed out via the **timedOut** terminal. If replies arrive after the timer has popped the message will be routed out via the **lateReplies** terminal.

Just before sending the reply out of the **out**, **timedout** or **latereplies** terminals, the SpAggregateReply node will restore the Reply To Queue and Reply To Queue Manager and set the correlation id into the Message Descriptor. The correlation id is set from the message id of the request message which was stored in the database by the SpAggregateCreate node. Note: the setting of the correlation id and message id is influenced by the value specified on the 'Advanced.AggregatedReplyDestination' property of the SpAggregateReply node.

If the node detects an error, the message will be routed to the **failure** terminal.

NB: If the message flow contains recoverable aggregates, the message flow inputs and outputs must be defined to be recoverable, thereby ensuring that the transactional log messages are committed as part of the message flow unit-of-work.

<b>SpAggregateReply node terminals</b>	
<b>Terminal</b>	<b>Description</b>
in	The input terminal that accepts a message for processing by the node
out	The output terminal to which the aggregated message is routed if all replies are received before the timer pops
timedOut	The output terminal to which the aggregated message is routed if the timer pops.
lateReplies	The output terminal to which any late messages are routed if the terminal is wired to an MQOutput node
failure	The output terminal to which the message is routed if there is an error during the node processing
catch	Internally, this is wired to the catch terminal of the MQInput node reading from the ReplyToQueue.

### ***SpAggregateReply node properties***

These properties are displayed when you right click an SpAggregateReply node entry in the Message Flow Types pane, and click **Properties**. The values displayed are the default properties for this instance of the node. They cannot be edited when displayed from the Message Flow Types pane.

#### **AggregateName**

Every set of SpAggregateCreate nodes in the flow and the associated SpAggregateReply node should have the same identifier.

#### **ReplyTimeoutSeconds**

Time in seconds to wait for all replies to the SpAggregateReply node

#### **ReplyToQueue**

Name of the queue that the back end applications should reply to. The SpAggregateReply node will process reply messages put to this queue. Note: this queue is also used by the timer thread for passing 'internal' timer messages to the SpAggregateReply node.

#### **AggregatedReplyDestination**

Enumerated value indicating how the aggregated reply message will be processed. This controls the characteristics of the message generated by the SpAggregateReply node. The possible values are

- default – no special processing
- ReplyToQueue – the message will be processed by either an MQReply node or an MQOutput node with Advanced.Destination Mode set to replyToQueue
- DestinationList – the message will be processed by an MQOutput node with Advanced.Destination Mode set to destinationList

#### **AggregatedReplyDataType**

Enumerated value indicating the format for the aggregated reply message. The possible values are

- BLOB – unstructured data format (this is the default value)

- XML – the message will be built as an XML format message

If the XML option is chosen, the following points should be noted.

1. You must ensure that each reply message contains valid XML – the SpAggregateReply node assumes this to be the case and does not check the contents of the reply messages
2. The aggregated XML reply message will consist of each of the reply messages wrapped in an XML tag, the name of which is formed from a combination of the AggregatedReplyLabelName attribute and the SequenceNumber specified on the corresponding SpAggregateCreate node.
3. The collection of reply messages will be wrapped in a top-level XML tag whose name is the value specified by the AggregatedReplyLabelName.

The following example shows the structure of an aggregated reply message with the listed characteristics.

- AggregatedReplyLabelName = 'MyReply'
- reply 1 data is '<reply>data1</reply>' and its SequenceNumber value is 30
- reply 2 data is '<reply>data2</reply>' and its SequenceNumber value is 50

Aggregated Reply Message Structure
<pre> &lt;MyReply&gt;   &lt;MyReply30&gt;     &lt;reply&gt;data1&lt;/reply&gt;   &lt;/MyReply30&gt;   &lt;MyReply50&gt;     &lt;reply&gt;data2&lt;/reply&gt;   &lt;/MyReply50&gt; &lt;/MyReply&gt; </pre>
Aggregated Reply Message internal MQSI structure
<pre> (0x1000010)XML      =(   (0x1000000)MyReply =(     (0x1000000)MyReply30 =(       (0x1000000)reply =(         (0x2000000) = data1       )     )     (0x1000000)MyReply50 =(       (0x1000000)reply =(         (0x2000000) = data2       )     )   ) ) </pre>



**AggregatedReplyLabelName**

Character string used to build an XML tag to wrap each of the replies in an XML format aggregated reply message. If `AggregatedReplyDataType=XML`, this property is mandatory. The tag name is built by concatenating this value together with the sequence number for each reply. For example, if `AggregatedReplyLabelName` is set to 'REPLY' and the `SequenceNumber` for the reply is 10, then the derived tag name will be 'REPLY10'.

The value is also used 'as is' to create the top-level XML tag for the reply message. For example, `Root.XML.REPLY`.

**SeparatorString**

Character string used to separate each reply in the combined reply to the requesting application. This property is ignored if the aggregated message format is XML.

**HexSeparatorString**

Hex string used to separate each reply in the combined reply to the requesting application. This property is ignored if the aggregated message format is XML.

**Recoverable**

A boolean value indicating whether or not recovery data should be created for the aggregate. The default value is 'false' which means that the aggregate data will be held in memory only. By specifying a value of 'true', the aggregator will create recovery data for the aggregate and write the data as transactional messages to the `SYSTEM.BROKER.AGGREGATOR.RECOVERY.QUEUE`. These messages are used during the restart of the Execution Group to recover aggregates that were incomplete when the Execution Group was previously stopped.

***Configuring the SpAggregateReply node***

For a description of the properties of the `SpAggregateReply` node and their possible values, see "SpAggregateReply node properties" above.

To configure an `SpAggregateReply` node:

1. In the Message Flow Definition pane, right click the symbol of the AggregatorReply node you want to configure and click **Properties**. The **SpAggregateReply node** dialog is displayed.
2. In the **SpAggregateReply node** dialog, type values for those properties that you want to set.
3. If you want to provide a description of this instance of the `SpAggregateReply` node (which is recommended if you want other Control Center users to be able to make use of it), click the **Description** tab of the **SpAggregateReply node** dialog. Type a short description, or a long description, or both.
4. Click **OK** to finish configuring this `SpAggregateReply` node.

---

## Tracing the aggregator plug-in nodes

To trace execution of the plug-in nodes, set the AGGREGATOR\_PLUGIN\_TRACE environment variable (system variable on NT) and reboot machine **before** restarting broker. Settings for trace are as follows.

AGGREGATOR\_PLUGIN\_TRACE =

- f *traceOutputFileName* - name of file to write trace to
- t - include time stamp on trace entries.
- i - include process and thread id on entries
- c - commit (flush) entries to file after every write
- l - trace level to output (see trace values below)
- a - append trace to existing trace file

Valid trace level settings are

- TRACE\_NONE
- TRACE\_ENTRY\_EXIT
- TRACE\_ERROR
- TRACE\_WARNING
- TRACE\_INFO
- TRACE\_SYSTEM\_ALL
- TRACE\_ALL

The following setting results in comprehensive tracing and will be sufficient in most cases.

```
AGGREGATOR_PLUGIN_TRACE=-f <name of output file> -i -c -l TRACE_SYSTEM_ALL
```

End of Document