MQSeries Integrator - Timer Plug-In Version 1.1

11th September 2002

Mike Brady
Senior Consulting IT Specialist
IBM
Australia

mjbrady@au1.ibm.com

Take Note!

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

Second Edition, September 2002

This edition applies to Version 1.1 of *MQSeries Integrator – Timer Plug-In* and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

Notices

Trademarks and service marks

Summary of Amendments

Date	Changes
6th July 2001	Initial internal release
11th July 2001	Added support for Solaris platform
19th July 2001	Added support for Platform scope timers
11th August 2001	Added support for HP-UX platform
3rd September 2001	Released as a SupportPac
11th September 2002	Added improved heap management logic and made garbage collection self-regulating
	Add preprocessor directives to allow source to build correctly against v2.1, v2.0.2 and v2.0.1 releases
	Shipped v2.0.1 and post v2.0.1 binaries

Preface

When implementing complex solutions with MQSeries Integrator V2, a common requirement is to initiate a message flow based on a time interval. For example, there may be processing that needs to be performed at regular intervals, or the arrival of a timer message might be used to check for completion of a previous operation. One solution to this problem is to write a custom application to manage the generation of such messages and pass them to MQSI, however, this involves a management overhead, that is. the application must be started, stopped and monitored.

The MQSI V2 Timer function provides an alternative mechanism for generating event or timer messages without the need for any external timer application. It is quick, intuitive, simple to implement and is highly flexible in the ways it can be deployed. The function is implemented using a pair of cooperating nodes; the TimerCreate node and TimerCancel node. The implementation allows timers to be visible in a variety of scopes, from Message Flows in the same Execution Group to Message Flows in any Execution Group in any Broker instance running on the same platform. Furthermore, the implementation is multi-threaded so it will work equally well for deployments where Message Flows have been configured with multiple worker threads (Additional Instances > 0).

Chapter 1. Overview

A timer instance is created when a TimerCreate node processes a suitable timer creation message. A timer instance can be configured so that it expires just once and is then deleted from the system, or it can be configured so that it expires a specified number of times. Whenever the timer expires, it generates a timer event message and sends it to the destination queue specified by the TimerCreate node. The timer instance can also be associated with data extracted from the message that triggered its creation. In this case, any extracted data will be included in the body of the timer event messages generated by the timer instance. Furthermore, the CorrelId of the timer event messages will always be set to the MsgId of the creation message.

Once created, a timer instance will exist until one of the following conditions is satisfied

- it expires the specified number of times, following which it is automatically removed from the system

- it is explicitly cancelled by a matching TimerCancel node

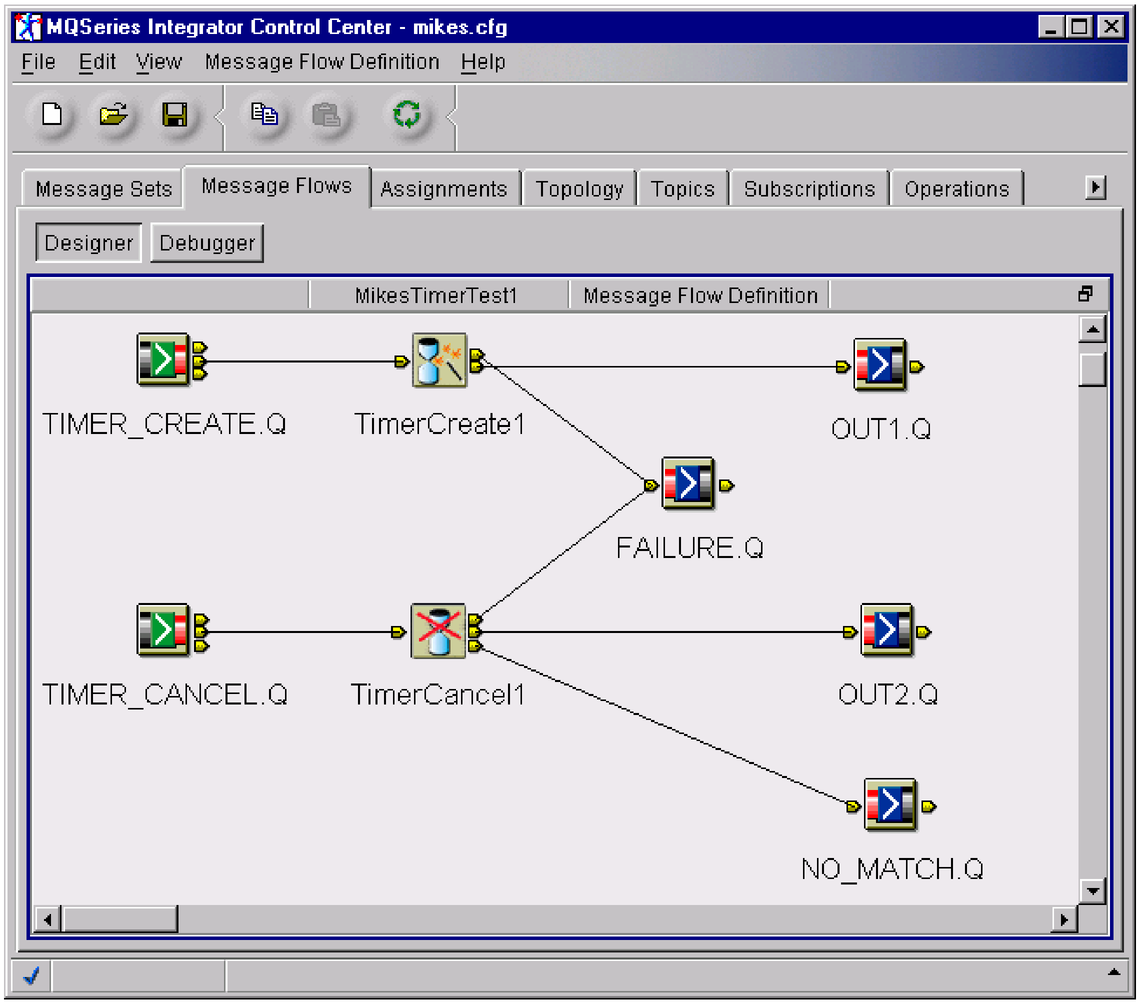
The TimerCancel node provides a mechanism for cancelling an existing timer instance. The TimerCancel node processes a timer cancellation message and identifies the timer instance to be cancelled by matching it with attributes extracted from the cancellation message.

By default, when a timer instance is created, it is identified by its MsgId and CorrelId values. Alternatively, you may specify a name for a timer instance in which case this is used exclusively to identify the timer instance. The name can be either a literal value or it can be a value derived from the creation message. Similarly, when cancelling a timer instance, a name may be provided either explicitly (a literal value defined in the TimerCancel node) or derived from the cancellation message. In order to cancel a named timer instance, the TimerCancel node must also provide a matching name.

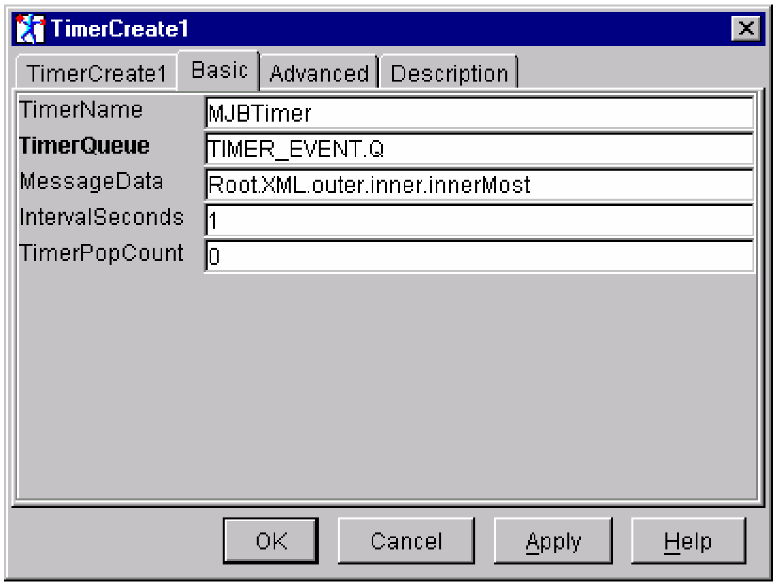
The following message flow shows a timer instance being created by putting a timer creation message on the TIMER_CREATE.Q. The message is then processed by a TimerCreate node and if the timer instance is created successfully, the creation message is passed on to the OUT1.Q (via the TimerCreate 'out' terminal). All other errors result in the creation message being propagated to the 'failure' terminal.

In this example, timer event messages created by the timer instance are sent to the TIMER_EVENT.Q (this is not shown in the message flow).

To cancel the timer instance, a cancellation message is put on the TIMER_CANCEL.Q and is then processed by the TimerCancel node. If the timer instance is successfully cancelled, the cancellation message is passed on to the OUT2.Q. If no matching timer instance can be located, the cancellation message is passed to the NO_MATCH.Q (via the TimerCancel 'noMatch' terminal). All other errors result in the cancellation message being propagated to the 'failure' terminal.



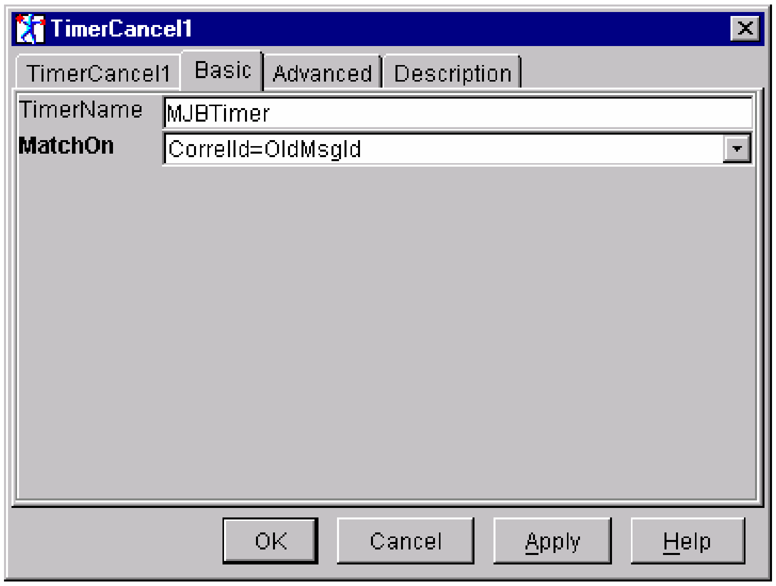
The properties window for the TimerCreate node (TimerCreate1) is shown below.



TimerName	MJBTimer
TimerQueue	TIMER_EVENT.Q
MessageData	Root.XML.outer.inner.innerMost
IntervalSeconds	1
TimerPopCount	0

In this example, the timer instance has a name 'MJBTimer'. The timer event messages will be sent to the queue 'TIMER_EVENT.Q' and they will contain the data extracted from the syntax element specified by the path 'Root.XML.outer.inner.innerMost'. The timer instance is configured to expire in 1 second and has its maximum number of expiries (TimerPopCount) set to 0 (zero) which means INFINITE. Assuming that the timer instance is successfully created, then it will continue to generate an event message every second until a TimerCancel node explicitly cancels it.

The properties window for the corresponding TimerCancel node (TimerCancel1) is shown below. Since the timer instance was created with a name, the TimerCancel node must also specify the same name. In this case, the (default) value specified by the MatchOn attribute are ignored.



TimerName MJBTimer
MatchOn CorrelId=OldMsgId

The timer event messages generated by the timer instance **always** contain XML format data. In this example, the body will contain the data element and contents extracted from the syntax element 'Root.XML.outer.inner.innerMost'. As an example, assume that the creation message contained the following message body.

"<outer><tag1>data1</tag1><tag2>data2</tag2><tag3>data3</tag3><inner>…

…**<innerMost><tag5>someTimerData</tag5></innerMost>**…

…</inner><tag4>data4</tag4></outer>"

The timer event messages will contain the following.

"<TimerEventData>**<innerMost><tag5>someTimerData</tag5></innerMost>**…

…</TimerEventData>"

If nothing was specified for the MessageData attribute of the TimerCreate node, the event message body will consist of '**<TimerEventData/>**'.

The correspondence between the structures of the creation message and the event message can be seen more clearly in the following diagrams.

Creation Message

```
(0x1000010)XML       = (
   (0x1000000)outer = (
     (0x1000000)tag1  = (
       (0x2000000) = ‘data1’
     )
     (0x1000000)tag2  = (
       (0x2000000) = 'data2'
     )
     (0x1000000)tag3  = (
       (0x2000000) = 'data3'
     )
     (0x1000000)inner = (
       (0x1000000)innerMost = (
         (0x1000000)tag5 = (
           (0x2000000) = 'someTimerData'
         )
       )
     )
     (0x1000000)tag4  = (
       (0x2000000) = 'data4'
     )
   )
 )
```

Timer Event Message

```
(0x1000010)XML       = (
   (0x1000000)TimerEventData = (
     (0x1000000)innerMost  = (
       (0x1000000)tag5  = (
         (0x2000000) = 'someTimerData'
       )
     )
   )
 )
```

Although the content of the timer event message is always an XML format message, the data copied from the creation message can be in any format. Any non-character data, such as BLOB data, will be converted into a format suitable for inclusion in an XML message. Datatypes are converted according to the following rules.

Data Type	**Conversion**
DECIMAL	Converted to string, e.g. 23.4 is converted to ’23.4’
INTEGER	Converted to string value, e.g. –2345 is converted to ‘-2345’
FLOAT	Converted to string, e.g. 23.4 is converted to ’23.4’
CHARACTER	No conversion
BLOB	Converted to a hexadecimal string
BIT	Converted to a hexadecimal string
DATE	Converted to ‘yyyy-mm-dd hh:mm:ss.sss’ format
GMTTIME	Converted to ‘hh:mm:ss.sss’ format
GMTTIMESTAMP	Converted to ‘yyyy-mm-dd hh:mm:ss.sss’ format
TIME	Converted to ‘hh:mm:ss.sss’ format
TIMESTAMP	Converted to ‘yyyy-mm-dd hh:mm:ss.sss’ format
BOOLEAN	‘TRUE’ or ‘FALSE’

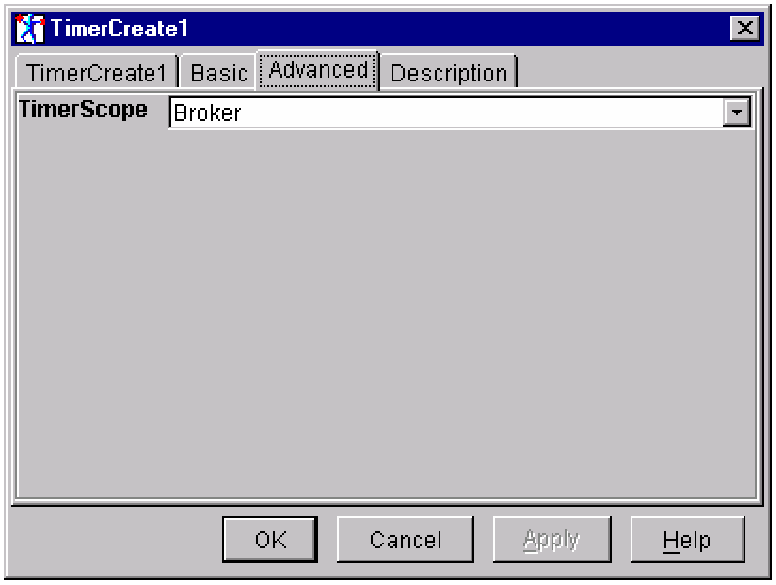
By default, a timer instance exists within the scope of the Execution Group. When processing a TimerCancel node, the search is contained within this scope, i.e. only timer instances created by Message Flows within the same Execution Group are considered.

However, it is also possible to extend the scope of a timer instance so that it becomes visible to all Execution Groups within the scope of the Broker or all Execution Groups within all Brokers within the scope of the current platform. This option is available under the Advanced tag of the properties window. The TimerScope attribute allows either **ExecutionGroup**, **Broker** or **Platform** to be selected (the default is ExecutionGroup).

If the scope is set to Broker, then whenever a TimerCancel node is executed, the search will include not only local timer instances, i.e. those created within the same Execution Group, but will also include any global timer instances created by other Execution Groups within the same broker. If the scope is set to Platform, the scope of the search is increased include global timer instances created by Execution Groups in any Broker running on the same platform. In order for a timer instance to be treated as global, it must have been created by a TimerCreate node with TimerScope=Broker or TimerScope=Platform.

It should be noted that whenever a timer is created, it should be unique within the specified scope. This means that for timers with a name, any existing timer with the same name in scope is replaced. For un-named timers, any timer within scope containing the same MsgId AND CorrelId is replaced.

The properties window below demonstrates one of the global settings of the TimerScope attribute.



TimerScope Broker

By default, the size of the shared memory segment that will be allocated is 64 Kbytes. This is the minimum permitted size, however it can be increased should more storage be required. To modify the size of the shared segment, you must set the environment variable TIMER_SHARED_MEMORY_KSIZE to the required number of Kbyte blocks.

You will probably need to run with your configuration and observe its behaviour in order to determine the optimal size of the shared memory segment, i.e. adjusting the shared memory size up or down until the system runs successfully with the minimum size segment. Some additional buffer should be included to allow for unexpected peaks in activity. Should the system run short on shared memory, a message will be written to the system log indicating this, and the Execution Group will be terminated.

Should you wish to calculate your shared memory requirements more methodically, as a rule of thumb, each global timer instance will require the following storage:

- 200 bytes for control block structure
- Storage for Execution Group name, calculated as length of Execution Group name + 1
- Storage for Message Flow name, calculated as (length of Message Flow + 1) x 2
- Storage for the TimerName (if specified), calculated as (length of TimerName +1) x 2
- Storage for saved data – the number and type of syntax element nodes that need to be saved will determine this. Using the example saved data shown previously, the contents would consist of

    1. Control block for ‘innerMost’ (approx. 70 bytes)
    2. The element name ‘innerMost’. This is stored in Unicode format and together with a field denoting its length. In this case it would require (9 x 2) + 4 bytes.
    3. Control block for ‘tag5’ (approx. 70 bytes)
    4. The element name ‘tag5’. This is stored in Unicode format and together with a field denoting its length. In this case it would require (4 x 2) + 4 bytes.
    5. Data in element in multibyte format (13 bytes, i.e. length of ‘someTimerData’) together with its length field. In this case 13 + 4 bytes

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
timer.c
plugin.c
cancel.c
create.c
alarm.c
retcodes.c
utilities.c
timer.h
timer_retcodes.h
timer_constants.h
plugin.h
unicode.h
threads.h
rc.h
trace_publ.h
trace_defs.h
trace_func.h
trace_data.h
timerTest.c
samples.mak.NT
samples.mak.AIX
samples.mak.Solaris
samples.mak.HPUX

/NT/help

MessageProcessingNodeType_TimerCreate.htm
MessageProcessingNodeType_TimerApply.htm

/NT/bin

timer.lil

v201/timer.lil

timerTest.exe

/NT/messages

MQSIV2_timer.msg

MQSIV2_timer_msg.h

MQSIV2_timer.dll

Makefile

/NT/objects

traceinit.obj

trace.obj

error.obj

unicode.obj

threads.obj

heap.obj

shmem.obj

/NT/config

TimerCreate

TimerCreate.wdp

TimerCancel

TimerCancel.wdp

/NT/images

TimerCreate.gif

TimerCreate 30.gif

TimerCreate 42.gif

TimerCreate 58.gif

TimerCreate 84.gif

TimerCancel.gif

TimerCancel 30.gif

TimerCancel 42.gif

TimerCancel 58.gif

TimerCancel 84.gif

/AIX

/AIX/bin

timer.lil
v201/timer.lil
timerTest

/AIX/messages

MQSIV2_timer.cat
MQSIV2_timer.msg
MQSIV2_timer_msg.h

/AIX/objects

traceinit.o
trace.o
error.o
threads.o
unicode.o
heap.o
shmem.o

/Solaris

/ Solaris /bin

timer.lil
v201/timer.lil
timerTest

/ Solaris /messages

MQSIV2_timer.cat
MQSIV2_timer.msg
MQSIV2_timer_msg.h

/ Solaris /objects

traceinit.o
trace.o
error.o
threads.o
unicode.o
heap.o
shmem.o

/HPUX

/ HPUX /bin

timer.lil

v201/timer.lil

timerTest

/ HPUX /messages

MQSIV2_timer.cat

MQSIV2_timer.msg

MQSIV2_timer_msg.h

/ HPUX /objects

traceinit.o

trace.o

error.o

threads.o

unicode.o

heap.o

shmem.o

license2.txt

ia0k.pdf

Prerequisites

This SupportPac provides a Plug-in node to be used with the IBM MQSeries Integrator for Windows NT or 2000 - V2.0.1, IBM MQSeries Integrator for AIX - V2.0.1, IBM MQSeries Integrator for Solaris - V2.0.1 and above and IBM MQSeries Integrator for HP-UX - V2.0.2. 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 environment

Installing the plug-in nodes on Windows NT

1. Unzip the packaged files into a temporary directory.
2. Copy the message catalogue file **NT\messages\MQSIV2_timer.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_timer
  (default)             (value not set)
  EventMessageFile   <fully qualified name of MQSIV2_timer.dll>
  TypesSupported        0x00000007 (7)
```

4. Copy the file **NT\bin\timer.lil** to the MQSeries Integrator bin directory, e.g. <MQSI_root>\bin.

5. Set the TIMER_SHARED_MEMORY_KSIZE environment variable if required.

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\bin\MQSIV2_timer.cat** to a directory specified by the NLSPATH setting, e.g. <MQSI_root>/messages.

3. Copy the file **AIX\bin\timer.lil** to the MQSeries Integrator lil directory (<MQSI_root>/lil).

4. Set the TIMER_SHARED_MEMORY_KSIZE environment variable if required.

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_timer.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\timer.lil** to the MQSeries Integrator lil directory (<MQSI_root>/lil).

4. Set the TIMER_SHARED_MEMORY_KSIZE environment variable if required.

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_timer.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\timer.lil** to the MQSeries Integrator lil directory (<MQSI_root>/lil).

4. Set the TIMER_SHARED_MEMORY_KSIZE environment variable if required.

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

   Note: create this directory if it does not already exist.
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 TimerCreate and TimerCancel nodes from the displayed list and add them to the palette.
6. Check in all new node types

Installation verification

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

The sample program timerTest creates a timer instance, waits for 2 timer event messages and then issues a timer cancel request. By default, this will use the following queues for its processing, TIMER_CREATE.Q, TIMER_CANCEL.Q and TIMER_EVENT.Q, however, you can specify your own queue manager and queues by invoking the application in the following way

```
timerTest <queueManagerName> <timerCreateQueue> <timerCancelQueue> <timerEventQueue>
```

Using the example configuration, the timer expiry interval is set to 1 second. Alternatively, you can have the timer plug-in extract the expiry value from the creation message. The sample program generates a message that contains a 5 second expiry. By setting IntervalSeconds to **Root.XML.outer.interval** this value will be extracted and used.

The source for the timerTest program is supplied along with an appropriate makefile for the target Operating System, e.g. samples.mak.NT to enable you to easily rebuild it.

Chapter 3. Nodes Reference

TimerCreate Node

A TimerCreate node is responsible for creating a timer instance. The characteristics of the timer instance are specified either as constant values or as values to be derived from the creation message. Such characteristics include expiry interval, timer name, timer lifetime and data to be included in timer event messages.

Once created, a timer instance will generate an event message every time its expiry interval is reached. It will continue to generate messages in this way until its maximum lifetime is reached or it is explicitly cancelled by a corresponding TimerCancel node. The lifetime of a timer instance is determined by the number of times its expiry interval can be reached. A timer can also be configured so that its lifetime is never exceeded, in which case, the only way to end the timer is via a TimerCancel node.

Event messages generated by a timer instance can optionally contain significant data, thereby providing applications with a mechanism for passing timer event information. The data can be a constant value or may be derived from the body of the creation message.

By default, the scope of a timer instance is local, i.e. it is visible only within the Execution Group which created it. However, it can also be configured to have global scope, in which case it is visible to all Execution Groups within the broker domain. If the timer is global and it is a named timer, there can only be one active instance of such a timer in the broker domain. When such a timer instance is created, any existing global timer with the same name is first set to an expired state and is then deleted.

Each message processed by a TimerCreate node will cause the TimerCreate node to perform the following actions:

- Create a new timer instance and save the Msgid and CorrelId
- If specified, locate any syntax element specified by MessageData property. If the element is not found, or does not contain any data value, the message is propagated to the failure terminal.
- If specified, serialise the data at the located syntax element and store in timer instance in process storage if local timer or in shared memory storage if global timer
- Add the timer to required indexes
- If the timer has a name, remove any existing timer within scope with the same name else remove any existing timer within scope with the same MsgId and CorrelId
- If a IntervalSeconds specified a non-zero value, or resolved to a non-zero value, add the timer to the alarm chain

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

TimerCreate node terminals	
Terminal	**Description**
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

TimerCreate node properties

These properties are displayed when you right click a TimerCreate 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.

TimerName

This optional value assigns a name to the timer instance. Timer instances that are created with a name can only be cancelled by a TimerCancel node which specifies the same name, i.e. they cannot be matched via MsgId and/or CorrelId.

If a name is specified, it can be either a literal value (as shown in the example flow) or it can be a name derived from the creation message.

TimerQueue

Specifies the name of the queue to which timer event messages should be sent. It is a mandatory attribute.

MessageData

This optional attribute defines the data to be included in the body of the event message(s). A literal value can be specified, or the value can be derived from the creation message. If the value specified begins with either '**Root**' or '**Root.**' then it is assumed that this represents a syntax element path and is a derived value.

IntervalSeconds

Specifies the expiry interval (in seconds) of the timer instance. It can be either a literal value or it can be an integer value derived from the creation message.

A value of 0 indicates an INFINITE lifetime, which means that it will never generate an event message. If a non-zero value is specified, an alarm is set to the expiry value. Once the expiry interval has been reached, a timer event message is generated, and, if the timer instance lifetime has not been reached when the alarm expires, the expiry interval is reset.

TimerPopCount

Specifies the maximum number of times the timer expiry interval may be reached before the timer instance is deleted. The default value is 1, which means that the timer will be deleted after its initial expiry.

TimerScope

Enumerated value indicating the scope of the timer. The possible values are

- ExecutionGroup – the timer will visible to other nodes within the same Execution Group (this is the default value). In this case, timers are stored in process memory.
- Broker – the timer will be visible to nodes in any Execution Group running within the Broker. In this case, timers are stored in a shared memory segment.
- Platform – the timer will be visible to nodes in any Execution Group in all Brokers running on the same Platform. In this case, timers are stored in a shared memory segment.

Configuring the TimerCreate node

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

To configure an TimerCreate node:

1. In the Message Flow Definition pane, right click the symbol of the TimerCreate node you want to configure and click **Properties**, then click **Basic**. The **TimerCreate node** dialog is displayed.

2. In the **TimerCreate 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 TimerCreate 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 **TimerCreate node** dialog. Type a short description, or a long description, or both.

4. Click **OK** to finish configuring this TimerCreate node.

TimerCancel Node

An TimerCancel node is the corresponding node to the TimerCreate node.

It searches the available collection of timers, looking for any matching timer instances. If any matches are found, the corresponding timer instances are cancelled.

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

- Extract the Msgid and CorrelId
- Use the value specified by the MatchOn property to search the available timer instances looking for a match – only the local indexes are searched if the TimerCancel node is local, otherwise the global indexes are also searched
- If TimerName was specified, then the MatchOn property is ignored and the search is confined to looking for a timer instance with the same name. The name specified can be either a literal value or can be derived from the cancellation message.
- Cancel and delete any matching timer instances and propagate the cancellation message to the out terminal.
- If no matching timer instance was found, propagate the cancellation message to the noMatch terminal

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

TimerCancel node terminals	
Terminal	**Description**
in	The input terminal that accepts a message for processing by the node
out	The output terminal to which the message is routed if the cancellation is successful
noMatch	The output terminal to which the message is routed if no matching timer is found
failure	The output terminal to which the message is routed if there is an error during the node processing

TimerCancel node properties

These properties are displayed when you right click an TimerCancel 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.

TimerName

This optional value assigns a name to the timer instance to be cancelled. If a name is specified, then the MatchOn attribute (see below) is ignored. Only timers with a matching name will be eligible for cancellation by this node.

If a name is specified, it can be either a literal value (as shown in the example flow) or it can be a name derived from the cancellation message.

MatchOn

Enumerated value indicating how the message attributes should be used to find matching timers. The possible values are

- CorrelId=OldMsgId – the CorrelId value of the current message must match the MsgId value of the original message that created the timer (this is the default value)
- MsgId=OldMsgId – the MsgId value of the current message must match the MsgId value of the original message that created the timer
- CorrelId=OldCorrelId – the CorrelId value of the current message must match the CorrelId value of the original message that created the timer
- MsgId =OldCorrelId – the MsgId value of the current message must match the CorrelId value of the original message that created the timer
- MsgId+CorrelId=OldMsgId+OldCorrelId – the combined MsgId and CorrelId values of the current message must match the combined MsgId and CorrelId values of the original message that created the timer

TimerScope

Enumerated value indicating the scope for matching against available timers. The possible values are

- ExecutionGroup – only the local indexes, i.e. those indexes that refer to timers created with PostitScope=ExecutionGroup will be searched.
- Broker – both the local indexes, i.e. those indexes that refer to timers created with PostitScope=ExecutionGroup, and the global indexes, i.e. those indexes that refer to timers created with PostitScope=Broker will be searched.
- Platform – both the local indexes, i.e. those indexes that refer to timers created with PostitScope=ExecutionGroup, and the global indexes, i.e. those indexes that refer to timers created with PostitScope=Broker or PostitScope=Platform will be searched. If PostitScope=Platform, both the Broker and Platform indexes are searched.

Configuring the TimerCancel node

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

To configure an TimerCancel node:

1. In the Message Flow Definition pane, right click the symbol of the TimerCancel node you want to configure and click **Propertie**s.The **TimerCancel node** dialog is displayed.
2. In the **TimerCancel 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 TimerCancel 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 **TimerCancel node** dialog. Type a short description, or a long description, or both.
4. Click **OK** to finish configuring this TimerCancel node.

Tracing the timer plug-in nodes

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

TIMER_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.

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

End of Document