Basic process triggering using runmqtrm defined as a service in WebSphere MQ 7.5 and 8.0

IBM Techdoc: 7041673

http://www.ibm.com/support/docview.wss?rs=171&uid=swg27041673

Date last updated: 23-Jun-2015

Angel Rivera – rivera@us.ibm.com
IBM WebSphere MQ Support

+++ Objective

The objective of this document is to provide the step-by-step details for defining a process that will be triggered by the arrival of a message into a queue in a MQ queue manager.

In addition, the trigger monitor (runmqtrm) will be defined as a Service controlled by the queue manager, that is, when the queue manager starts, the trigger monitor will start, and when the queue manager ends, the monitor will end.

The chapters are:

Note: In June 2015, the techdoc was updated based on testing on MQ 8.0.0.3

+++ Common pitfalls

- You must use the full path of the executable file or script that is invoked by the process that is launched by the trigger monitor.

- If the trigger monitor is NOT running, then there will NOT be a trigger message.

- The trigger monitor needs to monitor the dedicated queue where the trigger messages will be generated. A common mistake is to monitor the incoming queue where the normal messages are expected to be received.

+++ Configuration

a) MQ Explorer 7.5.0.2 running in Windows 7

b) Queue manager running 7.5.0.3 in Linux Intel 32-bit
Name: QM_VER75
Hostname: veracruz.x.com
Port: 1455

b.1) June-2015:
Queue manager running 8.0.0.3 in Linux Intel 64-bit
Name: QM_80
Hostname: mosquito.x.com
Port: 1418

c) The process to be invoked will be the UNIX command:

```
touch /tmp/file.txt
```

It does not do anything really useful, but it provides a very simple way to confirm when the process launched by triggering is actually executing.

d) The trigger monitor is the one shipped with MQ: runmqtrm

+++ References from the MQ 7.5 Information Center

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.dev.doc/q026910_.htm
WebSphere MQ > Developing applications > Writing a queuing application
Starting WebSphere MQ applications using triggers

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.adm.doc/q021080_.htm
WebSphere MQ > Administering > Administering local WebSphere MQ objects
Managing objects for triggering

+++ Other references

http://www.ibm.com/developerworks/websphere/library/techarticles/1204_gupta/1204_gupta.html
Using the Dead Letter Queue Handler in WebSphere MQWebsphere MQ

http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg27008375
Webcast: How to Automate Handling of WebSphere MQ Dead Letter Messages

http://www-01.ibm.com/support/docview.wss?uid=swg27039569
Webcast: Handling undelivered messages in WebSphere MQ 7: Dead Letter Queue, Poison Messages

++
+++ Chapter 1: Defining basic triggering in a queue to launch a Process
++

This chapter describes the steps to define the basic triggering attributes in a queue to launch a Process.

++ Overview of attributes and process

In this techdoc, the triggering attributes are extremely basic and they may not be realistic for most situations, but they are very convenient for testing and for getting familiar with triggering:

```
   TRIGGER
   TRIGDPTH(1)
   TRIGTYPE(EVERY)
   INITQ(SYSTEM.DEFAULT.INITIATION.QUEUE)
   PROCESS(TOUCH.PROCESS)
```

The above attributes mean that the desired process will be triggered EVERY time that a message arrives to the queue, and that the "initiation queue" is going to be the default SYSTEM queue for that purpose.

The Process to be launched is just the UNIX "touch" command which will create a file under /tmp. If the file already exists, then the command will just update the date stamp. The idea is that by looking at the recent files created/modified in /tmp, it is possible to quickly monitor if the triggering is taking place. This will be explained more in detail later during the testing of the scenario.

For more details see:
http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.ref.adm.doc/q085690_.htm#q085690___q085690_s2
WebSphere MQ > Reference > Administration reference > MQSC reference > The MQSC commands
DEFINE queues
+ begin excerpt

TRIGGER & NOTRIGGER
Specifies whether trigger messages are written to the initiation queue, named by the INITQ parameter, to trigger the application, named by the PROCESS parameter:
TRIGGER : Triggering is active, and trigger messages are written to the initiation queue.
NOTRIGGER : Triggering is not active, and trigger messages are not written to the initiation queue.

TRIGDPTH(integer)
The number of messages that have to be on the queue before a trigger message is written, if TRIGTYPE is DEPTH. The value must be in the range 1 - 999,999,999.

TRIGTYPE
Specifies whether and under what conditions a trigger message is written to the initiation queue. The initiation queue is (named by the INITQ parameter.
This parameter is supported only on local and model queues.
FIRST : Whenever the first message of priority equal to or greater than the priority specified by the TRIGMPRI parameter of the queue arrives on the queue.
EVERY : Every time a message arrives on the queue with priority equal to or greater than the priority specified by the TRIGMPRI parameter of the queue.
DEPTH : When the number of messages with priority equal to or greater than the priority specified by TRIGMPRI is equal to the number indicated by the TRIGDPTH parameter.
NONE : No trigger messages are written.

INITQ(string)
The local name of the initiation queue on this queue manager, to which trigger messages relating to this queue are written.

PROCESS(string)
The local name of the WebSphere MQ process.
This parameter is the name of a process instance that identifies the application started by the queue manager when a trigger event occurs.
The process definition is not checked when the local queue is defined, but it must be available for a trigger event to occur.

+ end excerpt

++ Optional if using multi-version) Setup the runtime environment for MQ

The host where the test queue manager used for this techdoc has multiple versions of MQ (7.0, 7.1 and 7.5) and there is no primary installation.

Therefore, it is necessary to use “setmqenv” to specify the desired version of MQ to use.

In this case, because MQ 7.5 is the desired version, the following command needs to be executed:

```
  . /opt/mqm75/bin/setmqenv -n Installation2
```

Where the MQ 7.5 code is installed in /opt/mqm75 and it is labeled as Installation2

++ Use the runmqsc utility to administer the queue manager

```
runmqsc QM_VER75
```

- Define the process

Notes:

- You need to specify the full path for the command to be executed!
- In Red Hat, the full path for the touch command is: /bin/touch

```
DEFINE PROCESS ('TOUCH.PROCESS') APPLTYPE(UNIX) +
APPLICID('/usr/bin/touch  /tmp/file.txt')
```

- Define a local queue with triggering attributes

```
DEFINE QLOCAL(TRIGGER.Q) TRIGGER TRIGDPTH(1) TRIGTYPE(EVERY) +
  INITQ(SYSTEM.DEFAULT.INITIATION.QUEUE) PROCESS(TOUCH.PROCESS)

end
```

++
+++ Chapter 2: Testing triggering scenario, using monitor in foreground
++

This is an “educational” chapter which allows us to study the trigger monitor in more detail, but it is not really practical to use the trigger monitor in the foreground.

See Chapter 3 for a more suitable solution for automating the handling the trigger monitor

For a basic test of the triggering configuration for the queue and process that were defined in Chapter 1, you can do the following scenario.

++ Open 2 windows for UNIX command prompts and login as an MQ administrator

Note: The MQ administrator is only needed for running the runmqsc command, and it is not really needed to run the other commands, but for simplicity, in this scenario only one user id is used.

1) Window 1: For running the trigger monitor in the foreground.

Note: the trigger monitor NEEDS to be running in order for triggering messages to be generated!

2) Window 2: For issuing amqsput to put a message, for looking at /tmp and for run-mqsc

++ (Optional if using multi-version) Setup the environment for MQ 7.5

Within each window:

```
 . /opt/mqm75/bin/setmqenv -n Installation2
```

++ Window 1: Start the trigger monitor in the foreground

Note: the trigger monitor NEEDS to be running in order for triggering messages to be generated!

Issue the following command to launch in the foreground the trigger monitor, which will handle messages that are generated by the triggering code and these messages are placed in the initiation queue:

```
runmqtrm -m QM_VER75 -q  SYSTEM.DEFAULT.INITIATION.QUEUE
```

Notice that the command will run in the foreground and will NOT terminate while the queue manager is active.

```
5724-H72 (C) Copyright IBM Corp. 1994, 2011.  ALL RIGHTS RESERVED.
03/19/2014  11:56:59 AM : WebSphere MQ trigger monitor started.
__________________________________________________
03/19/2014  11:56:59 AM : Waiting for a trigger message
```

Note: If the queue manager ends while the trigger monitor is running, then you will see the following 2 lines:

```
03/19/2014  11:57:55 AM : WebSphere MQ trigger monitor connection broken.
03/19/2014  11:57:55 AM : WebSphere MQ trigger monitor ended.
```

++ Window 2: Verifying that the trigger monitor is running

Let's verify that the trigger monitor is running:

```
ps -ef | grep runmqtr
rivera   10140  8954  0 11:37 pts/2    00:00:00 runmqtrm -m QM_VER75 -q SYSTEM.DE-
FAULT.INITIATION.QUEUE
```

++ Window 2: Verifying that there are no files created by the "touch" process

The process that is launched by the triggering code will issue the following command:

```
   /usr/bin/touch /tmp/file.txt
```

Before we start the test, let's ensure that there is no file in that location:

```
ls -lt /tmp/file*
ls: cannot access /tmp/file*: No such file or directory
```

++ Window 2: Verifying that there are no messages in the local queue

Notice that CURDPETH is 0, which means that there are no messages in the queue.

```
echo "DISPLAY QL(TRIGGER.Q) CURDEPTH" | runmqsc QM_VER75
5724-H72 (C) Copyright IBM Corp. 1994, 2011.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM_VER75.
     1 : DISPLAY QL(TRIGGER.Q) CURDEPTH
AMQ8409: Display Queue details.
   QUEUE(TRIGGER.Q)                        TYPE(QLOCAL)
   CURDEPTH(0)
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

NOTE:
It is tempting to look at the initiation queue to see if there are messages that will instruct the process to be launched.
However, it is going to be very difficult for you to see a CURDEPTH other than 0, because one of the triggering rules is that a trigger message is only generated when the initiation queue is being opened for input by the trigger monitor, and this monitor consumes the trigger messages very fast!

```
echo "DISPLAY QL(SYSTEM.DEFAULT.INITIATION.QUEUE) CURDEPTH" | runmqsc
QM_VER75
5724-H72 (C) Copyright IBM Corp. 1994, 2011.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM_VER75.
     1 : DISPLAY QL(SYSTEM.DEFAULT.INITIATION.QUEUE) CURDEPTH
AMQ8409: Display Queue details.
   QUEUE(SYSTEM.DEFAULT.INITIATION.QUEUE)
   TYPE(QLOCAL)                        CURDEPTH(0)
One MQSC command read.
```

++ Window 2: Put a message into the queue TRIGGER.Q

```
amqsput TRIGGER.Q QM_VER75
Sample AMQSPUT0 start
target queue is TRIGGER.Q
TESTING TRIGGERING
Sample AMQSPUT0 end
```

++ Window 2: Explore the contents of the queue TRIGGER.Q

Let's use the sample browse command which does a non-destructive get.
Notice that there is 1 message, and it is the one done by amqsput in the previous step.
To keep the output short, non relevant lines are deleted.

amqsbcg TRIGGER.Q QM_VER75

```
+ begin excerpt of output
AMQSBCG0 - starts here
**********************
 MQOPEN - 'TRIGGER.Q'
 MQGET of message number 1, CompCode:0 Reason:0
****Message descriptor****
  StrucId  : 'MD  '  Version : 2
  Format : 'MQSTR   '
  Priority : 0  Persistence : 0
  MsgId : X'414D5120514D5F5645523735202020FFBE295302220020'
  CorrelId : X'000000000000000000000000000000000000000000000000'
  ** Origin Context
  PutApplType    : '6'
  PutApplName    : 'amqsput                     '
  PutDate  : '20140319'    PutTime  : '16124998'
  ApplOriginData : '    '
****   Message      ****
 length - 18 of 18 bytes
00000000:  5445 5354 494E 4720 5452 4947 4745 5249          'TESTING TRIGGERI'
00000010:  4E47                                            'NG              '
 No more messages

+ end excerpt
```

++ Window 2: Explore the contents of the initiation queue

One of the conditions for a trigger message to be generated is that the initiation queue must be used by a running trigger monitor and this monitor consumes the messages very fast.
Thus, it is not practical to easily see the contents of the triggering messages.

An MQ guru was consulted for this techdoc to get an example of a real triggering message from the initiation queue before it is consumed by the trigger monitor.

Notice the following 4 items:

In the Message Descriptor:
   Format = MQTRIG

In the payload section (contents):
   Eyecatcher (first 2 bytes in the payload): TM
   Name of the queue that caused the triggering: TEST.LQ
   Name of the process to be invoked: TEST.PR

+ begin excerpt (non-relevant lines were removed, to keep the example short)

```
MQGET of message number 1, CompCode:0 Reason:0
****Message descriptor****

  StrucId  : 'MD  '  Version : 2
  Report   : 0  MsgType : 8
  Expiry   : -1  Feedback : 0
  Encoding : 273  CodedCharSetId : 819
  Format : 'MQTRIG  '
  Priority : 0  Persistence : 0
  MsgId : X'414D5120545249472E514D202020202020532AFDA220002303'
  CorrelId : X'000000000000000000000000000000000000000000000000'
...

****   Message      ****
 length - 684 of 684 bytes
00000000:  544D 2020 0000 0001 5445 5354 2E4C 5120 'TM  ....TEST.LQ '
00000010:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000020:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000030:  2020 2020 2020 2020 5445 5354 2E50 5220 '        TEST.PR '
00000040:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000050:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000060:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000070:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000080:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000090:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
000000A0:  2020 2020 2020 2020 0000 0006 2020 2020 '        ....    '
```

++ end excerpt

++ Window 1: Notice that the trigger monitor handled the trigger message

When a message was put into TRIGGER.Q, due to the triggering attributes in that queue, a trigger message was generated and placed in the initiation queue (SYSTEM.DEFAULT.INITIATION.QUEUE).
Then the running trigger monitor (runmqtrm) read that trigger message and processed it and the following lines were displayed in the console:

++ begin excerpt

```
/usr/bin/touch /tmp/file.txt 'TMC    2TRIGGER.Q
TOUCH.PROCESS                                                                              /
usr/bin/touch /tmp/file.txt
QM_VER75                                 '
/usr/bin/touch: cannot touch `TMC    2TRIGGER.Q
TOUCH.PROCESS                                                                              /
usr/bin/touch /tmp/file.txt
QM_VER75                                 ': No such file or directory
03/20/2014  11:38:07 AM : End of application trigger.

__________________________________________________
03/20/2014  11:38:07 AM : Waiting for a trigger message
```

++ end excerpt

NOTE:
It is ok to ignore the warning that appears in the console: No such file or directory

++ Window 2: Verifying that there is a new/updated file by the “touch” process

Notice that there is a new file in /tmp

```
 ls -lt /tmp/file*
-rw-r--r-- 1 rivera mqm 0 2014-03-20 11:41 /tmp/file.txt
```

++ Window 2: Put another message and verify that the file in /tmp is updated

Let's use again amqsput to put another message:

```
amqsput TRIGGER.Q QM_VER75
Sample AMQSPUT0 start
target queue is TRIGGER.Q
```

TEST 2
Sample AMQSPUT0 end
Then let's check the time stamp for the file in /tmp:

ls -lt /tmp/file*
```
-rw-r--r-- 1 rivera mqm 0 2014-03-20 11:47 /tmp/file.txt
```

Notice that in our previous “ls” command the time stamp for the file was:
   11:41
Now the time stamp is:
   11:47

The above is a visual indication that the triggered process was actually invoked and performed a task.

++ Window 1: Terminate the triggering process in the foreground

Use Ctrl-C to terminate the running triggering process

+ begin excerpt
```
03/20/2014  11:47:16 AM : Waiting for a trigger message

^C
```
+ end excerpt

++
+++ Chapter 3: Testing triggering scenario, using monitor defined as Service
++

The scenario if Chapter 2 is not really very practical because it required a manual start and a manual stop of the trigger monitor. However, it allowed us to see some details that otherwise will not be easily seen when exploring the triggering function.

A practical approach is to define the trigger monitor as a Service, under the control of the queue manager. Notice the attribute in the DEFINE SERVICE statement:
  CONTROL(QMGR)
This means that when the queue manager starts, then the Service (the trigger monitor) will start, and when the queue manager stops, then the Service will stop.

++ Window 1: Use runmqsc to define a Service for the trigger monitor, to start it and to display the status

```
runmqsc QM_VER75

DEFINE SERVICE(TRIGMON) CONTROL(QMGR) SERVTYPE(SERVER)  +
STARTCMD('+MQ_INSTALL_PATH+bin/runmqtrm')  +
STARTARG('-m +QMNAME+ -q SYSTEM.DEFAULT.INITIATION.QUEUE') +
STOPCMD('+MQ_INSTALL_PATH+bin/amqsstop')  +
STOPARG('-m +QMNAME+ -p +MQ_SERVER_PID+')  STDOUT('/tmp/TrigMon.stdout') +
STDERR('/tmp/TrigMon.stderr')

START  SERVICE(TRIGMON)
   AMQ8733: Request to start Service accepted.

DISPLAY SVSTATUS(TRIGMON)
  AMQ8632: Display service status details.
   SERVICE(TRIGMON)                    STATUS(RUNNING)
   PID(665)                            SERVTYPE(SERVER)
   STARTDA(2014-03-20)                 STARTTI(13.10.38)
   CONTROL(QMGR)                       STARTCMD(/opt/mqm75/bin/runmqtrm)
   STARTARG(-m QM_VER75 -q SYSTEM.DEFAULT.INITIATION.QUEUE)
   STOPCMD(/opt/mqm75/bin/amqsstop)    STOPARG(-m QM_VER75 -p 665)
   DESCR( )                            STDOUT(/tmp/TrigMon.stdout)
   STDERR(/tmp/TrigMon.stderr)

end
```

Notice that the DEFINE SERVICE in the above runmqsc command has interesting tokens:

+QMNAME+ is a token representing the name of the queue manager.

+MQ_INSTALL_PATH+ is a token representing the location of the MQ code, such as /opt/mqm/ (notice that it includes the slash at the end!)

+MQ_SERVER_PID+ represents the process id (pid) of the MQ queue manager

For a full list of the representative tokens see:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.adm.doc/q021010_.htm
WebSphere MQ > Administering > Administering local WebSphere MQ objects > Working with services
IBM WebSphere MQ information center, Version 7.5 Operating Systems: UNIX, Linux, Windows
Replaceable inserts on service definitions
+ begin excerpt
In the definition of a service object, it is possible to substitute tokens...
The following are common tokens
MQ_INSTALL_PATH The location where WebSphere® MQ is installed.
MQ_DATA_PATH The location of the WebSphere MQ data directory:
On UNIX and Linux systems, the WebSphere MQ data directory location is /var/mqm/
On Windows systems, the location of the WebSphere MQ data directory is the data directory selected during the installation of WebSphere MQ
QMNAME The current queue manager name.
MQ_SERVICE_NAME The name of the service.
MQ_SERVER_PID This token can only be used by the STOPARG and STOPCMD arguments.
MQ_Q_MGR_DATA_PATH The location of the queue manager data directory.
MQ_Q_MGR_DATA_NAME The transformed name of the queue manager

+ end excerpt

For those who are curious, notice that the trigger monitor is running

```
ps -ef | grep runmqtrm
rivera     665 12150  0 13:10 ?        00:00:00 /opt/mqm75/bin/runmqtrm -m QM_VER75
-q SYSTEM.DEFAULT.INITIATION.QUEUE
```

+ Window 2:

Notice that the output files defined for the trigger monitor are created:

```
 STDOUT('/tmp/TrigMon.stdout')
 STDERR('/tmp/TrigMon.stderr')
```

ls -lt /tmp/Trig*

```
-rw-r--r-- 1 mqm mqm 239 2014-03-20 13:18 /tmp/TrigMon.stdout
-rw-r--r-- 1 mqm mqm   0 2014-03-20 13:18 /tmp/TrigMon.stderr
```

At this point, notice that the trigger monitor has not handled any messages yet:

cat /tmp/TrigMon.stdout

```
5724-H72 (C) Copyright IBM Corp. 1994, 2011.  ALL RIGHTS RESERVED.
03/20/2014  01:18:24 PM : WebSphere MQ trigger monitor started.
__________________________________________________
03/20/2014  01:18:25 PM : Waiting for a trigger message
```

Let's put a message:

amqsput TRIGGER.Q QM_VER75

```
Sample AMQSPUT0 start
target queue is TRIGGER.Q
TEST 3
Sample AMQSPUT0 end
```

Notice that the file sizes have increased for the output files:

 ls -lt /tmp/Trig*

```
-rw-r--r-- 1 mqm mqm  791 2014-03-20 13:21 /tmp/TrigMon.stderr
-rw-r--r-- 1 mqm mqm 1167 2014-03-20 13:21 /tmp/TrigMon.stdout
```

Let's take a look at each one of the output files:

cat /tmp/TrigMon.stderr

```
/usr/bin/touch: cannot touch `TMC    2TRIGGER.Q
TOUCH.PROCESS                                                                      /
usr/bin/touch /tmp/file.txt
QM_VER75                                   ': No such file or directory
```

cat /tmp/TrigMon.stdout

```
5724-H72 (C) Copyright IBM Corp. 1994, 2011.  ALL RIGHTS RESERVED.
03/20/2014  01:18:24 PM : WebSphere MQ trigger monitor started.
__________________________________________________
03/20/2014  01:18:25 PM : Waiting for a trigger message
/usr/bin/touch /tmp/file.txt 'TMC    2TRIGGER.Q
TOUCH.PROCESS                                                          /
usr/bin/touch /tmp/file.txt
QM_VER75                                  '
03/20/2014  01:21:10 PM : End of application trigger.
__________________________________________________
03/20/2014  01:21:10 PM : Waiting for a trigger message
```

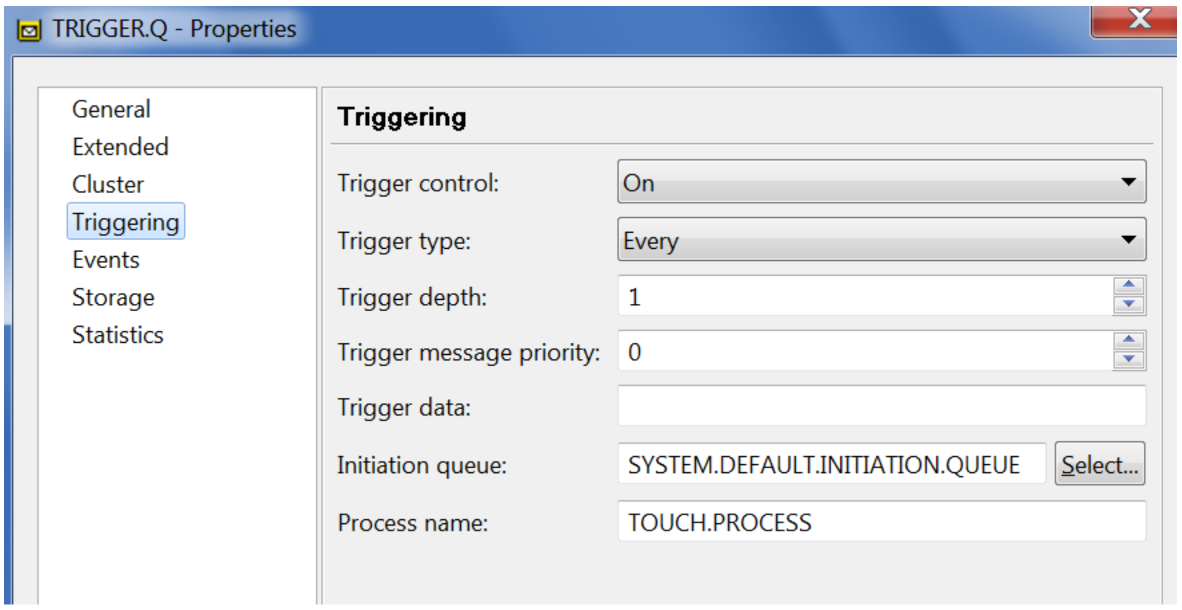
```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++ Chapter 4: Snapshots of triggering objects from MQ Explorer
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

The MQ Explorer allows an easier configuration and display of the triggering objects.

The following snapshot was obtained as follows:
 From the left navigation panel, expand the desired queue manager to show the folders.
 Select the folder: Queues
 Select the desired queue that will receive the normal messages.
 Right click and select Properties.
 Select the tab: Triggering

You will see the following fields:

```
  Trigger control:     On
   Trigger type:       Every
   Trigger depth:      1
   Trigger message priority: 0
   Trigger data:   <blank>
   Initiation queue: SYSTEM.DEFAULT.INITIATION.QUEUE
   Process name:     TOUCH.PROCESS
```



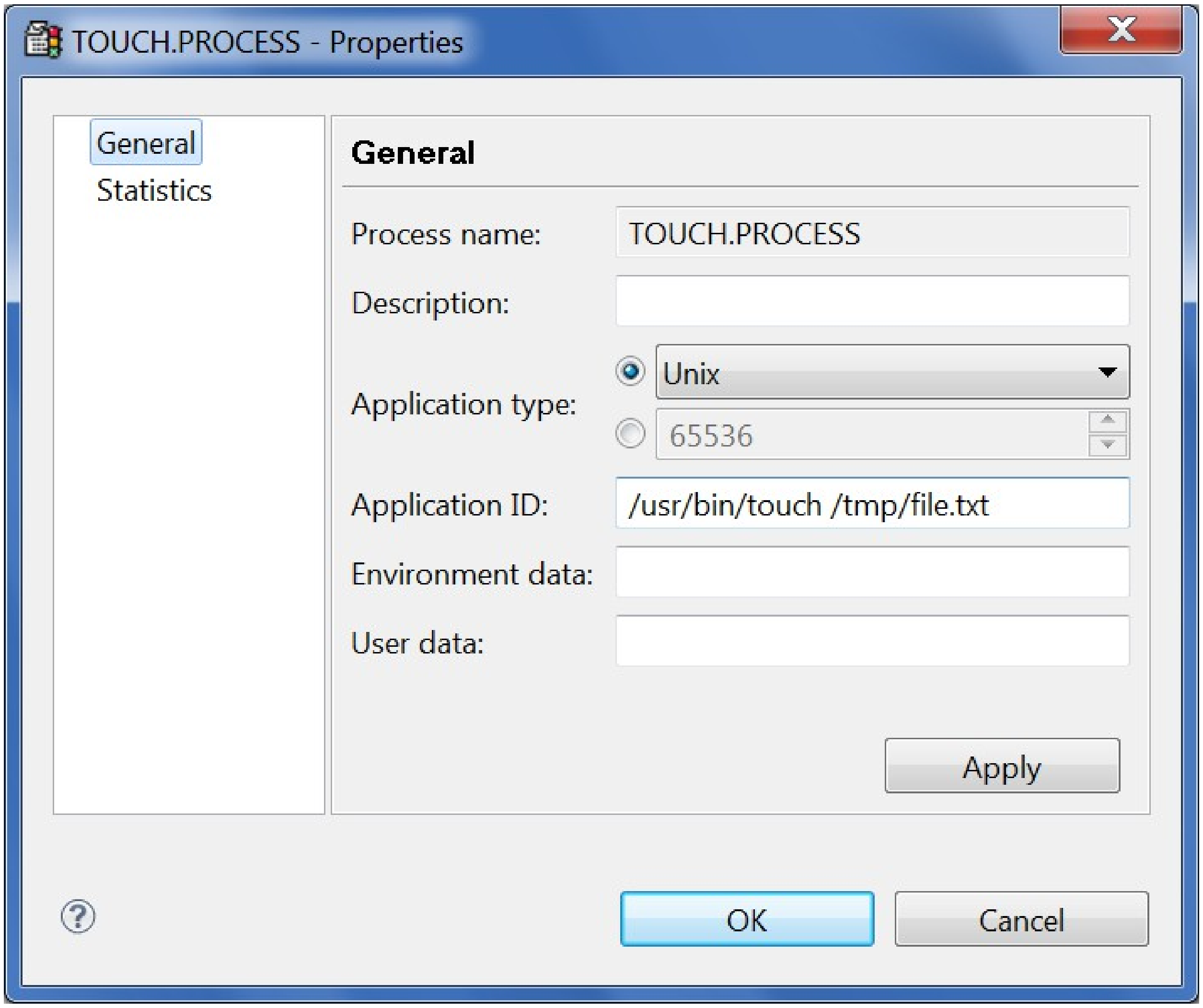
Triggering

Trigger control:	On
Trigger type:	Every
Trigger depth:	1
Trigger message priority:	0
Trigger data:	
Initiation queue:	SYSTEM.DEFAULT.INITIATION.QUEUE
Process name:	TOUCH.PROCESS

For the Process:
  Select and expand the folder: Process Definitions
  Select the desired process, and view the Properties

You will see the following fields:

Process name:	TOUCH.PROCESS
Description:	<blank>
Application type:	Unix
Application ID:	/usr/bin/touch /tmp/file.txt
Environment data:	<blank>
User data:	<blank>



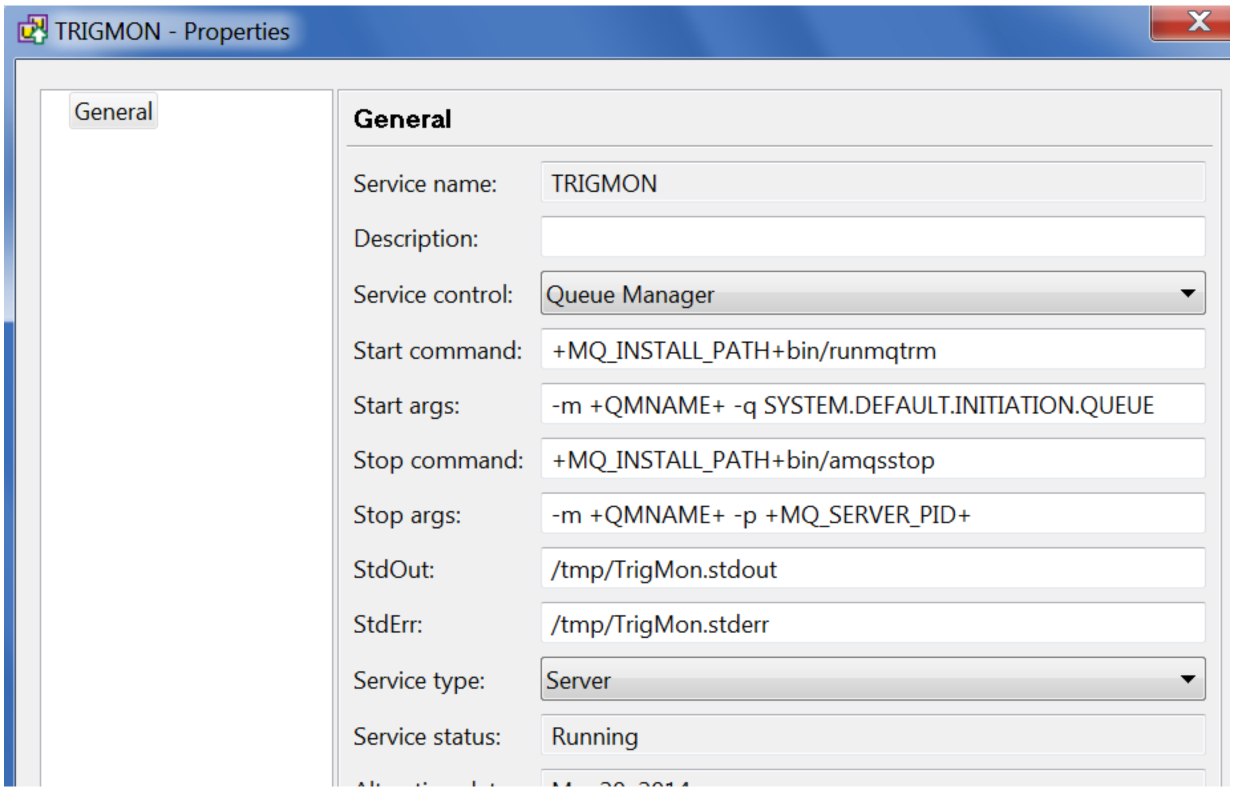
General

Process name: TOUCH.PROCESS
Description:
Application type: (● Unix) ○ 65536
Application ID: /usr/bin/touch /tmp/file.txt
Environment data:
User data:

For the Service:
  Select and expand the folder: Services
  Select the desired service, and view the Properties

You will see the following fields:

Service name:	TRIGMON
Description:	<blank>
Service control:	Queue Manager
Start command:	+MQ_INSTALL_PATH+bin/runmqtrm
Start args:	-m +QMNAME+ -q SYSTEM.DEFAULT.INITIATION.QUEUE
Stop command:	+MQ_INSTALL_PATH+bin/amqsstop
Stop args:	-m +QMNAME+ -p +MQ_SERVER_PID+
StdOut:	/tmp/TrigMon.stdout
StdErr:	/tmp/TrigMon.stderr
Service type:	Server
Service status:	Running



General

Service name:	TRIGMON
Description:	
Service control:	Queue Manager
Start command:	+MQ_INSTALL_PATH+bin/runmqtrm
Start args:	-m +QMNAME+ -q SYSTEM.DEFAULT.INITIATION.QUEUE
Stop command:	+MQ_INSTALL_PATH+bin/amqsstop
Stop args:	-m +QMNAME+ -p +MQ_SERVER_PID+
StdOut:	/tmp/TrigMon.stdout
StdErr:	/tmp/TrigMon.stderr
Service type:	Server
Service status:	Running

+++ end +++