Compiling IBM MQ sample programs on AIX using XL C/C++

[Prema Laxmanachar](javascript:;) |Oct 20 2016 Updated

IBM recently announced support for XL C/C++ 13.1 on AIX. XL compiler is used to compile the IBM MQ samples like amqsput, amqsget and amqsbcg to put , get , browse samples using both client and server libraries. This document is helps for the user who wants to build their own 32 bit or 64bit applications using IBM MQ to put and get messages.

This document includes the information on XLC compiler, support AIX OS, how to find out the PATH and version of XL compiler installed and setting up the PATH of XL, installation of MQ, PATH of the C/CPP MQ samples, modes of compilation and commands to compile the samples using threaded and unthreaded  modes.

**Identify the XL compiler installed on AIX machine. The following are the XL compiler versions supported by AIX**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Compiler** | **AIX 5.1** | **AIX 5.2** | **AIX 5.3** | **AIX 6.1** | | **AIX 7.1** |
| **XL C/C++ Compilers** | | | | |  | |
| **XL C/C++ for AIX, V13.1** | X | X | X | GA | GA | |
| **XL C/C++ for AIX, V12.1** | X | X | GA | GA | GA | |

You can find the more details [here](http://www-01.ibm.com/support/docview.wss?rs=43&uid=swg21326972).

**Check the path and version of the XL compiler installed on the AIX. Run the below command to know the XL version and Path where it is installed:**

bash-4.2$ which xlC

/opt/IBM/xlC/13.1.0/bin/xlC

Set PATH of the XL compiler as below if the PATH is not set. User might get warning like “xlc” not found when they issue xlc command.

# export PATH=$PATH:/opt/IBM/xlc/13.1.0/bin

Check the xlc version by passing the below commands

bash-4.2$ xlC -qversion=verbose

IBM XL C/C++ for AIX, V13.1 (5725-C72, 5765-J07)

Version: 13.01.0000.0000

Driver Version: 13.01(C/C++) Level: 140415 ID: \_9iE48sA8EeOFQpd5s7RXNw

C Front End Version: 13.01(C/C++) Level: 140416 ID: \_8DFr4sTxEeOClv7uAVQYOQ

C++ Front End Version: 13.01(C/C++) Level: 140416 ID: \_GH60gsW7EeOCnf7uAVQYOQ

High-Level Optimizer Version: 13.01(C/C++) and 15.01(Fortran) Level: 140424 ID: \_l\_3gYswEEeOC1f7uAVQYOQ

Low-Level Optimizer Version: 13.01(C/C++) and 15.01(Fortran) Level: 140424 ID: \_mH4z8swEEeOC1f7uAVQYOQ

**Install IBM MQ (v7.1 onwards) if it is not installed. If it is already installed check the version of the MQ installed by issuing the** dspmqver **command.**

IBM MQ will be installed by default on /usr/mqm directory.

Issue dspmqver to get the version of the MQ

bash-4.2$dspmqver

Name:        WebSphere MQ

Version:     7.5.0.6

Level:       p750-006-160212

BuildType:   IKAP - (Production)

Platform:    WebSphere MQ for AIX

Mode:        64-bit

O/S:         AIX 6.1

InstName:    Installation4

InstDesc:

Primary:     Yes

InstPath:    /usr/mqm

DataPath:    /var/mqm

MaxCmdLevel: 750

LicenseType: Production

If the you are new to IBM MQ, follow [this link](http://www.ibm.com/support/knowledgecenter/SSFKSJ_8.0.0/com.ibm.mq.ins.doc/q008600_.htm) to install MQ on AIX.

**Modes of Compiling MQ Samples**

There are Threaded and Unthreaded mode compiling the MQ Samples.

**Threaded Mode:**A thread is considered to be connected to IBM® MQ from MQCONN (or MQCONNX) until MQDISC. UNIX and Linux systems safely allow the setting up of a signal handler for such signals for the whole process. However, IBM MQ sets up its own handler for the following signals, in the application process, while any thread is connected to IBM MQ:

* SIGBUS
* SIGFPE
* SIGSEGV
* SIGILL

For more information on Threaded mode, read [here](http://www.ibm.com/support/knowledgecenter/en/SSFKSJ_8.0.0/com.ibm.mq.dev.doc/q025880_.htm).

**Unthreaded Mode:**All applications are considered threaded even if they use only a single thread.

Each MQI function sets up its own signal handler for the signals:

* SIGALRM
* SIGBUS
* SIGFPE
* SIGSEGV
* SIGILL

For more information on Unthreaded mode, read [here](http://www.ibm.com/support/knowledgecenter/en/SSFKSJ_8.0.0/com.ibm.mq.dev.doc/q025870_.htm).

**Threaded and Unthreaded modes have 32 bit mode and 64 bit mode of compilation.**

*If you are compiling for 32 bit applications use 32 bit libraries, and for 64 bit applications use 64 bit libraries to compile the samples. Below mentioned are the modes used to compile the samples on server and client.*

One more important thing while compiling is to not use threaded and non-threaded libraries at the same time. If used, it might cause compilation failure.

Server Unthreaded 32 bi

Server Unthreaded 64 bit

Server Threaded 32 bit

Server Threaded 64 bit

Client Unthreaded 32 bit

Client Unthreaded 64 bit

Client Threaded 32 bit

Client Threaded 64 bit

For more information on compilation modes on AIX, read [this](http://www.ibm.com/support/knowledgecenter/en/SSFKSJ_8.0.0/com.ibm.mq.dev.doc/q028350_.htm).

**Below are the IBM MQ samples which needs to be compiled using above mentioned modes and what does that samples do:**

Precompiled C programs are supplied in the MQINST-dir/samp/bin directory.  Below mentioned C/CPP samples present under MQINST-dir /samp directory.

**C:**

amqsput0.c :  amqsput sample program to put a message on the queue

amqsget0.c : amqsget sample program to get the message back from the queue

amqsbcg0.c : The Browser sample program reads and writes both the message descriptor and the message content fields of all the messages on a queue

**CPP/C++**

imqsput.cpp : imqsput sample program to put a message on the queue

imqsget.cpp : imqsget sample program to get the message back from the queue

imqdput.cpp : imqdput sample program puts messages to a distribution list consisting of two queues.

imqwrld.cpp :  imqwrld sample program shows how to put and get a regular datagram (C structure) using the ImqMessage class

**Follow the below steps to compile above mentioned C/CPP samples:**

**Example that explains the command used for compiling 32 bit and 64 bit amqsput.c sample using threaded mode:**

**For 32–bit applications:**

**$ xlc\_r -o amqsput\_32\_r amqsput0.c -I/**MQINST-dir**/inc -L/**MQINST-dir**/lib -lmqm\_r**

**Xlc\_r /**xlC – Compiler command

-o amqsput\_32\_r : It is the output file of the compiled sample

amqsput0.c :  amqsput sample C program

-I **/**MQINST-dir/inc – This refers to the C/CPP include files

-L **/**MQINST-dir/lib : This referes to the 32 bit library Path to link the 32 bit library file.

**For 64–bit applications:**

**$ xlc\_r -q64 -o amqsput\_64\_r amqsput0.c -I/**MQINST-dir**/inc -L/**MQINST-dir**/lib64 -lmqm\_r**

**-o amqsput\_64\_r : output name is saved as 64 bit (you can choose your own output name of your choice)**

**-L/**MQINST-dir**/lib64 :**This referes to the 64 bit library Path to link the 64 bit library file.

**Note:** If you compile for unthreaded applications, use **“lmqm”** library, whereas for threaded applications use “**lmqm\_r”** library to link.

If you want to use the programs on a machine that has only the WebSphere MQ client for AIX® installed, recompile the programs to link them with the client library (-lmqic)

Please read [here](http://www.ibm.com/support/knowledgecenter/SSFKSJ_7.0.1/com.ibm.mq.csqzal.doc/fg16160_.htm) for a detailed explanation:

**Below are the commands used for compiling C/CPP samples in Threaded and Unthreaded mode on server and client.**

**Server 32 bit : - Unthreaded:**

**Compiling cpp samples:**

xlC -o imqsput\_32 imqsput.cpp -qchars=signed -I /MQinst-dir/inc -L /MQinst-dir/lib -limqs23ia -limqb23ia -lmqm

xlC -o imqsget\_32 imqsget.cpp -qchars=signed -I /MQinst-dir qm/inc -L /MQinst-dir /lib -limqs23ia -limqb23ia -lmqm

**Compiling "c" samples:**

xlC -o amqsput\_32 amqsput0.c -qchars=signed -I /MQinst-dir/inc -L /MQinst-dir/lib -limqs23ia -limqb23ia -lmqm

**Server 32 bit  - Threaded :**

**Compiling cpp samples:**

xlC\_r -o imqsput\_32\_r imqsput.cpp -qchars=signed -I /MQinst-dir/inc -L /MQinst-dir/lib -limqs23ia\_r -limqb23ia\_r -lmqm\_r

**Compiling "c" samples:**

xlC\_r -o amqsput\_32\_r amqsput0.c -qchars=signed -I /MQinst-dir/inc -L /MQinst-dir/lib -limqs23ia\_r -limqb23ia\_r -lmqm\_r

**Server 64 bit : - Unthreaded:**

xlC -q64 -o imqsput\_64 imqsput.cpp -qchars=signed -I /MQinst-dir/inc -L /MQinst-dir/lib64 -limqs23ia -limqb23ia -lmqm

xlC -q64 -o imqsget\_64 imqsget.cpp -qchars=signed -I /MQinst-dir/inc -L /MQinst-dir/lib64 -limqs23ia -limqb23ia -lmqm

**Server 64 bit Threaded :**

xlC\_r -q64 -o imqsput\_64\_r imqsput.cpp -qchars=signed -I /MQinst-dir/inc -L /MQinst-dir/lib64 -limqs23ia\_r -limqb23ia\_r -lmqm\_r

xlC\_r -q64 -o imqsget\_64\_r imqsget.cpp -qchars=signed -I /MQinst-dir/inc -L /MQinst-dir/lib64 -limqs23ia\_r -limqb23ia\_r -lmqm\_r

**Client mode:-**

**Client 32 bit Unthreaded:**

xlC -o imqsputc\_32 imqsput.cpp -qchars=signed -I /MQinst-dir/inc -L /MQinst-dir/lib -limqc23ia -limqb23ia -lmqic

xlC -o imqsgetc\_32 imqsget.cpp -qchars=signed -I /MQinst-dir/inc -L /MQinst-dir/lib -limqc23ia -limqb23ia -lmqic

**Client 32 bit threaded:**

xlC\_r -o imqsputc\_32\_r imqsput.cpp -qchars=signed -I /MQinst-dir/inc -L /MQinst-dir/lib -limqc23ia\_r -limqb23ia\_r -lmqic\_r

xlC\_r -o imqsgetc\_32\_r imqsget.cpp -qchars=signed -I /MQinst-dir/inc -L /MQinst-dir/lib -limqc23ia\_r -limqb23ia\_r -lmqic\_r

**Client 64 bit unthreaded:**

xlC -q64 -o imqsputc\_64 imqsput.cpp -qchars=signed -I /MQinst-dir/inc -L /MQinst-dir/lib64 -limqc23ia -limqb23ia -lmqic

xlC -q64 -o imqsgetc\_64 imqsget.cpp -qchars=signed -I /MQinst-dir/inc -L /MQinst-dir/lib64 -limqc23ia -limqb23ia -lmqic

**Client 64 bit Threaded:**

xlC\_r -q64 -o imqsputc\_64\_r imqsput.cpp -qchars=signed -I /MQinst-dir/inc -L /MQinst-dir/lib64 -limqc23ia\_r -limqb23ia\_r -lmqic\_r

xlC\_r -q64 -o imqsgetc\_64\_r imqsget.cpp -qchars=signed -I /MQinst-dir/inc -L /MQinst-dir/lib64 -limqc23ia\_r -limqb23ia\_r -lmqic\_r

Note: -qchars=signed this parameter is applicable only for xl compiler 12 , 13 and above.

Make sure that while compiling or after compilation it should not throw any error message or warnings.

The error or warnings that you might notice during compilation are:

|  |  |
| --- | --- |
| Return Code | Error type |
| 1 | Any error with a severity level higher than the setting of the -qhalt compiler option has been detected. |
| 40 | An option error or an unrecoverable error has been detected. |
| 41 | A configuration file error has been detected. |
| 249 | A no-files-specified error has been detected. |
| 250 | An out-of-memory error has been detected. The compiler cannot allocate any more memory for its use. |
| 251 | A signal-received error has been detected. That is, an unrecoverable error or interrupt signal has occurred. |
| 252 | A file-not-found error has been detected. |
| 253 | An input/output error has been detected: files cannot be read or written to. |

**Create the Queue manager and its objects**

crtmqm QM

strmqm QM

runmqsc QM

define ql(Q) defpsist(yes)

define chlannel(Channel name) chltype(svrconn) trptype(tcp) sslcauth(optional) mcauser('username ')

end

When trying to connect to the queue manager in the client mode using the compiled c/cpp samples, you need to modify some security changes in Queue manager objects.

Start the Listener : runmqlsr -t tcp -m QM -p 5656 &

After changing the properties, you can do the put and get using the compiled c/cpp samples. Go to the directory where the output of the compiled c/cpp samples are present. Using those samples you can easily put and get the messages.

#./amqsput\_32 Q QM

Sample AMQSPUT0 Start

target queue is Q

Test message

#./amqsget\_32 Q QM

Sample AMQSGET0 Start

message <Test message>

#Likewise you can put and get messages using other compiled samples.

Below is the attached snapshot for the put and get messages using compiled samples:

Please check for System-generated errors /var/mqm/errors or in Queue manager error logs for any errors or FDC's.

Generally errors like mqrc 2035 error (This error will generally appear when client tries to connect to the queue manager running on the machine) when user tries to put or get the messages to the queue. To overcome this problem, need to modify the Queue manager objects as mentioned above in step 7.

In system / queue manager error logs usually gets the error something like:

----- amqccita.c : 4116 -------------------------------------------------------  
23/04/15 12:50:40 - Process(28901550.134) User(ats) Program(amqrmppa)  
                    Host(aorta.v6.hursley.ibm.com) Installation(Installation3)  
                    VRMF(8.0.0.2) QMgr(QM)  
  
AMQ9999: Channel 'C' to host '9.20.29.82' ended abnormally.  
  
EXPLANATION:  
The channel program running under process ID 28901550 for channel 'C' ended  
abnormally. The host name is '9.20.29.82'; in some cases the host name cannot  
be determined and so is shown as '????'.  
ACTION:  
Look at previous error messages for the channel program in the error logs to  
determine the cause of the failure. Note that this message can be excluded  
completely or suppressed by tuning the "ExcludeMessage" or "SuppressMessage"  
attributes under the "QMErrorLog" stanza in qm.ini. Further information can be  
found in the System Administration Guide.  
----- amqrmrsa.c : 930 --------------------------------------------------------

The AMQERR01.LOG shows that an application ended without a clean disconnect. This kind of error might occur when compilation of sample is in process and user terminated the samples before it had exited, which would be sufficient to cause this error in the log.

If we expect the application to exit cleanly, it should be re-run and given time to do so, in order to check that it does not hang indefinitely.

**Note:** While the compilation of samples is in process, don’t terminate until unless it exit after completing the compilation.