

# CICS TS MQ Interface for All

Thomas Dunlap Themis, Inc. [tomd@themisinc.com](mailto:tomd@themisinc.com)

Please use the 'Question Tab' for questions instead of the Chat window.

Link to get the PDF : [www.themisinc.com/webinars](http://www.themisinc.com/webinars)

# **A LOOK AT THE APPLICATION PROGRAM**

## MQ Usage in a COBOL Program

### WORKING-STORAGE SECTION:

```
01 MQ-STRUCTURES.  
   COPY CMQODV.  
   COPY CMQMDV.  
   COPY CMQGMV.  
   COPY CMQPMV.  
   COPY CMQV SUPPRESS.  
  
01 WS-VARIABLES.  
   05 MQ-HCONN          PIC S9(9) BINARY VALUE 0.  
   05 MQ-HOBJ-I        PIC S9(9) BINARY VALUE 0.  
   05 MQ-HOBJ-O        PIC S9(9) BINARY VALUE 0.  
   05 MQ-OPTIONS       PIC S9(9) BINARY.  
   05 MQ-MSGIN-LENGTH  PIC S9(9) BINARY VALUE 80.  
   05 MQ-MSGOUT-LENGTH PIC S9(9) BINARY VALUE 80.  
   05 MQ-DATA-LENGTH  PIC S9(9) BINARY.  
   05 MQ-COMPCODE      PIC S9(9) BINARY.  
   05 MQ-REASON        PIC S9(9) BINARY.
```

Five copy statements necessary for all MQ application programs.

The WS\_VARIABLES are those required at a minimum in support the application program.

Each application program which will execute using MQ call statements require a few basic inclusions of generated code. The COPY shown are the one required by all applications using MQ functions. We have suppressed the list of the CMQV structure since it is over 20 pages of COBOL variable constants.

The WS-VARIABLES are the minimum necessary to support bot getting and putting messages from and to queues. The MQ-CONN would normally be the handle acquired by an MQCONN function, but that call along with the MQDISC call are ignored by CICS / MQ interface. It is CICS itself that has performed the MQCONN to the queue manager.

The MQ-HOBJ-I and MQ-HOBJ-O represent the handles, acquired on MQOPEN for a queue, used to specify which queue the other calls relate to. The MQ-COMPCODE and MQ-REASON are the variables which the application must test after every MQ call to determine its success.

## MQ Usage in a COBOL Program

### WORKING-STORAGE SECTION (cont.):

```
01 MQ-MSGOUT.  
05 MQ-MSGOUT-ACTION      PIC X(1).  
05 MQ-MSGOUT-ITEMNO     PIC 9(6).  
05 MQ-MSGOUT-QTY        PIC 9(7).  
05                       PIC X(66) VALUE SPACES.  
  
01 MQ-MSGIN.  
05 MQ-MSGIN-ITEMNO     PIC X(6).  
05 MQ-MSGIN-DESC       PIC X(25).  
05 MQ-MSGIN-UNITP      PIC S9(3)V99.  
05 MQ-MSGIN-QTYOH      PIC S9(7).  
05 MQ-MSGIN-QTYSHIP    PIC S9(7).  
05 MQ-MSGIN-QTYBORD    PIC S9(7).  
05 MQ-MSGIN-ACTION     PIC X(7).  
05 MQ-MSGIN-ORDQTY     PIC S9(7).  
05                       PIC X(9).
```

Simple input and output message areas in WORKING STORAGE.

If the actual message areas exceed 200-300K, then they should probably be in the LINKAGE SECTION and their storage acquired by a CICS GETMAIN command.



This slide contains the message input and output areas used on the MQGET and MQPUT calls. Our messages are rather simple and limited in size. SO we have put the data areas in the WORKING-STORAGE SECTION. Many times you will have these as copy books rather than native COBOL variables.

Since the WORKING-STORAGE area is being copied for each task using the program, we recommend that larger messages areas be put into the LINKAGE SECTION. Here we use a size of 200-300 K as a suggestion. However, that size will vary depending upon your CICS environment. When the application is using message areas in the LINKAGE SECTION, it will be responsible to perform a CICS GETMAIN command to acquire the storage for the message area.

## MQ Usage in a COBOL Program

### PROCEDURE DIVISION:

```
MOVE CA-REQCODE TO MQ-MSGOUT-ACTION
MOVE CA-REQITEM TO MQ-MSGOUT-ITEMNO
MOVE CA-REQQTY TO MQ-MSGOUT-QTY

MOVE MQ-OBJECT-O TO MQOD-OBJECTNAME
MOVE SPACES TO MQOD-OBJECTQMGRNAME
MOVE MQMT-REQUEST TO MQMD-MSGTYPE
MOVE MQ-OBJECT-I TO MQMD-REPLYTOQ
MOVE SPACES TO MQMD-REPLYTOQMGR
MOVE MQRO-NONE TO MQMD-REPORT
MOVE MQFMT-STRING TO MQMD-FORMAT
MOVE MQMI-NONE TO MQMD-MSGID
MOVE MQCI-NONE TO MQMD-CORRELID
COMPUTE MQPMO-OPTIONS = MQPMO-NEW-MSG-ID + MQPMO-NO-SYNCPOINT
CALL 'MQPUT1' USING MQ-HCONN MQOD MQMD MQPMO
                    MQ-MSGOUT-LENGTH MQ-MSGOUT
                    MQ-COMPCODE MQ-REASON

EVALUATE MQ-COMPCODE
  WHEN MQCC-OK CONTINUE
  WHEN MQCC-WARNING PERFORM 900-WARNING THROUGH 900-EXIT
  WHEN OTHER PERFORM 910-ERROR THROUGH 910-EXIT
  GO TO 240-SEND-MAINMAP-ERROR
END-EVALUATE
```

The procedural code to complete an MQPUT1 call, combination of and MQOPEN, MQPUT, and MQCLOSE in a single call.



Now to the actual executable MQ call statements on this slide. Here we see the COBOL statements necessary to populate the variables for an MQPUT1 call. The MQPUT1 call is a combination of an MQOPEN, MQPUT, and MQCLOSE in a single call. Many times your CICS application will work with a single message within a task, so this call saves you a bit of code, plus interacting with the MQ interface two times. This does mean less overhead to the executing task. However, this savings disappears when you process more than one message with an MQPUT1 call.

Since we are putting a message to a queue, the first three COBOL MOVES populate the message area for our simple message. It is a “request message”, passed to another CICS task to process and return a result.

The MOVES to the MQOD- variables setup the queue name for the MQOPEN portion of the call.

The MOVES to the MQMD- variables setup MQ options which describe the message and functions desired. The MQMT-REQUEST identifies this as a “request message”, plus indicates the use of the MQ “ReplyToQ” function. The MOVES for MQMD-REPLYTOQ and MQMD-REPLYTOQMGR provide the values which will be used by the other MQ application to respond with the “reply message”. The MQFMT-STRING informs MQ that the data portion of the message is strictly character data and may participate in data conversion. The setting of MQMI-NONE is important to ensure that each message obtains a unique message identifier. The setting of MQCI-NONE means that the receiving application will not be retrieving specific messages.

The COMPUTE statement sets the MQPMO-NO-SYNCPOINT option, which means that the message is not put within Unit-of-Work (UOW) control. If the CICS task were to abend, the message would not be rolled back.

Next is the actual MQPUT1 call itself. It is a “static call” which means it will resolve to an actual module at Linkage Edit time. You cannot use the COBOL DYNAM option with MQ applications since the name on the call is not the actual module name. We will show how to code MQ dynamic calls at the end of this section. The EVALUATE statement show a simple example of testing how the call completed.

## MQ Usage in a COBOL Program

### PROCEDURE DIVISION (cont.):

```
      COMPUTE MQ-OPTIONS = MQOO-INPUT-AS-Q-DEF + MQOO-FAIL-IF-QUIESCING +
                          MQOO-BROWSE
      MOVE MQ-OBJECT-I TO MQOD-OBJECTNAME
      MOVE SPACES      TO MQOD-OBJECTMGRNAME
      CALL 'MQOPEN'    USING MQ-CONN MQOD MQ-OPTIONS MQ-HOBJ-I
                          MQ-COMPCODE MQ-REASON

      EVALUATE MQ-COMPCODE
        WHEN MQCC-OK CONTINUE
        WHEN MQCC-WARNING PERFORM 900-WARNING THROUGH 900-EXIT
        WHEN OTHER PERFORM 910-ERROR THROUGH 910-EXIT
        GO TO 240-SEND-MAINMAP-ERROR
      END-EVALUATE
```

The procedural code to complete an MQOPEN call.

On this slide we show an example of the MQOPEN call to access a queue. The COMPUTE statement sets options to specify the queue is input to the application, browsing of messages will be allowed, and the call is to fail if the queue manager is being shutdown.

The two MOVE statements provide the queue name and clear the queue manager name. This is followed by the actual MQOPEN call. The MQ-HOBJ-I is the returned handle which will be used on subsequent MQGET calls to specify which queue is being utilized.

The EVALUATE statement will check how the call completed.

## MQ Usage in a COBOL Program

### PROCEDURE DIVISION (cont.):

```
MOVE MQMI-NONE TO MQMD-MSGID
MOVE MQCI-NONE TO MQMD-CORRELID
MOVE 20000      TO MQGMO-WAITINTERVAL
COMPUTE MQGMO-OPTIONS = MQGMO-WAIT + MQGMO-NO-SYNCPPOINT
CALL 'MQGET'    USING MQ-CONN MQ-HOBJ-I MQMD MQGMO
                MQ-MSGIN-LENGTH MQ-MSGIN MQ-DATA-LENGTH
                MQ-COMPCODE MQ-REASON

EVALUATE MQ-COMPCODE
  WHEN MQCC-OK
    IF MQMD-MSGID EQUAL TO MQMI-NONE
      MOVE 'N' TO INPUT-SWITCH
    END-IF
    CONTINUE
  WHEN MQCC-WARNING PERFORM 900-WARNING THROUGH 900-EXIT
  WHEN OTHER PERFORM 910-ERROR THROUGH 910-EXIT
  GO TO 240-SEND-MAINMAP-ERROR
END-EVALUATE
```

The procedural code to complete an MQGET call.

Here we see an example of an MQGET call. The first to MOVEs are necessary to ensure retrieval of the “next message” from a queue. If the application were to exclude setting these variable before an MQGET, then ME would take the value in them and look for a specific message. We will cover this condition at the end of this section.

The MOVE of 20000 to the variable MQGMO-WAITINTERVAL set the time to wait for a message to arrive in the queue of 20 seconds. The COMPUTE statement sets the MQGMO-WAIT option to activate the “get with wait” function, plus to get the message outside of UOW control. That means the message will not be rolled back if the task abends. It also means that by default, the message when successfully retrieved will be removed from the queue.

We then see the actual MQGET call.

Finally, we see the EVALUATE statement to check completion condition of the call. Please note the check for MQMI-NONE, which is our method to check for existence of at least one message.

## MQ Usage in a COBOL Program

### PROCEDURE DIVISION (cont.):

```
COMPUTE MQ-OPTIONS = MQCO-NONE
CALL 'MQCLOSE' USING MQ-HCONN MQ-HOBJ-I MQ-OPTIONS
                    MQ-COMPCODE MQ-REASON

EVALUATE MQ-COMPCODE
  WHEN MQCC-OK CONTINUE
  WHEN MQCC-WARNING PERFORM 900-WARNING THROUGH 900-EXIT
  WHEN OTHER PERFORM 910-ERROR THROUGH 910-EXIT
  GO TO 240-SEND-MAINMAP-ERROR
END-EVALUATE
```

The procedural code to complete an MQCLOSE call.



On this slide we see an example of an MQCLOSE call. The setting of the MQ-OPTIONS is just to clear the variable to indicate a normal MQCLOSE call. This is followed by the actual MQCLOSE call itself.

As with previous calls, the EVALUATE statement is used to check the status of the call.



## MQ Usage in a COBOL Program

### WORKING-STORAGE SECTION:

```
01  MQ-MODULES_CICS.  
05  MQ-CLOSE          PIC X(08) VALUE 'CSQCCLOS'.  
05  MQ-CONN          PIC X(08) VALUE 'CSQCCONN'.  
05  MQ-CONNX         PIC X(08) VALUE 'CSQCCONX'.  
05  MQ-DISC          PIC X(08) VALUE 'CSQCDISC'.  
05  MQ-GET           PIC X(08) VALUE 'CSQCGET'.  
05  MQ-INQ           PIC X(08) VALUE 'CSQCINQ'.  
05  MQ-OPEN          PIC X(08) VALUE 'CSQCOPEN'.  
05  MQ-PUT           PIC X(08) VALUE 'CSQCPUT'.  
05  MQ-PUT1          PIC X(08) VALUE 'CSQCPUT1'.  
05  MQ-SET           PIC X(08) VALUE 'CSQCSET'.
```

These module names represent the actual load library module names.

It maybe desirable to code dynamic calls as opposed to static calls.

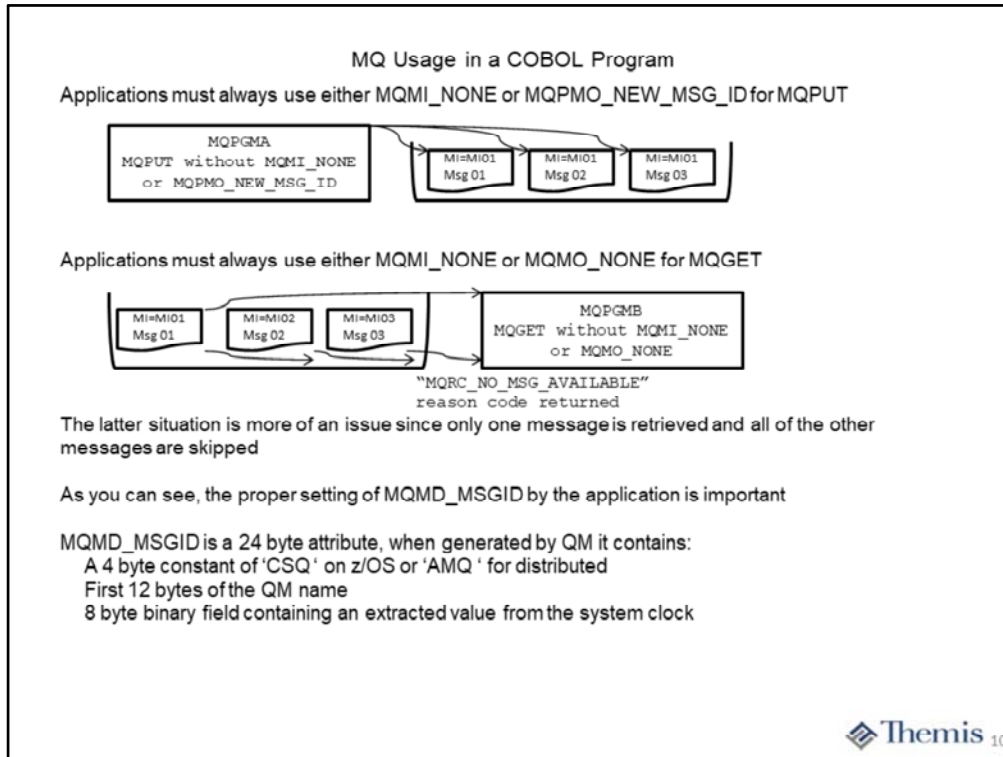
For example:

```
CALL MQ-PUT1 USING MQ-HCONN MQOD MQMD MQPMO  
              MQ-MSGOUT-LENGTH MQ-MSGOUT  
              MQ-COMPCODE MQ-REASON
```

To accomplish an MQPUT1 call under CICS.

This slide contains the variables with the values for each of the actual module names that can be used to code dynamic calls in the application. You will notice that the module names are different than the call module names we have seen earlier. This is why you cannot use the COBOL DYNAM option when compiling a CICS MQ application.

We use the MQPUT1 call to show an example of the minor change necessary to any MQ call to make it a dynamic call.



The MQMD\_MSGID will always be a unique, generated value as long as the applications do their part. The application is required to always set the attribute to MQMI\_NONE or include the option MQPMO\_NEW\_MSG\_ID for every MQPUT or MQPUT1 call. If they do not provide this code then the MsgId generated on the first call is reused for all subsequent calls. This is due to the fact that the generated MsgId is passed back to the application by MQ in the MQMD-MSGID attribute. This is intended to provide the application with a unique “tracking marker” for each message.

When retrieving messages the MQMD\_MSGID attribute should be set to MQMI\_NONE before the MQGET call, or the MQGMO\_MATCHOPTIONS attribute set with MQMO\_NONE. If the application does not provide this code then only the first message is retrieved and the second MQGET call issued will completely pass through the rest of the messages in the queue and return an error. This is due to the fact that with the first MQGET call the MsgId is populated with a valid value so when the second MQGET is issued the queue manager acts like you are searching for a particular message which will not be found.

# **A LOOK AT TRACE FOR THE APPLICATION PROGRAM**



### Trace for MQ Calls in the Application


```

00262 L800A AP 2520 ERM ENTRY COBOL-APPLICATION-CALL-TO-TRUE(MQM ) RET-2831E398 05:51:15.0459833125 00.0000327187 =001041=
00262 L800A RM 0301 RMLN ENTRY ADD_LINK_CLIENT_NAME(RMT) REMOTE_ACCESS_ID_BUFFER(26739654 , 00000000 , 00000008) LINK_ID_BUFFER
(00099B08 , 26A666EC , 00000008) RMC_TOKEN(26739600) LAST(NO) PRESUMPTION(ABORT) PRELOGGING
(NO) SINGLE_UPDATER(NO) COORDINATOR(NO) INITIATOR(NO) RECOVERY_STATUS(UNNECESSARY)
00262 L800A RM 0302 RMLN EXIT ADD_LINK_OK_LINK_TOKEN(010E0018) REMOTE_ACCESS_ID_BUFFER(26739654 , 00000000 , 00000008)
LINK_ID_BUFFER(00099B08 , 26A666EC , 00000008) LINK_ID() RET-80097B08 05:51:15.0460205625 00.0000372500 =001042=
00262 L800A AP 2522 ERM EVENT PASSING-CONTROL-TO-REQUIRED-TRUE(MQM ) RET-2831E398 05:51:15.0460432343 00.0000226718 =001043=
00262 L800A SM 0C01 SMMG ENTRY GETMAIN GET_LENGTH(4D68) SUSPEND(YES) INITIAL_IMAGE(00) STORAGE_CLASS(CICS) RET-2831E398 05:51:15.0460585000 00.0000152656 =001044=
00262 L800A SM 1201 SMLQ ENTRY ALLOCATE_PAGEPOOL_STORAGE SUBPOOL_TOKEN(2644A350) GET_LENGTH(4D80) SUSPEND(YES) RET-A9E7AD52 05:51:15.0460748750 00.0000163750 =001045=
00262 L800A SM 1202 SMLQ EXIT ALLOCATE_PAGEPOOL_STORAGE/OK ALLOCATED_LENGTH(5000) ADDRESS(290A4000) EXTENT_TOKEN(266DAC00) RET-26340E79 05:51:15.0460876408 00.0000127656 =001046=
RESUME_SYSTEM_TASK(NO) RET-26340E79 05:51:15.0460982187 00.0000105781 =001047=
00262 L800A SM 0C02 SMMG EXIT GETMAIN/OK ADDRESS_64(00000000_290A4008) RET-A9E7AD52 05:51:15.0467919843 00.0006937656 =001048=
00262 L800A AP A090 MQTRU ENTRY APPLICATION-REQUEST MQOPEN RET-2831E398 05:51:15.0468151093 00.0000231250 =001049=
00262 L800A AP A094 MQTRU EVENT CSQCOPND ABOUT TO ISSUE MQOPEN QUEUE(TH200) TRANSACT
) RET-2831E398 05:51:15.0474675781 00.0000524687 =001050=
00262 L800A AP A176 MQTRU EVENT ABOUT_TO_ISSUE_MQ_IDENTITY_TO_QMR(MQAA) RET-2831E398 05:51:15.0474853281 00.0000177500 =001051=
00262 L800A AP A177 MQTRU EVENT RETURN_FROM_MQ_IDENTITY RET-2831E398 05:51:15.1031758437 00.0556905156 =001052=
00262 L800A AP A095 MQTRU EVENT CSQCOPNH RETURN FROM MQOPEN RET-2831E398 05:51:15.1070199062 00.0038440625 =001053=
00262 L800A AP A091 MQTRU EXIT APPLICATION-REQUEST MQOPEN - MQCC(00000000) MQRC(00000000) RET-2831E398 05:51:15.1103231093 00.0000188281 =001070=
00262 L800A AP 2523 ERM EVENT REGAINING-CONTROL-FROM-REQUIRED-TRUE(MQM ) RET-2831E398 05:51:15.1103483750 00.0000252656 =001071=
00262 L800A AP 2521 ERM EXIT COBOL-APPLICATION-CALL-TO-TRUE(MQM ) RET-2831E398 05:51:15.1103691875 00.0000208125 =001072=

```

	Time in MQ interface	Time in MQ region	
	.1103691875	.1103483750	
	<u>.0459833125</u>	<u>.0474675781</u>	
	.0643858750	.0628807969	

CALL 'MQOPEN' being issued



This slide contains the portion of trace entries representing the MQOPEN call to access the specified queue. The bracketed entries on the left represent the total time spent in the CICS MQ interface. Most of this time (processor time) will actually be captured and reported on the MQ region side. This is because control is transferred from the MQ interface to MQ as a secondary address space very quickly. We have always classified the time spent between “ERM ENTRY” and “ERM EXIT” as time when the application has turned control over to the queue manager and entered a wait state.

The bracketed entries on the right represent the total time spent in the queue manager region itself. The “MQTRU ENTRY” trace entry occurs just before giving control to MQ. The “MQTRU EXIT” occurs just after MQ gives control back to the CICS MQ interface. This time is considered to be solely done on behalf of the queue manager region and the CICS application is in a wait state for its completion.

While the times shown on the slide might seem to be relatively low, on our small little mainframe we have executed around 1 million instructions to complete this MQOPEN call.

### Trace for MQ Calls in the Application


```

00262 L800A AP 2520 ERM ENTRY COBOL-APPLICATION-CALL-TO-TRUE(MQM ) RET-2831E4B0 05:51:15.1130470625 00.0000345468 =001081=
00262 L800A AP 2522 ERM EVENT PASSING-CONTROL-TO-REQUIRED-TRUE(MQM ) RET-2831E4B0 05:51:15.1130694218 00.0000223593 =001082=
00262 L800A AP A090 MQTRU ENTRY APPLICATION-REQUEST MQGET RET-2831E4B0 05:51:15.1130860000 00.0000165781 =001083=
00262 L800A AP A097 MQTRU EVENT CSQCQMGH ABOUT TO ISSUE MQGET RET-2831E4B0 05:51:15.1136101875 00.0005241875 =001084=
00262 L800A AP A098 MQTRU EVENT CSQCQMGH & CSQCQMGD - MESSAGE ID AND DATA RET-2831E4B0 05:51:15.1154803437 00.0018701662 =001085=
00262 L800A AP A091 MQTRU EXIT APPLICATION-REQUEST MQGET - MQCC(00000000) MQQC(00000000) RET-2831E4B0 05:51:15.1154928281 00.0000124843 =001086=
00262 L800A AP 2523 ERM EVENT REGAINING-CONTROL-FROM-REQUIRED-TRUE(MQM ) RET-2831E4B0 05:51:15.1155151093 00.0000222812 =001087=
00262 L800A RM 0301 RMLN ENTRY SET_LINK LINK_TOKEN(010E001B) LINK_ID_BUFFER(2673965C , 26A6666C , 00000008) SINGLE_UPDATER(YES) RET-80097C36 05:51:15.1155292031 00.0000140937 =001088=
RECOVERY_STATUS(NECESSARY) RET-80097C36 05:51:15.1155292031 00.0000140937 =001088=
00262 L800A RM 0302 RMLN EXIT SET_LINK_OK LINK_ID_BUFFER(2673965C , 26A6666C , 00000008) LINK_ID() RET-80097C36 05:51:15.1166660156 00.0005368125 =001089=
00262 L800A AP 2521 ERM EXIT COBOL-APPLICATION-CALL-TO-TRUE(MQM ) RET-2831E4B0 05:51:15.116775625 00.0005115468 =001090=

```

	.1165775625	.1155151093	
Time in MQ interface	.1130470625	.1130860000	Time in MQ region
	.0035305000	.0024291093	

CALL 'MQGET' being issued



This slide contains the portion of trace entries representing the MQGET call to retrieve the reply message from the queue. The bracketed entries on the left represent the total time spent in the CICS MQ interface. Most of this time (processor time) will actually be captured and reported on the queue manager region side. This is because control is transferred from the MQ interface to the queue manager as a secondary address space very quickly. We have always classified the time spent between “ERM ENTRY” and “ERM EXIT” as time when the application has turned control over to the queue manager and entered a wait state.

The bracketed entries on the right represent the total time spent in the queue manager itself. The “MQTRU ENTRY” trace entry occurs just before giving control to MQ. The “MQTRU EXIT” occurs just after DB2 give control back to the CICS MQ interface. This time is considered to be solely done on behalf of the queue manager and the CICS application is in a wait state for its completion.

While the times shown on the slide might seem to be relatively low, on our small little mainframe we have executed around 250,000 instructions to complete this MQGET call.

### Trace for MQ Calls in the Application


```

00262 L800A AP 2520 ERM ENTRY COBOL-APPLICATION-CALL-TO-TRUE(MQM ) RET-2831ESC8 05:51:15.1250901406 00.0000439687 =001111=
00262 L800A AP 2522 ERM EVENT PASSING-CONTROL-TO-REQUIRED-TRUE(MQM ) RET-2831ESC8 05:51:15.1251207031 00.0000305625 =001112=
00262 L800A AP A090 MQTRU ENTRY APPLICATION-REQUEST MQPUT1 RET-2831ESC8 05:51:15.1251425468 00.0000218427 =001113=
00262 L800A AP A098 MQTRU EVENT CSQCP10N & CSQCP2MD - ABOUT TO ISSUE MQPUT1 QUEUE(THM200.RESULTS ) RET-2831ESC8 05:51:15.1251425468 00.0000218427 =001113=
00262 L800A AP A09C MQTRU EVENT CSQCP1MI MESSAGE ID RET-2831ESC8 05:51:15.1251425468 00.0000218427 =001113=
00262 L800A AP A091 MQTRU EXIT APPLICATION-REQUEST MQPUT1 - MQCC(00000000) MQRC(00000000) RET-2831ESC8 05:51:15.1300504218 00.0041357812 =001115=
00262 L800A AP 2523 ERM EVENT REGAINING-CONTROL-FROM-REQUIRED-TRUE(MQM ) RET-2831ESC8 05:51:15.1300673750 00.0000169531 =001116=
00262 L800A AP 2521 ERM EXIT COBOL-APPLICATION-CALL-TO-TRUE(MQM ) RET-2831ESC8 05:51:15.1300996562 00.0000322812 =001117=
00262 L800A AP 2521 ERM EXIT COBOL-APPLICATION-CALL-TO-TRUE(MQM ) RET-2831ESC8 05:51:15.1301260625 00.0000264082 =001118=

```

	.1301260625	.1300996562	
Time in MQ interface	.1250901406	.1251425468	Time in MQ region
	.0050355469	.0049571094	

CALL 'MQPUT1' being issued



This slide contains the portion of trace entries representing the MQPUT1 call to put the request message to a queue. The bracketed entries on the left represent the total time spent in the CICS MQ interface. Most of this time (processor time) will actually be captured and reported on the queue manager side. This is because control is transferred from the MQ interface to the queue manager as a secondary address space very quickly. We have always classified the time spent between “ERM ENTRY” and “ERM EXIT” as time when the application has turned control over to the queue manager and entered a wait state.

The bracketed entries on the right represent the total time spent in the queue manager itself. The “MQTRU ENTRY” trace entry occurs just before giving control to the queue manager. The “MQTRU EXIT” occurs just after the QM gives control back to the CICS MQ interface. This time is considered to be solely done on behalf of the QM and the CICS application is in a wait state for its completion.

While the times shown on the slide might seem to be relatively low, on our small little mainframe we have executed around 500,000 instructions to complete MQPUT1 call.

Trace for MQ Calls in the Application


```

00262 L800A AP 2520 ERM ENTRY COBOL-APPLICATION-CALL-TO-TRUE(MQM ) RET-2831E480 05:51:15.1335078750 00.0000421718 =001127=
00262 L800A AP 2522 ERM EVENT PASSING-CONTROL-TO-REQUIRED-TRUE(MQM ) RET-2831E480 05:51:15.1335353906 00.0000275156 =001128=
00262 L800A AP A090 MQTRU ENTRY APPLICATION-REQUEST MQGET RET-2831E480 05:51:15.1335550468 00.0000196562 =001129=
00262 L800A AP A097 MQTRU EVENT CSQCQMGH ABOUT TO ISSUE MQGET RET-2831E480 05:51:15.1342130937 00.0006580468 =001130=
00262 L800A AP A085 MQTRU *EXC* CSQCQCCRE MQC(00000002) MQPC(000007F1) RET-2831E480 05:51:15.1352546250 00.0010415312 =001131=
00262 L800A AP A091 MQTRU EXIT APPLICATION-REQUEST MQGET - MQCC(00000002) MQPC(000007F1) RET-2831E480 05:51:15.1361856250 00.0009310000 =001132=
00262 L800A AP 2523 ERM EVENT REGAINING-CONTROL-FROM-REQUIRED-TRUE(MQM ) RET-2831E480 05:51:15.1362310312 00.0000454062 =001133=
00262 L800A AP 2521 ERM EXIT COBOL-APPLICATION-CALL-TO-TRUE(MQM ) RET-2831E480 05:51:15.1362588281 00.0000277968 =001134=

```

	.1362588281	.1362310312	
Time in MQ interface	.1335078750	.1335550468	Time in MQ region
	.0027509531	.0026759855	

CALL 'MQGET' being issued



This slide contains the portion of trace entries representing the MQGET call to retrieve a message from the queue. The bracketed entries on the left represent the total time spent in the CICS MQ interface. Most of this time (processor time) will actually be captured and reported on the queue manager side. This is because control is transferred from the MQ interface to queue manager as a secondary address space very quickly. We have always classified the time spent between “ERM ENTRY” and “ERM EXIT” as time when the application has turned control over to the queue manager and entered a wait state.

The bracketed entries on the right represent the total time spent in the queue manager itself. The “MQTRU ENTRY” trace entry occurs just before giving control to the queue manager. The “MQTRU EXIT” occurs just after the QM gives control back to the CICS MQ interface. This time is considered to be solely done on behalf of the QM and the CICS application is in a wait state for its completion.

While the times shown on the slide might seem to be relatively low, on our small little mainframe we have executed around 250,000 instructions to complete this MQGET call.


### Trace for MQ Calls in the Application

00262 L800A AP 2520 ERM	ENTRY COBOL-APPLICATION-CALL-TO-TRUE(MQM )	RET-2831E424 05:51:19.3258534687	00.0000552812 =001141=
00262 L800A AP 2522 ERM	EVENT PASSING-CONTROL-TO-REQUIRED-TRUE(MQM )	RET-2831E424 05:51:19.3258888437	00.0000353750 =001142=
00262 L800A AP A090 MQTRU	ENTRY APPLICATION-REQUEST MQCLOSE	RET-2831E424 05:51:19.3259418281	00.0000529843 =001143=
00262 L800A AP A096 MQTRU	EVENT CSQCLOSH ABOUT TO ISSUE MQCLOSE	RET-2831E424 05:51:19.3266977343	00.0007559062 =001144=
00262 L800A AP A091 MQTRU	EXIT APPLICATION-REQUEST MQCLOSE - MQCC(00000000) MQRC(00000000)	RET-2831E424 05:51:19.3317763906	00.0015242031 =001161=
00262 L800A AP 2523 ERM	EVENT REGAINING-CONTROL-FROM-REQUIRED-TRUE(MQM )	RET-2831E424 05:51:19.3318293125	00.0000529218 =001162=
00262 L800A AP 2521 ERM	EXIT COBOL-APPLICATION-CALL-TO-TRUE(MQM )	RET-2831E424 05:51:19.3318609218	00.0000316093 =001163=

Time in MQ interface	.3318609218	.3318293125	Time in MQ region
	<u>.3258534687</u>	<u>.3259418281</u>	
	.0060074531	.0058874844	

CALL 'MQCLOSE' being issued

 16

This slide contains the portion of trace entries representing the SQL SELECT statement to find the average and maximum SALARY. The bracketed entries on the left represent the total time spent in the CICS DB2 interface. Most of this time (processor time) will actually be captured and reported on the DB2 region side. This is because control is transferred from the DB2 interface to DB2 as a secondary address space very quickly. We have always classified the time spent between “ERM ENTRY” and “ERM EXIT” as time when the application has turned control over to DB2 and entered a wait state.

The bracketed entries on the right represent the total time spent in the DB2 region itself. The “MQTRU ENTRY” trace entry occurs just before giving control to the queue manager. The “MQTRU EXIT” occurs just after DB2 give control back to the CICS MQ interface. This time is considered to be solely done on behalf of the queue manager and the CICS application is in a wait state for its completion.

While the times shown on the slide might seem to be relatively low, on our small little mainframe we have executed around 500,000 instructions to complete this MQCLOSE call.



### Trace for MQ Calls in the Application

```

00262 L800A AP 00E1 EIP ENTRY RETURN REQ(0004) FIELD-A(24A0AAE0 . .\ ) FIELD-B(09000E08 ....)
                                BOUNDARY(0200) RET-AR319062 05:51:19.3346709687 00.0000251875 =001172=
00262 L800A AP E160 EXEC ENTRY RETURN COBOLII STMT_#(00375) RET-80085F4E 05:51:19.3347014062 00.0000304375 =001173=

00262 L800A RM FA11 RMUO ENTRY COMMIT_UOW CONTINUE(NO) RET-A635C698 05:51:19.3383370781 00.0000190468 =001188=

A 00262 L800A AP 2500 BRMSP ENTRY SEND_DO_COMMIT RMC_TOKEN(26739600) CONTINUE(NO) SINGLE_UPDATER(YES) LINK_TOKEN(010E001B) TRUE(MQM)
                                ) RET-A63D02DE 05:51:19.3436197187 00.0000061875 =001199=
00262 L800A AP 2509 BRMSP EVENT INVOKE_RMI TRUE(MQM) ) FOR ONLY UPDATER (UERTONLY) REQUEST RET-274898F8 05:51:19.3385262656 00.0000065468 =001200=
00262 L800A AP 2520 BRM ENTRY SYNCPOINT-MANAGER-CALL-TO-TRUE(MQM) ) RET-274898F8 05:51:19.3394118906 00.000886250 =001201=
00262 L800A AP 2522 BRM EVENT PASSING-CONTROL-TO-REQUIRED-TRUE(MQM) ) RET-274898F8 05:51:19.3394599531 00.0000480625 =001202=
00262 L800A AP A090 MQTRU ENTRY SYNCPOINT-MANAGER REQUEST RET-274898F8 05:51:19.3394971562 00.0000372031 =001203=
00262 L800A AP A0AA MQTRU EVENT CSQNDRT ABOUT TO ISSUE SINGLE PHASE COMMIT AND END THREAD RET-274898F8 05:51:19.3402617812 00.0007646250 =001204=
00262 L800A AP A091 MQTRU EXIT SYNCPOINT-MANAGER REQUEST RET-274898F8 05:51:19.3435634843 00.0033017031 =001205=
00262 L800A AP 2523 BRM EVENT REGAINING-CONTROL-FROM-REQUIRED-TRUE(MQM) ) RET-274898F8 05:51:19.3436066250 00.0000421406 =001206=
00262 L800A AP 2521 BRM EXIT SYNCPOINT-MANAGER-CALL-TO-TRUE(MQM) ) RET-274898F8 05:51:19.3436257500 00.0000201250 =001207=
00262 L800A AP 2510 BRMSP EVENT RETURN FROM RMI TRUE(MQM) ) WITH OK (UERFOK) RESPONSE RET-A63D02DE 05:51:19.3436367500 00.0000110000 =001208=
00262 L800A AP 2501 BRMSP EXIT SEND_DO_COMMIT/OK ACCESSIBLE(YES) VOTE(YES) TRUE(MQM) ) RET-A63D02DE 05:51:19.3436439062 00.0000071562 =001209=
                                )

00262 L800A AP 2500 BRMSP ENTRY PERFORM_COMMIT RMC_TOKEN(26739600) CONTINUE(NO) SINGLE_UPDATER(YES) COORDINATOR(YES) INITIATOR(NO)
                                RESTART(NO) UOW_STATUS(FORWARD) LINK_TOKEN(010E001B) PRESUMPTION(ABORT) RECOVERY_STATUS
                                (NECESSARY) TRUE(MQM) ) RET-A63D1F86 05:51:19.3437306662 00.0000053750 =001217=
00262 L800A AP 2501 BRMSP EXIT PERFORM_COMMIT/OK ACCESSIBLE(YES) FORGET(YES) PASS(YES) ABEND(NO) NEXT_RECOVERY_STATUS(UNNECESSARY)
                                TRUE(MQM) ) RET-A63D1F86 05:51:19.3437355468 00.0000048906 =001218=

00262 L800A AP 2520 BRM ENTRY CALL-TRUES-FOR-TASK-END RET-27489728 05:51:19.3446409687 00.0000430468 =001229=
00262 L800A AP 2522 BRM EVENT PASSING-CONTROL-TO-REQUIRED-TRUE(MQM) ) RET-27489728 05:51:19.3453838125 00.0007428437 =001224=
00262 L800A AP A090 MQTRU ENTRY TASK-MANAGER REQUEST RET-27489728 05:51:19.3454362187 00.0000524062 =001225=
00262 L800A AP A091 MQTRU EXIT TASK-MANAGER REQUEST RET-27489728 05:51:19.3455018125 00.0000655937 =001226=
00262 L800A AP 2523 BRM EVENT REGAINING-CONTROL-FROM-REQUIRED-TRUE(MQM) ) RET-27489728 05:51:19.3455317187 00.0000299062 =001227=
00262 L800A AP 2521 BRM EXIT CALL-TRUES-FOR-TASK-END RET-27489728 05:51:19.3455626093 00.0000308906 =001228=

00262 L800A RM FA12 RMUO EXIT COMMIT_UOW/OK FAILED_LINK(0000001E) RET-A635C698 05:51:19.3464612812 00.0000600937 =001236=

```

EXEC CICS RETURN issued



One last time through the CICS MQ interface as part of the EXEC CICS RETURN command being executed. CICS is the coordinator for committing modified data during execution of the application within the CICS region. There is a lot of activity going on during this process of ending and cleanup of the application. We have carved out the activity associated with the CICS MQ interface only.

The trace entries represented by connector **A**, represent the total time within the CICS MQ interface itself. The entries represented by the connector **B**, show the time the interface communicated with Resource Recovery Services (RRS) address space on z/OS. The entries represented by connector **C**, is the actual time spent in the queue manager itself for “prepare to commit”. The entries represented by connector **D**, is the actual time spent in the queue manager to perform its portion of the commit.

# **CEDF SCREENS FOR MQ INTERFACE**



CEDF Screens for MQ Calls

INVENTORY UPDATE VIA WMQ


ENTER USER ID -- THM200    REQUEST -- R    ITEM -- 10    QTY -- 10

ITEMNO	DESCRIPTION	PRICE	QONHAND	QSHIP	QBACORD
--------	-------------	-------	---------	-------	---------

HIT CLEAR / F3 WHEN COMPLETE

CALL =                    MQCC =                    MQRC =

Application screen for request entry



On this slide we can see the application map which allows for entry of the information that will produce the request message. This is a simple inventory update application where two CICS transaction, two queues and one VSAM file are involved. It is as close to a real production application we can get during a class workshop.

### CEDF Screens for MQ Calls

```
TRANSACTION: THZR PROGRAM: THMZ00R TASK: 0000353 APPLID: TCICSA3 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
001: ARG 000 ('.....&')
001: ARG 001 ('OD .....THMZ')
001: ARG 002 ('MD .....')
001: ARG 003 ('PMO .....')
001: ARG 004 ('...&...&.....')
001: ARG 005 ('I0000110000010 ')
001: ARG 006 ('.....I0')
001: ARG 007 ('.....I00001')
```

OFFSET: X'0058B6'                      LINE: UNKNOWN                      EIBFN=X'1802'

```
ENTER: CONTINUE
PF1 : UNDEFINED                      PF2 : SWITCH HEX/CHAR                      PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS              PF5 : WORKING STORAGE                      PF6 : USER DISPLAY
PF7 : SCROLL BACK                    PF8 : SCROLL FORWARD                      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY               PF11: EIB DISPLAY                          PF12: ABEND USER TASK
```

MQPUT1 call being issued



On this slide we see the MQPUT1 call about to execute. As you can see, there is no clear indication that it is an MQPUT1. We can determine it is the MQPUT1 because ARG 001 is the MQOD, ARG 002 is the MQMD, and ARG003 is the MQPMO. Please remember, ARG000 through ARG 007 represent the “call parameters” in the order which they are specified on the MQPUT1 call. Also, IBM does not show the complete data area represented by each argument, only the first 16 bytes.

### CEDF Screens for MQ Calls

```
TRANSACTION: THZR PROGRAM: THMZ00R TASK: 0000353 APPLID: TCICSA3 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
001: ARG 000 ('.....&')
001: ARG 001 ('OD .....THMZ')
001: ARG 002 ('MD .....')
001: ARG 003 ('PMO .....')
001: ARG 004 ('...&...&.....')
001: ARG 005 ('I0000110000010 ')
001: ARG 006 ('.....I0')
001: ARG 007 ('.....I00001')
```

OFFSET: X'0058B6'                      LINE: UNKNOWN                      EIBFN=X'1802'

```
ENTER: CONTINUE
PF1 : UNDEFINED                      PF2 : SWITCH HEX/CHAR                      PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS              PF5 : WORKING STORAGE                      PF6 : USER DISPLAY
PF7 : SCROLL BACK                    PF8 : SCROLL FORWARD                      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY               PF11: EIB DISPLAY                          PF12: ABEND USER TASK
```

MQPUT1 call complete



On this slide we can see the completion of the MQPUT1 call. Again, this determined by the MQOD, MQMD, and MQPMO being supplied. Also, IBM shows the first 16 bytes of each data area. However, ARG 006 is the completion code and ARG 007 the reason code. Both of these are binary full words. So to determine their value you have to hit the F2 key to switch to Hex. You will then be able to see the MQ returned completion and reason codes.

### CEDF Screens for MQ Calls

```
TRANSACTION: THZR PROGRAM: THMZ00R TASK: 0000353 APPLID: TCICSA3 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
001: ARG 000 ('.....&')
001: ARG 001 ('OD .....THMZ')
001: ARG 002 ('.....&...&...&')
001: ARG 003 ('.....&...&')
001: ARG 004 ('.....IO')
001: ARG 005 ('.....IO0001')
```

OFFSET: X'005686'            LINE: UNKNOWN            EIBFN=X'1802'

```
ENTER: CONTINUE
PF1 : UNDEFINED            PF2 : SWITCH HEX/CHAR            PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS    PF5 : WORKING STORAGE            PF6 : USER DISPLAY
PF7 : SCROLL BACK          PF8 : SCROLL FORWARD            PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY     PF11: EIB DISPLAY                PF12: ABEND USER TASK
```

MQOPEN call being issued



On this slide we can see the MQOPEN call about to be executed. The only indication of this is that ARG001 is the MQOD. The ARG 000 through ARG 005 represent the MQOPEN call parameters in the sequence which they were coded.

### CEDF Screens for MQ Calls

```
TRANSACTION: THZR PROGRAM: THMZ00R TASK: 0000353 APPLID: TCICSA3 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
001: ARG 000 ('.....&')
001: ARG 001 ('OD .....THMZ')
001: ARG 002 ('.....&...&...&')
001: ARG 003 ('.....&...&')
001: ARG 004 ('.....IO')
001: ARG 005 ('.....I00001')
```

OFFSET: X'005686'                      LINE: UNKNOWN                      EIBFN=X'1802'

```
ENTER: CONTINUE
PF1 : UNDEFINED                      PF2 : SWITCH HEX/CHAR                      PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS              PF5 : WORKING STORAGE                      PF6 : USER DISPLAY
PF7 : SCROLL BACK                    PF8 : SCROLL FORWARD                      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY               PF11: EIB DISPLAY                          PF12: ABEND USER TASK
```

MQOPEN call complete



On this slide we can see the completion of the MQOPEN call. Once again, ARG 004 is the completion code and ARG 005 the reason code. To view them you have to hit F2 to switch to Hex. Both are full word binaries, so only look at the first 4 bytes of ARG 004 and ARG 005.

### CEDF Screens for MQ Calls

```
TRANSACTION: THZR PROGRAM: THMZ00R TASK: 0000353 APPLID: TCICSA3 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
001: ARG 000 ('.....&')
001: ARG 001 ('.....&.&')
001: ARG 002 ('MD .....')
001: ARG 003 ('GMO .....+')
001: ARG 004 ('...&...&...&...')
001: ARG 005 ('000010PLAIN 100w')
001: ARG 006 ('...&.....')
001: ARG 007 ('.....I0')
001: ARG 008 ('.....I00001')
```

OFFSET: X'00579E'                      LINE: UNKNOWN                      EIBFN=X'1802'

```
ENTER: CONTINUE
PF1 : UNDEFINED                      PF2 : SWITCH HEX/CHAR                      PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS              PF5 : WORKING STORAGE                      PF6 : USER DISPLAY
PF7 : SCROLL BACK                    PF8 : SCROLL FORWARD                      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY               PF11: EIB DISPLAY                          PF12: ABEND USER TASK
```

MQGET with WAIT call being issued

On this slide we can see the MQGET call about to execute. We determined this by the fact that ARG 002 is the MQMD and ARG 003 is the MQGMO.



### CEDF Screens for MQ Calls

```
TRANSACTION: THZR PROGRAM: THMZ00R TASK: 0000353 APPLID: TCICSA3 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
001: ARG 000 ('.....&')
001: ARG 001 ('.....&..&')
001: ARG 002 ('MD .....')
001: ARG 003 ('GMO .....+'.')
001: ARG 004 ('...&...&...&...')
001: ARG 005 ('000011CLOSE THE ')
001: ARG 006 ('...&.....')
001: ARG 007 ('.....I0')
001: ARG 008 ('.....I00001')
```

OFFSET: X'00579E'                    LINE: UNKNOWN                    EIBFN=X'1802'

```
ENTER: CONTINUE
PF1 : UNDEFINED                    PF2 : SWITCH HEX/CHAR            PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS           PF5 : WORKING STORAGE           PF6 : USER DISPLAY
PF7 : SCROLL BACK                 PF8 : SCROLL FORWARD            PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY            PF11: EIB DISPLAY                PF12: ABEND USER TASK
```

MQGET with WAIT call complete

On this slide we can see the completion of the MQGET call. The completion code is ARG 007 and the reason code ARG 008. To see their values you have to switch to Hex mode.

### CEDF Screens for MQ Calls

```
TRANSACTION: THZR PROGRAM: THMZ00R TASK: 0000353 APPLID: TCICSA3 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
001: ARG 000 ('.....&')
001: ARG 001 ('.....&..&')
001: ARG 002 ('.....&..&..&')
001: ARG 003 ('.....I0')
001: ARG 004 ('.....I00001')
```

OFFSET: X'005712'            LINE: UNKNOWN            EIBFN=X'1802'

```
ENTER: CONTINUE
PF1 : UNDEFINED            PF2 : SWITCH HEX/CHAR            PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS    PF5 : WORKING STORAGE            PF6 : USER DISPLAY
PF7 : SCROLL BACK          PF8 : SCROLL FORWARD            PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY     PF11: EIB DISPLAY                PF12: ABEND USER TASK
```

MQCLOSE call being issued

On this slide we can see the MQCLOSE call about to execute. We can only determine this by the lack of any MQ structure being present in the arguments.

### CEDF Screens for MQ Calls

```
TRANSACTION: THZR PROGRAM: THMZ00R TASK: 0000353 APPLID: TCICSA3 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
001: ARG 000 ('.....&')
001: ARG 001 ('.....&..&')
001: ARG 002 ('.....&..&..&')
001: ARG 003 ('.....I0')
001: ARG 004 ('.....I00001')
```

OFFSET: X'005712'                      LINE: UNKNOWN                      EIBFN=X'1802'

```
ENTER: CONTINUE
PF1 : UNDEFINED                      PF2 : SWITCH HEX/CHAR                      PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS              PF5 : WORKING STORAGE                      PF6 : USER DISPLAY
PF7 : SCROLL BACK                    PF8 : SCROLL FORWARD                      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY               PF11: EIB DISPLAY                          PF12: ABEND USER TASK
```

MQCLOSE call complete



On this slide we can see the completion of the MQCLOSE call. The first full words of ARG003 and ARG 004 are the completion code and reason code, respectively.

CEDF Screens for MQ Calls

INVENTORY UPDATE VIA WMQ


ENTER USER ID -- THM200    REQUEST -- R    ITEM -- 10            QTY -- 10

ITEMNO	DESCRIPTION	PRICE	QONHAND	QSHIP	QBACORD
000010	PLAIN 100W LIGHT BULB	.99	0500040	0000000	0000000

HIT CLEAR / F3 WHEN COMPLETE

CALL = MQCLOSEI    MQCC =                    MQRC =

Application screen with returned information



Finally we see the application map being populated with the returned values from the replay message.

Our little application took the values entered on the top of the screen and constructed a “Request message”. It then issued an MQPUT1 call to put the message to the “transaction queue”, which cause MQ to trigger the QT00 transaction to execute as a background task. The QT00 application then retrieved the messages from the queue and updated the inventory VSAM file. It then issued an MQPUT1 to put the “Reply message” to the “results queue”. This caused the RT00 transaction (the one working with the map) to wake-up from its MQGET with wait, populate the screen and send the map back to the user.

# STATISTICS FOR MQ INTERFACE

CICS MQ Interface Statistics			
Applid TCICSA3	Sysid TCA3	Jobname TCICSA3	Date 03/18/2018 Time 08:46:10 CICS 7.0.0 PAGE 43
<u>WebSphere MQ Connection</u>			
MQCONN name . . . . .	: MQAA	websphere MQ connect date / time:	03/18/2018 08:25:13.46780
WebSphere MQ Connection Status . . . . .	: CONNECTED		
Mqname . . . . .	: MQAA		
WebSphere MQ Queue Manager Name . . . . .	: MQAA		
Resyncmember . . . . .	: YES		
WebSphere MQ Release . . . . .	: 8.00		
Initiation Queue Name . . . . .	: TCICSA3.INITQ		
Number of current tasks . . . . .	: 1		
Number of futile attempts . . . . .	: 0		
Total number of API calls . . . . .	: 49	API Crossing Exit Name . . . . .	: CSQCAPX
Number of API calls completed OK . . . . .	: 41	API Crossing Exit Concurrency Status . . . . .	: Threadsafef
Number of OPEN requests . . . . .	: 11		
Number of CLOSE requests . . . . .	: 8		
Number of GET requests . . . . .	: 19		
Number of GETWAIT requests . . . . .	: 11		
Number of GETWAITS that waited . . . . .	: 11		
Number of PUT requests . . . . .	: 0		
Number of PUTI requests . . . . .	: 11		
Number of INQ requests . . . . .	: 0		
Number of SET requests . . . . .	: 0		
Number of internal MQ calls . . . . .	: 72		
Number that completed synchronously . . . . .	: 56		
Number that needed I/O . . . . .	: 0		
Number of calls with TCB switch . . . . .	: 0		
<b>Statistics from CICS interval statistics</b>			



This slide contains the first portion of statistics for the CICS MQ Interface. The top portion (through “futile attempts”) contains basic information about the interface including the name of the queue manager, its MQ release, name of the initiation queue, plus the connection status and when the connection was established. We also see the current number of tasks using MQ and the number of calls issued while the connection was not active (futile attempts).


The next portion contains the total number of API calls issued, the number which have completed successfully, plus the name of the API crossing exit.

The next portion contains the number of API calls by each type, from OPEN (MQOPEN) to SET (MQSET) requests.

The lower portion provides the number of internal MQ calls issued, number completed successfully, number of I/Os required, and the number of times a TCB switch was necessary.

CICS MQ Interface Statistics	
Number of indoubt units of work . . . . :	0
Number of unresolved units of work . . . :	0
Number of resolved committed UOWs . . . :	0
Number of resolved backout UOWs . . . . :	0
Number of Backout UOWs . . . . . :	4
Number of Committed UOWs . . . . . :	4
Number of tasks . . . . . :	11
Number of Single Phase Commits . . . . . :	4
Number of Two Phase Commits . . . . . :	0
Number of CB requests . . . . . :	0
Number of msgs consumed . . . . . :	0
Number of CTL requests . . . . . :	0
Number of SUB requests . . . . . :	0
Number of SUBRQ requests . . . . . :	0
Number of STAT requests . . . . . :	0
Number of CRTMH requests. . . . . :	0
Number of DLTMH requests. . . . . :	0
Number of SETMP requests. . . . . :	0
Number of INMP requests. . . . . :	0
Number of DLTMP requests. . . . . :	0
Number of MHBUF requests. . . . . :	0
Number of BUFMH requests. . . . . :	0

Statistics from CICS interval statistics (cont.)



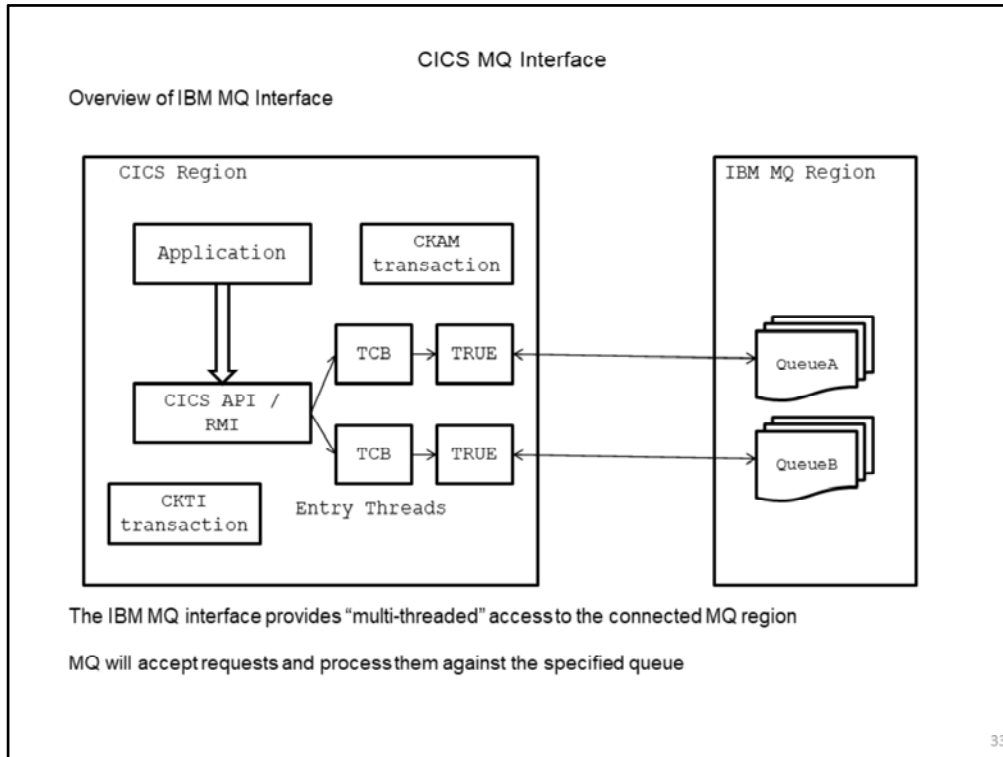
This slide show the rest of the MQ interface statistics. The first 4 lines are statistics related to recovery processing as required during MQ interface start up. Unless the CICS region was not terminated abnormally, there should be no values in these 4 lines.

The next 2 lines show the number of tasks which were Backed out or Committed. This is assuming that not task issued a EXEC CICS SYNCPOINT or SYNCPOINT ROLLBACK command. We then see the Number of tasks perform along with the number of single or 2 phase commits performed.

The last 13 line provide statistics about the number of each of the unusual MQ calls issued. These calls represent new features added in MQ Version 7 and are not widely used.

# MQ RESOURCE DEFINITIONS





This slide provides a simple view of how CICS will interface to resource managers like IBM MQ or DB2. The key to working with other resource managers is the **Task Related User Exit** or **TRUE** programs. These are provided by CICS to provide the cross region communication with the resource manager in the other address space.

The CICS application will execute the command to access one of the defined resource managers. These commands are the normal API functions expected by that resource manager, which include:

- CALL 'MQxxx' – for WebSphere MQ requests
- EXEC SQL – for DB2 requests
- EXEC DLI – for IMS requests

CICS maintains a pool of Task Control Blocks or TCBs to process these requests on, thus freeing up the normal CICS TCBs to perform other work. There is an SI parameter, MAXOPENTCBS, which specified the number of TCBs CICS could use. All of the work performed on these TCBs is "threadsafe" to minimize TCB switching and improve performance.

The **CKAM** transaction is the "alert monitor" transaction for the MQ adapter. The transaction handles unscheduled events which are produced by the queue manager. This includes such events like the queue manager being shutdown or waiting for the queue manager to start up.

The **CKTI** transaction is the CICS trigger monitor. The trigger monitors in MQ perform MQGET with wait on a define initiation queue (INITQ) associated with the application. In our case for the CICS region that CKTI is running in. When a trigger event occurs, the queue manager puts a message into the INITQ. This will wake up CKTI, which then analyzes the message contents (it is known as a trigger message) and will start the application it indicates.

## MQ Resource Definitions

### Define MQCONN:

```
DEFINE MQCONN(MQAA) GROUP(THEMIS00)  
  DESCRIPTION(SPECIAL TEST CONNECTION)  
  INITQNAME(TCICSA3.INITQ) MQNAME(MQAA)
```

INITQNAME(TCICSA3.INITQ) – specifies the name of the “initiation queue” to be used by the CICS MQ trigger monitor

MQNAME(MQAA) specifies the name of the queue manager to which CICS should establish the connection

The MQ interface will utilize the OPENAPI interface and dynamically create and allocate thread TCBs up to the MAXOPENTCBS limit

Two permanent CICS transactions will be started when the connection is made:

- CKTI – the CICS MQ trigger monitor
- CKAM – the MQ alert monitor

You may also supply an “API crossing exit” to interact with the MQ calls, sample exit supplied is CSQCAPX



This slide shows the only resource definition associated with the CICS MQ interface. It serves to define the connection to the desired queue manager. CICS can only be connected to one queue manager at a time. To connect to a different queue manager you would have to stop the interface, change this resource, and start the interface once again.

The option **MQNAME(MQAA)** specifies the name of the queue manager with which CICS will establish a connection, maximum of 4 characters. The queue manager must be on the same LPAR where CICS is running.

The option **INITQNAME(TCICSTA3.INITQ)** specifies the name of the initiation queue the CICS trigger monitor will use. This queue must be defined in the queue manager and specified in the INITQ option for application queues. CICS will automatically start the CKTI transaction, passing it this name to use in its processing.

Unlike the CICS DB2 Interface, there is no option to limit the number of “Open TCBs” the CICS MQ Interface will use. We know that it will may approach the value you specify in MAXOPENTCBS. The limiting factor is the number of currently in use TCBs by the CICS DB2 interface.

## MQ Resource Definitions

### Define Initiation Queue for CICS Trigger Monitor:

```
DEFINE QLOCAL( 'TCICSA3.INITQ' ) REPLACE +  
  DESCR( 'TCICSA3 CKTI initiation queue' ) +  
  SHARE  DEFSOPT( SHARED ) +  
  MSGDLVSQ( FIFO ) +  
  MAXDEPTH( 1000 ) MAXMSGL( 4000 ) +  
  STGCLASS( 'SYSVOLAT' ) TRIGTYPE( NONE )
```

QLocal definition for initiation queue used by the CICS trigger monitor



Here we see the QLocal definition for the “initiation queue” in support of MQ triggering in a CICS region. At minimum there must one of these for each CICS region you define the MQCONN resource. However, it is possible to have more than one CKTI (the trigger monitor) task running within the same region. This means you may need additional initiation queues to support them.

We have added the SHARE and DEFSOPT(SHARED) options to ensure that multiple concurrent CKTI task could access the same initiation queue if necessary.

The MAXDEPTH and MAXMSGL options have been set to more than is necessary for an initiation queue. Since the trigger monitor is processing trigger messages as fast as they arrive, the depth should never approach anywhere near the 1000 message limit. The trigger message is currently less than 1000 bytes, so 4000 byte size limit is 4 times what is necessary.

You should always specify MSGDLVSQ (message delivery sequence) as FIFO (first in first out). This takes functions like priority message processing out of the way and helps ensure that the trigger monitor can process the trigger messages as soon as they arrive.

## MQ Resource Definitions

### Define Initiation Queue for CICS Trigger Monitor:

```
DELETE QLOCAL( 'THMZ00.TRANSACT' ) PURGE
DEFINE QLOCAL( 'THMZ00.TRANSACT' ) REPLACE +
  DESCR( 'Test queue for requests' ) +
  MAXDEPTH( 100 ) +
  SHARE DEFSOPT( SHARED ) +
  TRIGGER TRIGTYPE(FIRST) +
  PROCESS('THMZ00.EXECCICS') INITQ('TCICSA3.INITQ')
```

```
DELETE QLOCAL( 'THMZ00.RESULTS' ) PURGE
DEFINE QLOCAL( 'THMZ00.RESULTS' ) REPLACE +
  DESCR( 'Test queue for replies' ) +
  MAXDEPTH( 100 ) +
  SHARE DEFSOPT( SHARED )
```

```
DELETE PROCESS( 'THMZ00.EXECCICS' )
DEFINE PROCESS('THMZ00.EXECCICS') REPLACE +
  DESCR( 'PROCESS TO TRIGGER TRANSACTION QT00' ) +
  APPLTYPE(CICS) APPLICID(QT00)
```

The QLocal definitions utilized by the application program.

The PROCESS definition for the triggered transaction which executes in the background.



This slide contains the queue and process definitions we used in support of our little inventory application. The **THMZ00.TRANSACT** queue is the one where the “Request message” was put with the input from the initial screen to pass to the QT00 update transaction. This is why you see the “trigger” attributes populated plus the PROCESS and INITQ options.

The **THMZ00.RESULTS** queue is the one which the initiating transaction, RT00, will issue the MQGET with wait against, plus the queue where the “Reply message” will be put by the QT00 transaction.

The **THMZ00.EXECCICS** is the process definition which represents the QT00 transaction to be triggered.

# **SVC DUMP INFORMATION FOR MQ INTERFACE**

### MQ Interface Information in an SVC Dump

```

CICS DUMP: SYSTEM=TCICSA3 CODE=MT0001 ID=4/0001                22 07:50:50 03/19/18

==MQ: CICS/MQ - SUMMARY

==MQ: GLOBAL STATE SUMMARY

Connection status:                Connected
In standby mode:                  NO
Mqname:                           MQAA
Qmgr:                             MQAA
WebSphere MQ release:            0800
Initiation Queue:                 TCICSA3.INITQ
API Crossing exit active:         NO
Date/time connection completed:   18/03/18 13:25:13
Date/time disconnection completed:

==MQ: TRIGGER MONITOR TRANSACTIONS SUMMARY


Tran Task  TcaAddr  TieAddr  LotAddr  ThrAddr  Uowid      Tcb
Id  num
-----
CKTI 00051 267A2800 28422220 284222A0 284222F8 D40C00E5CE0D45C3 No

==MQ: ALL TRANSACTIONS SUMMARY

Tran Task  TcaAddr  TieAddr  LotAddr  ThrAddr  Uowid      Tcb
Id  num
-----
CKTI 00051 267A2800 28422220 284222A0 284222F8 D40C00E5CE0D45C3 No

The key portion is the "TRANSACTION SUMMARY"

```



This slide contains the “CICS MQ Interface Summary” from an SVC dump. The upper portion simply contains information about the MQ connection. This includes the name of the queue manager name involved and its version. Another key piece of information is the name of the specified initiation queue. You can also see the current status of the connection.

The “TRANSACTION SUMMARY” show all of the tasks in the CICS region which have or are currently accessing MQ resources. Much of the information is of minimal interest, such as TcaAddr, TieAddr, and Uowid. The LotAddr is the control blocks used for the task during execution. The Tcb in MQ indicates if the task currently has a MQ request outstanding.

We have ignored the “TRIGGER MONITOR TRANSACTION SUMMARY” since the CKTI transactions will also show the same information in the “All TRANSACTION SUMMARY”.

### MQ Interface Information in an SVC Dump

```

DFHMQINI 27752D00 CICS MQINI BLOCK
  Definetime 2018/03/07 11:52:25   Definesource MQAA
  Changetime 2018/03/07 11:52:25   Changeusrfd CICSUSER   Changeagent DYNAMIC   Changeagrel 0700
  Installtime 2018/03/07 11:52:25   Installusrfd CICSUSER   Installagent DYNAMIC
0000 00A06E24 C6C8D4D8 C9D5C940 40404040 C4C6C8D4 D8C9D5C9 E3C3C9C3 E3C1F348 *...DFHMQINI DFHMQINITCICSA3.* 27752D00
0020 C3D5C9E3 D8404040 40404040 40404040 40404040 40404040 40404040 40404040 *INITQ " " 27752D20
0040 40404040 40404040 0000051C 00000000 D40C00E5 CE1BFAC0 00000000 00000000 " " .....M...V...{....." 27752D40
0060 D40BC1C1 40404040 D3FE17A1 04F3FC00 D3FE17A1 04F3FC00 C3C9C3E2 E4E2C5D9 *MQAA L...3..L...3..CICSUSER* 27752D60
0080 000BF0F7 F0F0D3FE 17A1D4F3 FC00C3C9 C3E2E4E2 C5D90008 00000000 00000000 *..0700L...3..CICSUSER....." 27752D80

Task number of trigger monitor: 00051
Date/time trigger monitor started: 18/03/18 13:25:16
Date/time trigger monitor stopped: 18/03/18 13:25:16

LOA.MQH 2843B210 CICS/MQ LIFE OF ADAPTER BLOCK
0000 C3010020 C3D3D6C1 01000000 800330E8 00000000 00010001 00010000 27752A50 *C...CLOA.....&" 2843B210
DFHMQIC 27752A50 CICS/MQ LIFE OF CONNECTION BLOCK
0000 C30201A8 C3D3D6C3 02000000 00000000 00181E05 D4D8C1C1 E3C3C9C3 E2C1F340 *C...YLOC.....MQAATCICSA3 " 27752A50
0020 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 " " " 27752A70
0040 00000000 00000000 00000000 2843B210 D40BC1C1 00000000 00000000 00000000 " " .....MQAA....." 27752A90
0060 00000001 00000000 284222A0 58795FC8 A9FE1830 000001F4 F0F7F0F0 00000001 " " .....^Hz.....40700....." 27752AB0
0080 00000000 00000000 00000009 0000000E 00000008 00000000 00000002 00000000 " " " 27752AD0
00A0 00000002 00000002 00000004 00000002 00000000 00000002 00000000 00000000 " " " 27752AF0
00C0 00000000 00000000 00000000 00000000 00000000 00000000 00000001 00000002 " " " 27752B10
00E0 00000001 00000000 284222F8 284222F8 00000005 00000000 27752BC8 F0F8F0F0 " " .....S...S.....H0B00" 27752B30
0100 008FC100 00000047 00000000 00000000 00000000 00000000 00000000 00000000 " " .....A....." 27752B50
0120 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000320 " " " 27752B70
0140 A9FA73D8 D9509FE0 D4D8C1C1 40404040 40404040 40404040 40404040 40404040 "z...QR&..MQAA " 27752B90
0160 40404040 40404040 40404040 40404040 40404040 40404040 C1010030 C1C2D3D2 " " .....A...ABLK" 27752BB0
0180 00000000 29FC3D00 00000000 58795FC8 00000000 A9F7C968 008A7218 000001F4 " " .....}.....^H...z7I.....4" 27752BD0
01A0 29FE4748 00000000 " " " 27752BF0
  
```

The 2 control blocks used for the life of the CICS MQ interface

No longer documented, IBM dropped the Supplementary Data Areas manual



- The first data area is the DFHMQINI or MQ initialization control block.
- +0 Half-word length of the data area
  - +2 14 byte identifier of the data area
  - +18 Name of the initiation queue
  - +48 Task number of the CKAM monitor task
  - +60 Name of the MQCONN resource definition

The second data area is the "LIFE OF ADAPTER" data area. There is not much information of use to use in the data area.

- The third data area is the "LIFE OF CONNECTION".
- +14 Sub-system name of the queue manager
  - +18 CICS Applid
  - +50 Sub-system of the connected QM

Most of the rest of the data is the statistics which are written to SMF.

### MQ Interface Information in an SVC Dump

```
DFHMQLOT 284222A0 CICS/MQ LIFE OF TASK BLOCK
0000 01706EC4 C6C8D4D8 D3D6E340 40404040 00000000 00000000 00000000 2843B230 *..>DFHMQLOT .....* 284222A0
0020 C3D2E3C9 0000051C C3C9C3E2 E4E2C5D9 29182030 29182030 000019C0 00400000 *CKTI...CICSUSER.....{;..* 284222C0
0040 278B8194 284222F8 267A2800 D40C00E5 CEF046C3 00000009 00806EC4 C6C8D4D8 *..m...B...M...V...C.....>DFHMQ* 284222E0
0060 E3C8C440 40404040 00148000 00000000 00000000 00000000 00000000 284222A0 *THD .....* 28422300
0080 284222A0 80033108 00000000 00000000 00000000 00000000 28422328 00000000 *.....* 28422320
00A0 00000000 00000000 09000003 29183318 C6D9C240 00050001 29182DA0 00010000 *.....FRB .....* 28422340
00C0 000007F1 00000000 00181E05 00010000 00000000 00000000 00000000 00000000 *...I.....* 28422360
00E0 00000000 00000000 C3080020 C3D4D7C1 00000000 00000000 5879EFC8 0000002F *...C...MPA.....M.....* 28422380
0100 000001F4 00000000 E3C3C3C3 E2C1F348 C9D5C9E3 D8404040 40404040 40404040 *...4....TCICSA3..INITQ .....* 284223A0
0120 40404040 40404040 40404040 40404040 40404040 E3C8D4E9 F0F04BE3 *.....THM200..T* .....* 284223C0
0140 D9C1D5E2 C1C3E340 40404040 40404040 40404040 40404040 40404040 *RANSACT .....* 284223E0
0160 40404040 40404040 29139088 00000000 .....* .....* 28422400
```

Thread information:

```
STATUS = GetWait
FRBRC1 = 0000
FRBRC2 = 000007F1
FRBFBACK = 00000000
INITQ = TCICSA3..INITQ
APPLQ = THM200..TRANSACT
```

The control block used while the task is executing

No longer documented, IBM dropped the Supplementary Data Areas manual



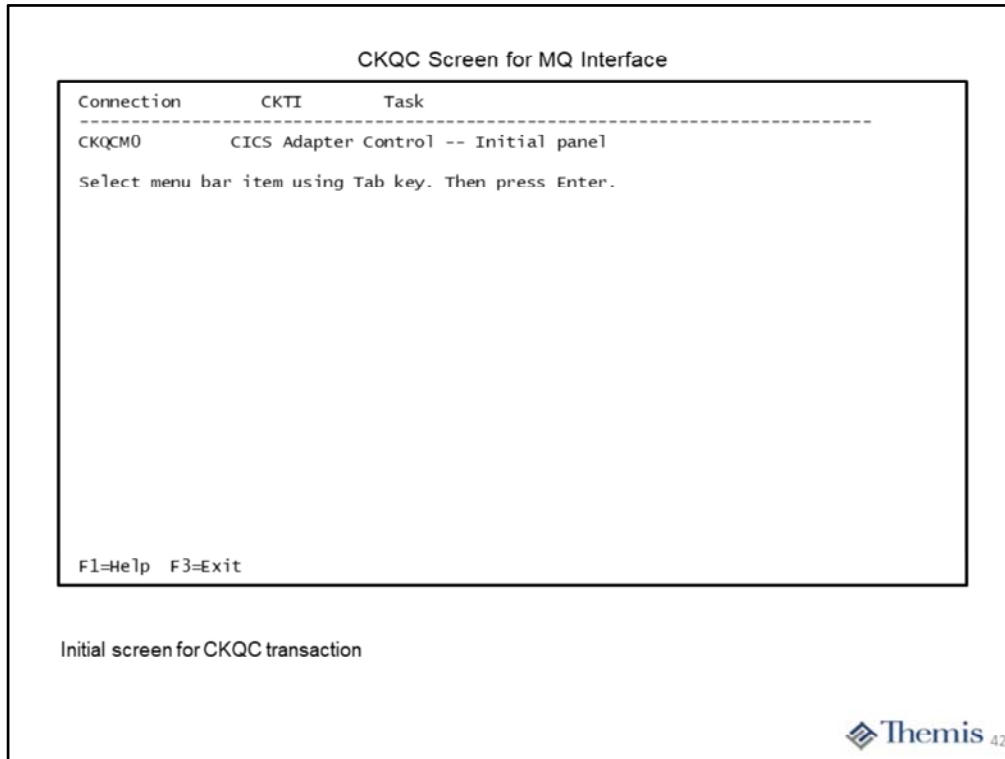
Here we find the Life Of Task (LOT) control block. The data area is used while the task is execution to manage the application making requests to the queue manager. The data area is no longer documented. However, the last Supplemental Data Areas manual was for CICS TS V4.1 and the control block seems to have not changed with subsequent releases.

The LOT basic format is as follows:

- +0 Half-word length of data area
- +2 14 character eye-catcher
- +20 CICS transaction ID
- +24 CICS task number
- +28 User ID
- +48 TCA address
- +108 Name of the initiation queue if the LOT represents a CKTI transaction
- +138 Name of the queue accessed by last MQ call if LOT is for a user transaction



# CKQC FUNCTIONS




This is the initial screen for the CKQC transaction. The three items at the top of the screen; Connection, CKTI and Task; are drop down menus with more selection choices.

CKQC Screen for MQ Interface

Connection	CKTI	Task
-----		
Select an action.		apter Control -- Initial panel
4 1. Start...		sing Tab key. Then press Enter.
2. Stop...		
3. Modify...		
4. Display		
-----		
F1=Help F12=Cancel		
-----		
DFHM00424 I TCICSA3 Invalid key entered.		
F1=Help F3=Exit		

Drop down menu for "Connection" option



This drop down is shown by putting the cursor on the "Connection topic" and hitting the enter key. You will notice the four choices, and we have entered a "4" to go to the "Display" function.


**CKQC Screen for MQ Interface**

```

CKQCM2                Display Connection panel
Read connection information. Then press F12 to cancel.

  CICS Applid = TCICSA3  Connection Status = Connected  QMgr name= MQAA
  Mqname =      MQAA     Tracing           = On         API Exit = Off
  Initiation Queue Name = TCICSA3.INITQ
----- STATISTICS -----
Number of in-flight tasks = 1      Total API calls      =      4393
Number of running CKTI   = 1
  APIs and flows analysis              Syncpoint              Recovery
-----
Run OK          4381  MQINQ          355  Tasks          316  Indoubt       0
Futile         0    MQSET           44  Backout        0    UnResol      0
MQOPEN        485  ----- Flows -----  Commit         201  Commit       0
MQCLOSE       471  Calls          4393  S-Phase       201  Backout      0
MQGET         1214  SyncComp       4393  2-Phase       0
GETWAIT       306  SuspReqd       0
MQPUT         1022  Msg Wait       12
MQPUT1        673  Switched       4358
-----
F1=Help  F12=Cancel  Enter=Refresh
  
```

Display screen from "Option 4" on Connection drop down



This slide shows the output of the Display function. It is set of general information about the CICS MQ interface. On the top portion we see the CICS region name, connection status, queue manager name, MQCONN resource name, and the name of the initiation queue.

In the center portion we see the current number of tasks in-flight, total MQ calls issued, and the current number of running CKTI transactions.

The lower portion contains a break down of the MQ calls issued to this point in time. The MQOPEN, MQCLOSE, MQGET, GETWAIT, MQPUT, MQPUT1, MQING, and MQSET and simply the number of each call executed. The GETWAIT is the number of MQGET calls issued with the MQGMO-WAIT option.

The Run OK value represent the number of calls which have completed successfully. The value in Futile represents the number of MQ calls issued while the connection was not active.

Under the "Syncpoint" category, Tasks is the total number of tasks using MQ to this point in time. The Backout and Commit are counts for the tasks which have been rolled back and committed.

Under the "Recovery" category, Indoubt is the number of UOWs which were indoubt at MQ interface start up. The UnResol is the number which were not resolved. The Commit is the number resolved by a Commit and Backout is the number resolved by Backout (rollback).


Under the "Flows" category, Calls is the number of flows to the queue manager on the connection, and SyncComp is the number which have completed synchronously. The values SuspReqd and Msg wait indicate the number flows which had to wait for completion of the MQ call. The value of Switched represents the number of calls where a TCB switch occurred.

CKQC Screen for MQ Interface

Connection	CKTI	Task
Select an action.		apter Control -- Initial panel
3 1. Start...		sing Tab key. Then press Enter.
2. Stop...		
3. Modify...		
4. Display		
F1=Help F12=Cancel		
		Modification options Select modify option. Then press Enter. 1. Reset statistics 2. Enable API Exit 3. Disable API Exit F1=Help F12=Cancel
		F1=Help F12=Cancel

F1=Help F3=Exit

Drop down from Connection and then "Option 3" pop-up selection




From the Connection drop down menu, we now choose option 3 to Modify the connection. Here you see the pop-up menu with the modify options. We have a limited set of choices on what to modify. We can simply reset the statistic discussed earlier, and either enable or disable the API crossing exit.

CKQC Screen for MQ Interface

Connection	CKTI	Task
-----		
Select an action.		apter Control -- Initial panel
2 1. Start...		sing Tab key. Then press Enter.
2. Stop...		
3. Modify...		-----
4. Display		Stop Connection
-----		
F1=Help F12=Cancel		Select stop type.
-----		
		Then press Enter.
		1. Quiesce
		2. Force
-----		
		F1=Help F12=Cancel
-----		
F1=Help F3=Exit		

Drop down from Connection and then "Option 2" pop-up selection




We then choose option 2 on the Connection drop down menu, Stop the MQ connection. Here you see the pop-up for the Stop function. We either choose to "Quiesce" or "Force" stop the connection. Quiesce will allow all currently active tasks to complete and disallow any new MQ tasks from starting. Force will cause the abnormal termination of any current tasks and disallow any new MQ tasks from starting.

CKQC Screen for MQ Interface

Connection	CKTI	Task
Select an action.	apter Control -- Initial panel	
1. Start...	sing Tab key. Then press Enter.	
2. Stop...		<div style="text-align: center;">start a Connection</div>
3. Modify...		Type parameters. Then press Enter.
4. Display		1. Queue manager name (SN) . . . MQAA
F1=Help F12=Cancel		2. Initiation Queue Name (IQ) . . . . . TCICSA3.INITQ
		F1=Help F12=Cancel
F1=Help F3=Exit		

Drop down from Connection and then "Option 1" pop-up selection




Finally we choose option 1 from the Connection drop down menu, Start the connection. Here you see the pop-up menu for the Start function. We are required to enter two values, 1 is the name of the queue manager to connect with, and 2 is the name of the initiation queue the CKTI transaction is to use. The queue manager name may be different than the one specified by the MQCONN resource definition.

CKQC Screen for MQ Interface

Connection	CKTI	Task
CKQCM0	CIC	Select an action. Initial panel
Select menu bar it	3	1. Start... 2. Stop... 3. Display
		press Enter.
		F1=Help F12=Cancel

F1=Help F3=Exit

Drop down from CKTI option



On this slide we have placed the cursor on the CKTI option on the top menu line and hit enter. You are presented with the CKTI drop down menu. You will notice the three possible choices. We have picked option 3 to display the current CKTI information.



CKQC Screen for MQ Interface

CKQC M4                      Display CKTI panel


Read CKTI status information. Then press F12 to cancel.

CKTI 1 to 1 of 1

Task Num	Task Status	Thread Status	Num of APIs	Last API
0000051	Normal	Msg Wait	9	MQGET
Initiation Queue Name: TCICSA3.INITQ				

DFHMQ0461 I TCICSA3 Top of display.  
F1=Help F7=Backward F8=Forward F12=Cancel Enter=Refresh

Display screen from "Option 3" on CKTI drop down



Here is the Display screen for the current CKTI transaction(s). We see only one CKTI running in the CICS region at the moment. However, we can start up more CKTI transactions as required. There is not much information on this screen. Simply the CICS task number, status, number of API calls issued, and the name of the initiation queue.

CKQC Screen for MQ Interface


Connection	CKTI	Task
CKQCM0	CIC	Select an action. Initial panel
Select menu bar it	2	1. Start... 2. Stop... 3. Display

F1	Stop Task Initiator
	Type Initiation Queue Name. Then press Enter.
	Initiation Queue Name (IQ) . . . . .
	F1=Help F12=Cancel

F1=Help F3=Exit

Drop down from CKTI and then "Option 2" pop-up selection



Now we choose option 2 from the CKTI drop down menu, Stop the CKTI task. Here you see the pop-up menu where you specify the name of the initiation queue for the CKTI task you want to stop. The assumption is that every CKTI task will be using a different initiation queue.

CKQC Screen for MQ Interface

Connection	CKTI	Task
CKQCM0	CIC	Select an action. Initial panel
Select menu bar it	1	1. Start... 2. Stop... 3. Display


  

START TASK INITIATOR	
F1	Type Initiation Queue Name. Then press Enter.
	Initiation Queue Name (IQ) . . . . .
	F1=Help F12=Cancel

DFHM0424 I TCICSA3 Invalid key entered.

F1=Help F3=Exit

Drop down from CKTI and then "Option 1" pop-up selection



Finally we chose option 1 on the CKTI drop down menu, Start a CKTI transaction. Here you see the pop-up menu for the Start function. You need only specify the name of the initiation queue that the CKTI transaction will use. You can start more than one CKTI transaction by simply specifying the same or a different initiation queue.

**CKQC Screen for MQ Interface**

CKQC M3                      Display Task panel


Read task status information. Then press F12 to cancel.

Tasks 1 to 1 of 1

Tran Id	User Id	Task Num	Task Status	Thread Status	Total APIS	Res Sec	API Exit	Last MQ call	Thread ID
CKTI	CICSUSER	00051	Normal	Msg Wait		9 No	No	MQGET	284222A0

F1=Help F7=Backward F8=Forward F12=Cancel Enter=Refresh

Display screen from the "Task" option on the CKQC main screen



This screen is obtained by putting the cursor on the Task topic at the top of the initial screen and hitting enter. This display shows a list of all of the current tasks which has performed MQ calls. You will see any of your tasks , plus the CKTI tasks currently running in the CICS region. On this display you find the basic information about the task, plus its current status. The Res Sec column indicates if "resource security" is active for the transaction. The API Exit column indicates it the API crossing exit is in effect for the transaction. You will also see the "Last MQ call" issued and the internal Thread ID for the task.