Advanced scripting with MQSCX

T.Rob Wyatt, IoPT Consulting
t.rob@ioptconsulting.com

Advanced scripting with MQSCX - Intermediate

If you do any MQSC scripting at all, you are probably used to running discrete commands and parsing the output in BASH or KSH. Thanks to popular demand and feedback from the community, MQSCX from MQ Gem now has extensive scripting support and can slash script development time and complexity. For the first time scripting tasks that should be simple actually are. Automation can make it easier for MQ admins and other stakeholders to do the right thing than to take shortcuts that leave the network vulnerable. MQSCX makes that automation accessible to everyone, which in turn helps keep the network secure. Show up for the MQSCX training or show up to see some really cool scripts and automation. Either way you will leave with some ideas on how to automate some tasks in your own MQ network.

Agenda

- A word about…
- Download this session!
- Native tools lack some usability features.
- Comparison: KSH and MQSCX versions of queue depth report
- Cluster consistency checker
- CCDT file automation
- Q&A
- File listings (in case you just can’t wait to download them).

A word about MQSCX and MQGem

- **MQGem is Paul Clarke's company started after leaving the MQ Dev team**
  - Selling MQMon and MQSCX, plus some free stuff and much more on the way.
  - Go to http://mqgem.com for more information, trial versions and to buy licenses.
  - The new and improved MQGem – now with Morag Hughson!

- **MQSCX is the successor to SupportPac MO71**
  - MQSCX is a generation or two updated from MO71

- **MQSCX is a superset of the functionality of RUNMQSC**
  - Variables!
  - Loops!
  - True regex!
  - File read/write/include!
  - Lots of really cool UI features we won't cover because this is about scripting and because you didn't dedicate the whole day to MQSCX, which we'd need to do it justice.

Bottom line:
MQSCX makes scripting MQ Admin tasks easy.

A word about IoPT

T.Rob left IBM to form IoPT Consulting in 2013.

MQ Security Guy

The firm offers the same MQ consulting services as before:

- Security (of course!)
- Architecture
- High Availability
- Upgrades
- HA/DR
- Troubleshooting
- Staff Augmentation
- Much more
- Conventional + retainer engagements

704-443-TROB (8762)   https://ioptconsulting.com

Why am I giving this session? If more shops had decent tools, we'd spend less time on fires and more on strategic tasks.

As tools go, MQSCX is so useful that I pay out of pocket for a full license, and I don't even have a paid MQ license.

Session Downloads

Session slides, scripts and supporting materials may be downloaded from https://t-rob.net/links

Native MQ scripting tools

- **Use runmqsc? There is no 1-line format option in runmqsc.**
  - Necessary to either parse multiple lines, or join lines to grep.
- **Use dmpmqcfg? There is no where clause in dmpmqcfg.**
  - Necessary to cast a wider net and do the filtering in the KSH script.
- **Neither tool has variable manipulation or program structures.**
  - All variables managed in KSH
  - All program control managed in KSH
  - Many calls to runmqsc & dmpmqcfg, many reconnections to MQ.

Of course, none of this seems like a restriction if runmqsc and dmpmqcfg are all you've ever worked with.

In fact, with the recent upgrades in functionality runmqsc and dmpmqcfg probably seem like they've been supercharged.

I apologize in advance for any bubbles that might be burst in that regard.

Sample scenario: Queue Depth Report

Objective seems simple enough:

Report of all XMitQs and Dead Queues with depth greater than zero.

Some assumptions:

- Newest runmqsc and dmpmqcfg
- CCDT file to manage connections
- Scripted on Linux using KSH.

These stipulations paint the IBM-supplied tools in the best possible light.

Report criteria

- **Dead queue**
  - Must be the one associated with the queue manager.
  - If no DLQ associated with the QMgr, then report that. (Negative confirmation!)
  - Report if depth is greater than zero.
- **Transmission queues**
  - Must be associated with a channel.
  - Report if depth is greater than zero.
- **Queue managers**
  - Iterate over multiple queue managers.
  - Provide negative acknowledgement (i.e. “Nothing to report”) per queue manager.

The general approach

Iterate over the queue managers running these steps on each one:

1. **Get the DEADQ( ) attribute from the QMgr.**
2. **Get the depth of the queue identified in Step 1.**
3. **Get the XMITQ( ) value for all outbound channels.**
4. **Get the CURDEPTH( ) for all queues where USAGE(XMITQ).**
5. **Report the set of queues within the intersection of Steps 3 & 4.**

Get the DLQ name/depth KSH style

```
DLQ=\$(echo dis qmgr deadq | runmqsc $QMGR 2>&1 | grep
'DEADQ(' | sed 's|.* DEADQ(||' | sed 's|).*||' | tr -d ' ')

if [ -z \${DLQ} ]; then
  print "QMgr $QMGR does not specify a Dead Letter Queue“
else
  DPTH=\$(echo "dis q(\$DLQ)" | runmqsc $QMGR 2>&1 | grep
'CURDEPTH(' | sed 's|.* CURDEPTH(||' | sed 's|).*||')
  print "\$DLQ CURDEPTH(\$DPTH)\n“
fi
```

1. Echo the command to runmqsc
2. Extract the line containing the attribute
3. Delete everything before the DEADQ attribute
4. Delete everything after the DEADQ attribute
5. Capture that to the env variable $DLQ
6. Check for existence of DLQ
7. Print depth

Get the DLQ name/depth MQSCX style

```
DIS QMGR DEADQ
if (DEADQ)
  @dlq=DEADQ
  DIS QL(<@dlq>) CURDEPTH
  print :n:@dlq,:n:"(",:n:CURDEPTH,:n:")"
else
  print "QMgr $QMGR does not specify a Dead Letter Queue"
endif
```

1. **Displaying the QMgr caches its attributes**
2. **Simple MQSCX test for presence of an attribute. DLQ defined?**
   1. Yes! Capture the DLQ name so we do not lose it in the next DIS command
   2. Print the DLQ name and depth
3. **Else**
   1. Print negative acknowledgement

Get depth of the XMitQs KSH style

```
echo "dis ql(*) where(USAGE eq XMITQ) curdepth CLCHNAME" | runmqsc $QMGR | sed -e :a -e
'$!N;s/\n   / /;ta' -e 'P;D' | sed 's|  *| |g' | grep ^AMQ | grep -v 'CURDEPTH(0)' | grep -v
'CURDEPTH( )' | sort | {
  while read XQDetails
  do
    XMitQ=\$(echo \$XQDetails | sed 's|.* QUEUE(\([^)]*\)).*|\1|')
    Curdepth=\$(echo \$XQDetails | sed 's|.* CURDEPTH(\([^)]*\)).*|\1|')
    ClChName=\$(echo \$XQDetails | sed 's|.* CLCHNAME(\([^)]*\)).*|\1|' | tr -d ' ')
    if [ -z \${ClChName} ]; then
      for Chl in \$(echo "dis chl(*) where(XMITQ eq \$XMitQ)" | runmqsc $QMGR | grep
'CHANNEL(' | sed 's|.* CHANNEL(\([^)]*\)).*|\1|')
      do
        print "Channel=\$Chl, XMitQ=\$XMitQ(\$Curdepth)"
      done
    else
      print "Channel=\$ClChName, XMitQ=\$XMitQ(\$Curdepth)"
    fi
  done
} | sort
```

1. Echo the command to runmqsc
2. Extract the line containing the attribute
3. Delete everything before the CURDEPTH attribute
4. Delete everything after the CURDEPTH attribute
5. Capture that to the env variable $DPTH
6. Extract values using sed
7. Check and print cluster chl XMitQ
8. Otherwise, look for channels using this XMitQ and print.

Get XMitQ names for outbound channels

```
foreach(DISPLAY QUEUE(*) CURDEPTH CLCHNAME =WHERE(USAGE EQ XMITQ AND CURDEPTH > 0))
  if (CLCHNAME)
    print "Channel=",:n:CLCHNAME,:n:", XMitQ=",:n:QUEUE,:n:"(",:n:CURDEPTH,:n:")"
  else
    @qname = QUEUE
    @depth = CURDEPTH
    foreach(DISPLAY CHL(*) WHERE(XMITQ EQ <@qname>))
      print "Channel=",:n:CHANNEL,:n:", XMitQ=",:n:@qname,:n:"(",:n:@depth,:n:")"
    endfor
  endif
endfor
```

1. Use DIS to get queue names and depths, use MQSCX compound =WHERE clause to filter just the Xmit queues with depth greater than zero.
2. Iterate through all the objects returned
3. Display those that have CLCHNAME (associated with cluster channels)
4. Otherwise, look up channels to see if one uses this XMitQ before printing.

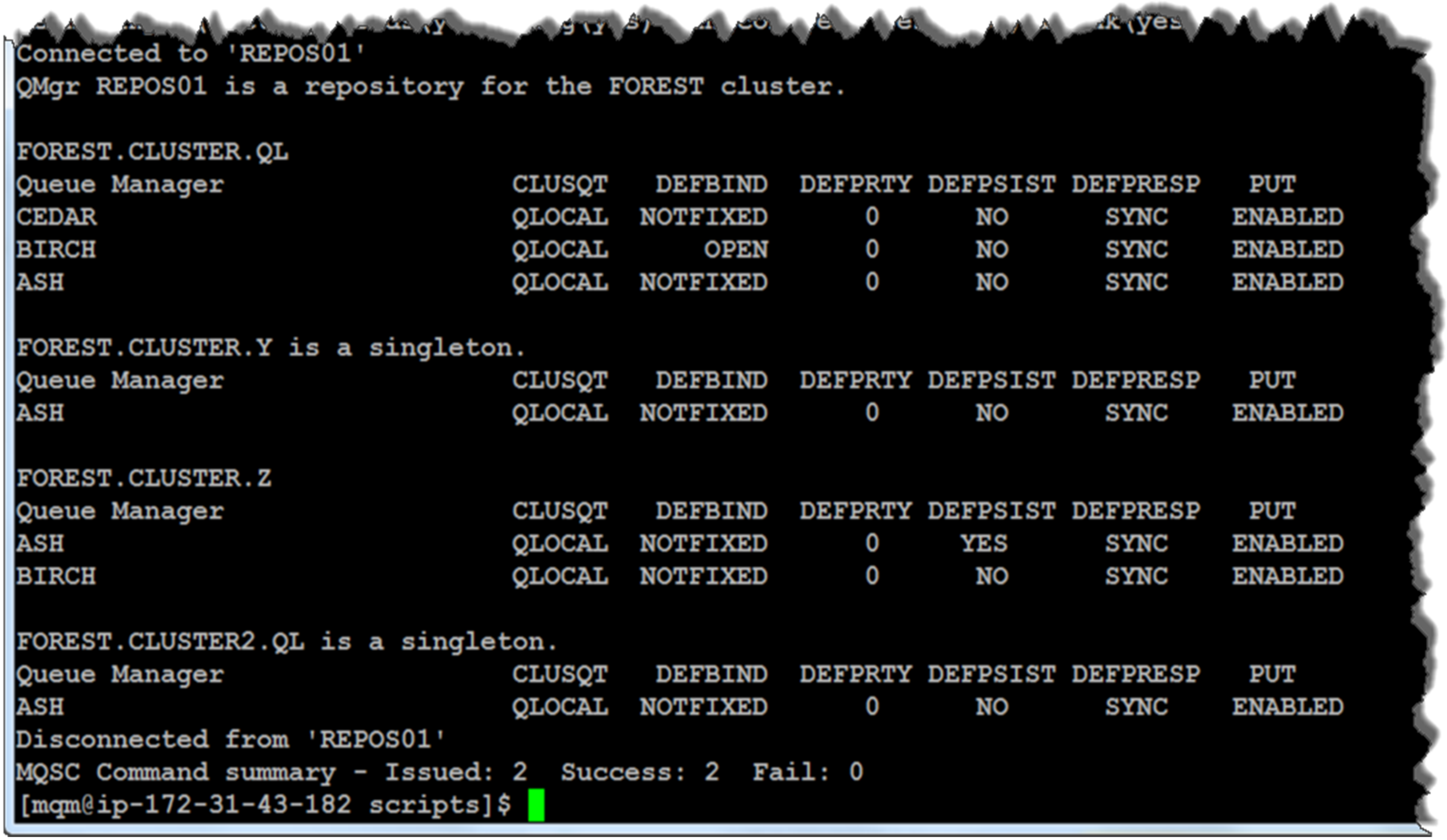
Decompress

That’s a lot of very dense text. There’s more like it on the way.
Take this moment to take a deep breath and chill out a bit.

Inconsistent cluster objects much?

- **Let’s find out!**
- **Sorry, no KSH equivalent. It’s too freakin’ much work. :-/**
- **Script runs on one full repository QMgr and reports any cluster objects…**
  - That are singletons
  - In which the cluster attributes are inconsistent
- **Despite all the cool stuff MQSCX does, the one thing I need that it does not do (yet) is defining subroutines or functions.**
  - Managing a local function in the same file is *much* easier.
  - Adds two lines to the code to dynamically create a callable subroutine.
  - When Paul saw how I did this, he said he will probably add function definition. Win!
  - For widely reusable modules, use the native MQSCX function to import external file.
- **Possible enhancement: Compare objects across two repositories**
  - Depends on how much feedback I get from this one. (Hint, hint!)

Cluster Check script



```
Connected to 'REPOS01'
QMgr REPOS01 is a repository for the FOREST cluster.

FOREST.CLUSTER.QL
Queue Manager                  CLUSQT   DEFBIND  DEFPRTY DEFPSIST DEFPRESP   PUT
CEDAR                          QLOCAL  NOTFIXED     0       NO      SYNC   ENABLED
BIRCH                          QLOCAL      OPEN     0       NO      SYNC   ENABLED
ASH                            QLOCAL  NOTFIXED     0       NO      SYNC   ENABLED

FOREST.CLUSTER.Y is a singleton.
Queue Manager                  CLUSQT   DEFBIND  DEFPRTY DEFPSIST DEFPRESP   PUT
ASH                            QLOCAL  NOTFIXED     0       NO      SYNC   ENABLED

FOREST.CLUSTER.Z
Queue Manager                  CLUSQT   DEFBIND  DEFPRTY DEFPSIST DEFPRESP   PUT
ASH                            QLOCAL  NOTFIXED     0      YES      SYNC   ENABLED
BIRCH                          QLOCAL  NOTFIXED     0       NO      SYNC   ENABLED

FOREST.CLUSTER2.QL is a singleton.
Queue Manager                  CLUSQT   DEFBIND  DEFPRTY DEFPSIST DEFPRESP   PUT
ASH                            QLOCAL  NOTFIXED     0       NO      SYNC   ENABLED
Disconnected from 'REPOS01'
MQSC Command summary - Issued: 2  Success: 2  Fail: 0
[mqm@ip-172-31-43-182 scripts]$
```

Cluster Check: Main program

```
foreach (DISPLAY QCLUSTER(*) cluster(*) CLUSQMGR CLUSQT DEFBIND DEFPRTY DEFPSIST DEFPRESP
PUT)
  if (@lastQueue != QUEUE AND @lastQueue != "")

    =import file($CtrlBreak) * <-- Pull in external Control Break routine

    * Reset control break fields
    @idx=0
    delvar(@queues)
  endif

  * Otherwise, save values for queue
  @idx=@idx+1
  @lastQueue = QUEUE
  @queues[@idx, 1] =  QUEUE
  @queues[@idx, 2] =  CLUSQMGR
  @queues[@idx, 3] =  CLUSQT
  @queues[@idx, 4] =  DEFBIND
  @queues[@idx, 5] =  DEFPRTY
  @queues[@idx, 6] =  DEFPSIST
  @queues[@idx, 7] =  DEFPRESP
  @queues[@idx, 8] =  PUT
endfor

* Final control break
=import file($CtrlBreak) * <-- Pull in external Control Break routine
```

The reason we needed a function was that the control break must be executed at least once outside the main processing loop. Either the first time to initialize it, then on every loop. Or else once each time the queue name changes and again at the end.

Cluster check: Control Break subroutine

```
CtrlBreak=$(mktemp $_Template) || exit 1
cat >> $CtrlBreak << EODCtrlBrk
if (@idx = 1)
  if (${_Singleton:=0} > 0)
    print _nl,@queues[1,1], "is a singleton."
    print :28:"Queue Manager","  CLUSQT   DEFBIND  DEFPRTY DEFPSIST DEFPRESP   PUT"
    print :28:@queues[1, 2], :8r:@queues[1, 3], :9r:@queues[1, 4], :6r:@queues[1, 5],
:7r:@queues[1, 6], :9r:@queues[1, 7], :10r:@queues[1, 8]
  endif
else
  * Check for discrepancies and print
  @flag = 0
  @x = 2
  while(@x <= @idx)
    if (exists(@queues[@x,1]))
      if (@queues[1, 3] != @queues[@x, 3] | @queues[1, 4] != @queues[@x, 4] |
@queues[1, 5] != @queues[@x, 5] | @queues[1, 6] != @queues[@x, 6] | @queues[1, 7] !=
@queues[@x, 7])
        @flag = 1
        @x = @idx
      endif
    endif
    @x = @x + 1
  endwhile
```

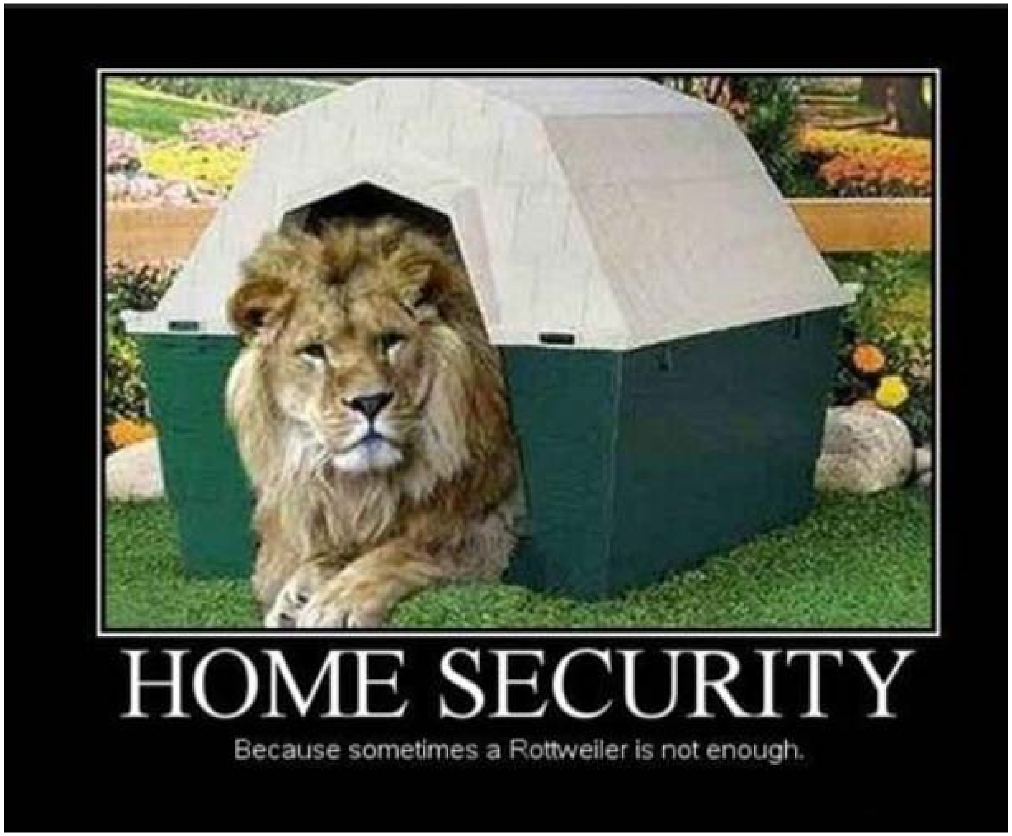
```
 if (@flag = 1)
    print _nl,@lastQueue
    print :28:"Queue Manager","  CLUSQT   DEFBIND  DEFPRTY DEFPSIST DEFPRESP   PUT"
    @x = 1
    while(@x <= @idx)
      if (exists(@queues[@x, 1]))
        print :28:@queues[@x, 2], :8r:@queues[@x, 3], :9r:@queues[@x, 4],
:6r:@queues[@x, 5], :7r:@queues[@x, 6], :9r:@queues[@x, 7], :10r:@queues[@x, 8]
      endif
      @x = @x + 1
    endwhile
  endif
endif
EODCtrlBrk
```

Lines in RED are added to make subroutines work.

- **Counts number of object instances for a given object.**
- **Reports all singletons. (Because if there’s only one instance, we don’t necessarily need it clustered to address it.)**
- **Otherwise compares attributes. If any differ, then print the object.**

Relax. We've got your back.

That code was even more dense than the first script.
As before, there's more like it on the way.
Take another deep breath and reset your brain.



Automating CCDT files

CCDT Files are indexed by channel name.
The implications of this include:

- **All CLNTCONN channels in a CCDT must be uniquely named so we will need to exclude channels with dupe names**
  - Particularly the default SVRCONN channels, but any others as well.
  - Not a limitation of the script. If you are using dupe channel names, they aren't in the same CCDT now and this won't change that.

- **Including App HLQ, SDLC level (Prod, QA, etc.) and/or QMgr in the channel name leads to unique names that are also useful filters.**

- **The CONNAME for a CLNTCONN cannot be derived from the matching SVRCONN. Therefore, we need a source of CONNAME values. The example uses a simple txt file. Typically you would want to use MQ_Handles.xml from Explorer, a master CCDT, an IR-360 service, etc.**

Saving QMgrs

The CCDT script scans the backups of all the QMgrs to get channel names. That assumes a particular format and tree structure based on this snippet:

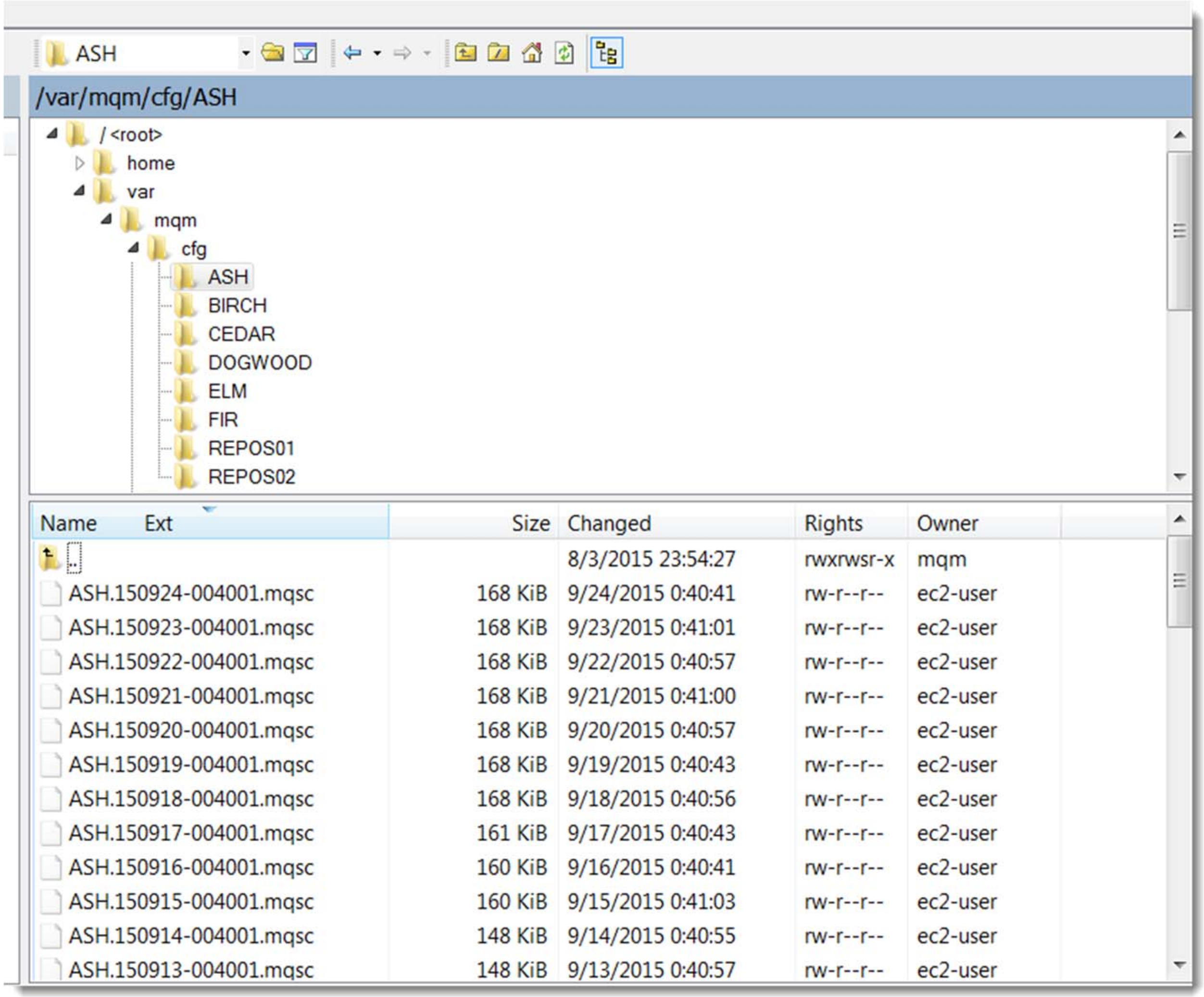
```
while read MyQMGR;do
  echo Processing \$MyQMGR at `date`
  mkdir -p /var/mqm/cfg/\$MyQMGR/ > /dev/null 2>&1
  /opt/mqm/bin/dmpmqcfg -c default -m \$MyQMGR -a -o 2line \
       > /var/mqm/cfg/\$MyQMGR/\$MyQMGR.$_DATE.mqsc 2>&1
done
```

Saving QMgrs

…to produce this

directory and

file structure:



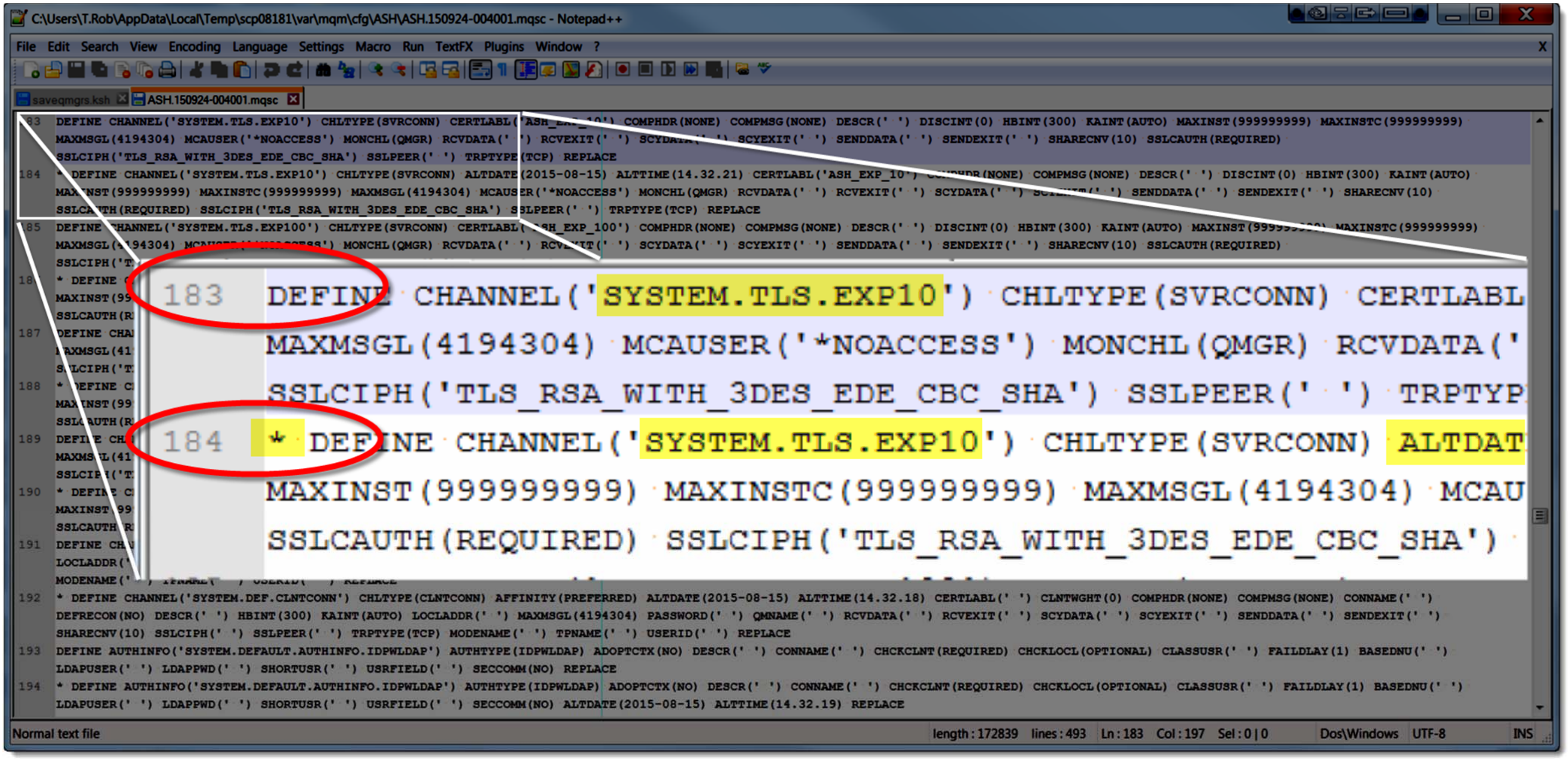
/var/mqm/cfg/ASH

- / <root>
  - home
  - var
    - mqm
      - cfg
        - ASH
        - BIRCH
        - CEDAR
        - DOGWOOD
        - ELM
        - FIR
        - REPOS01
        - REPOS02

Name	Ext	Size	Changed	Rights	Owner
..			8/3/2015 23:54:27	rwxrwsr-x	mqm
ASH.150924-004001.mqsc		168 KiB	9/24/2015 0:40:41	rw-r--r--	ec2-user
ASH.150923-004001.mqsc		168 KiB	9/23/2015 0:41:01	rw-r--r--	ec2-user
ASH.150922-004001.mqsc		168 KiB	9/22/2015 0:40:57	rw-r--r--	ec2-user
ASH.150921-004001.mqsc		168 KiB	9/21/2015 0:41:00	rw-r--r--	ec2-user
ASH.150920-004001.mqsc		168 KiB	9/20/2015 0:40:57	rw-r--r--	ec2-user
ASH.150919-004001.mqsc		168 KiB	9/19/2015 0:40:43	rw-r--r--	ec2-user
ASH.150918-004001.mqsc		168 KiB	9/18/2015 0:40:56	rw-r--r--	ec2-user
ASH.150917-004001.mqsc		161 KiB	9/17/2015 0:40:43	rw-r--r--	ec2-user
ASH.150916-004001.mqsc		160 KiB	9/16/2015 0:40:41	rw-r--r--	ec2-user
ASH.150915-004001.mqsc		160 KiB	9/15/2015 0:41:03	rw-r--r--	ec2-user
ASH.150914-004001.mqsc		148 KiB	9/14/2015 0:40:55	rw-r--r--	ec2-user
ASH.150913-004001.mqsc		148 KiB	9/13/2015 0:40:57	rw-r--r--	ec2-user

Saving QMgrs

…and files where every object is on one line for easy grepping.



```
DEFINE CHANNEL('SYSTEM.TLS.EXP10') CHLTYPE(SVRCONN) CERTLABL('ASH_EXP_10') COMPHDR(NONE) COMPMSG(NONE) DESCR(' ') DISCINT(0) HBINT(300) KAINT(AUTO) MAXINST(999999999) MAXINSTC(999999999)
MAXMSGL(4194304) MCAUSER('*NOACCESS') MONCHL(QMGR) RCVDATA(' ') RCVEXIT(' ') SCYDATA(' ') SCYEXIT(' ') SENDDATA(' ') SENDEXIT(' ') SHARECNV(10) SSLCAUTH(REQUIRED)
SSLCIPH('TLS_RSA_WITH_3DES_EDE_CBC_SHA') SSLPEER(' ') TRPTYPE(TCP) REPLACE
* DEFINE CHANNEL('SYSTEM.TLS.EXP10') CHLTYPE(SVRCONN) ALTDATE(2015-08-15) ALTTIME(14.32.21) CERTLABL('ASH_EXP_10') COMPHDR(NONE) COMPMSG(NONE) DESCR(' ') DISCINT(0) HBINT(300) KAINT(AUTO)
MAXINST(999999999) MAXINSTC(999999999) MAXMSGL(4194304) MCAUSER('*NOACCESS') MONCHL(QMGR) RCVDATA(' ') RCVEXIT(' ') SCYDATA(' ') SCYEXIT(' ') SENDDATA(' ') SENDEXIT(' ') SHARECNV(10)
SSLCAUTH(REQUIRED) SSLCIPH('TLS_RSA_WITH_3DES_EDE_CBC_SHA') SSLPEER(' ') TRPTYPE(TCP) REPLACE
DEFINE CHANNEL('SYSTEM.TLS.EXP100') CHLTYPE(SVRCONN) CERTLABL('ASH_EXP_100') COMPHDR(NONE) COMPMSG(NONE) DESCR(' ') DISCINT(0) HBINT(300) KAINT(AUTO) MAXINST(999999999) MAXINSTC(999999999)
MAXMSGL(4194304) MCAUSER('*NOACCESS') MONCHL(QMGR) RCVDATA(' ') RCVEXIT(' ') SCYDATA(' ') SCYEXIT(' ') SENDDATA(' ') SENDEXIT(' ') SHARECNV(10) SSLCAUTH(REQUIRED)
SSLCIPH('T
```

```
183  DEFINE CHANNEL('SYSTEM.TLS.EXP10') CHLTYPE(SVRCONN) CERTLABL
     MAXMSGL(4194304) MCAUSER('*NOACCESS') MONCHL(QMGR) RCVDATA('
     SSLCIPH('TLS_RSA_WITH_3DES_EDE_CBC_SHA') SSLPEER(' ') TRPTYP
184  * DEFINE CHANNEL('SYSTEM.TLS.EXP10') CHLTYPE(SVRCONN) ALTDAT
     MAXINST(999999999) MAXINSTC(999999999) MAXMSGL(4194304) MCAU
     SSLCAUTH(REQUIRED) SSLCIPH('TLS_RSA_WITH_3DES_EDE_CBC_SHA')
```

```
MODENAME(' ') TPNAME(' ') USERID(' ') REPLACE
* DEFINE CHANNEL('SYSTEM.DEF.CLNTCONN') CHLTYPE(CLNTCONN) AFFINITY(PREFERRED) ALTDATE(2015-08-15) ALTTIME(14.32.18) CERTLABL(' ') CLNTWGHT(0) COMPHDR(NONE) COMPMSG(NONE) CONNAME(' ')
DEFRECON(NO) DESCR(' ') HBINT(300) KAINT(AUTO) LOCLADDR(' ') MAXMSGL(4194304) PASSWORD(' ') QMNAME(' ') RCVDATA(' ') RCVEXIT(' ') SCYDATA(' ') SCYEXIT(' ') SENDDATA(' ') SENDEXIT(' ')
SHARECNV(10) SSLCIPH(' ') SSLPEER(' ') TRPTYPE(TCP) MODENAME(' ') TPNAME(' ') USERID(' ') REPLACE
DEFINE AUTHINFO('SYSTEM.DEFAULT.AUTHINFO.IDPWLDAP') AUTHTYPE(IDPWLDAP) ADOPTCTX(NO) DESCR(' ') CONNAME(' ') CHCKCLNT(REQUIRED) CHCKLOCL(OPTIONAL) CLASSUSR(' ') FAILDLAY(1) BASEDNU(' ')
LDAPUSER(' ') LDAPPWD(' ') SHORTUSR(' ') USRFIELD(' ') SECCOMM(NO) REPLACE
* DEFINE AUTHINFO('SYSTEM.DEFAULT.AUTHINFO.IDPWLDAP') AUTHTYPE(IDPWLDAP) ADOPTCTX(NO) DESCR(' ') CONNAME(' ') CHCKCLNT(REQUIRED) CHCKLOCL(OPTIONAL) CLASSUSR(' ') FAILDLAY(1) BASEDNU(' ')
LDAPUSER(' ') LDAPPWD(' ') SHORTUSR(' ') USRFIELD(' ') SECCOMM(NO) ALTDATE(2015-08-15) ALTTIME(14.32.19) REPLACE
```

The actual script

- **Same script prologue as before to set up logging and global variables.**
- **When parsing backup files, we do not ones with** **`AMQ9508: Program cannot connect to the queue manager.`**
- **And we don’t want to use QMgrs that haven’t been backed up for a while.**
  - Arbitrarily picked 10 days at the cutoff point. YMMV.

```
# Get the epoch time from 10 days ago
((Oldest=\$( date "+%s") - (10*24*60*60) ))
```

Check backup dir against connection list

```
ls -1 ${_BkupRoot} | {
  while read QMgr;do
    CONNAME=\$(grep "^\$QMgr/" ./connections.txt | wc -l | tr -d " ")
    if [[ \$CONNAME -lt 1 ]]; then
      print "\$QMgr: No connection entry found.  Skipping.\n"
      break
    fi
    print "\$QMgr: Connection entry found.  Checking for recent
backups."
    Candidates=0
    GotOne=0
```

1. Backup root contains one directory per QMgr. List them.
2. Make sure each directory has a connection entry.
3. Candidates will be our count of backup files newer than 10 days before aborting or continuing.
4. The Got One flag is set if we find a useable backup file.

Now process the QMgr’s backup diredtory

```
 ls -1r ${_BkupRoot}/\$QMgr/*.??????-??????.mqsc | {
      while read FileName;do
        BaseName=\$(basename \$FileName)
        YYMMDD=\${BaseName#*.}  # Strip QMgr name from front of string
        Age=\$(printf "%(%#)T" "\${YYMMDD:2:2}/\${YYMMDD:4:2}/\${YYMMDD:0:2}")
        [[ \$Age -lt \$Oldest ]] && break
        ((Candidates+=1))
        echo \$FileName
        GotOne=\$(grep -v '^*' \$FileName | grep 'CHLTYPE(SVRCONN' | wc -l | tr
-d " ")
        [[ \$GotOne > 0 ]] && break
      done
```

1. **Use a template to filter on backup files only.**
2. **The file name contains a timestamp. Convert that to it’s age.**
3. **The files are sorted by descending age so break when we find one too old.**
4. **Otherwise, we have an eligible candidate.**
5. **Grep to see if it has any SVRCONN channel definitions.**
6. **If so, set the GotOne flag and stop looking at backup files. We only need the first good one.**

Save any SVRCONNs we find

```
 if [[ \$Candidates > 0 && \$GotOne == 0 ]]; then
        print "No SVRCONN channels found among most recent \$Candidates
backups. Skipping.\n"
      else
        # Save off the channels so we can check for dup names before making
CCDT
        grep -v '^*' \$FileName | grep 'CHLTYPE(SVRCONN' | {
          while read Channel;do
            # print "\t\$Cursor - \$Channel"
            Channels[\$Cursor]="\$Channel QMNAME(\$QMgr)"
            ((Cursor+=1))
          done
        }
      fi
```

1. If we get here and didn’t find a backup, notify the user in the log.
2. Otherwise, fetch all SVRCONN definitions in the file and save the relevant object attributes to an array.

Sort the array

- **Rather than sort the KSH array ourselves, just print it and pipe to sort**

```
i=-1
LastChannel=""
Dupe=0
{
  while [[ i -lt \$Cursor ]]; do
    ((i+=1))
    print \${Channels[\$i]}
  done
} | sort | {
```

Undupe and pipe to MQSCX

```
 while read Channel;do
    ThisChl=\$(echo \${Channel} | sed "s/.* CHANNEL('\([^']*\).*/\1/")
    QMNAME=\$(echo \${Channel} | sed "s/.* QMNAME(\([^)]*\).*/\1/")
    SSLCIPH=\$(echo \${Channel} | sed "s/.* SSLCIPH('\([^']*\).*/\1/")
    CONNAME=\$(grep "^\$QMNAME/" ./connections.txt | cut -d / -f 3)
    if [[ \$ThisChl == \$LastChannel ]];then
      ((Dupe+=1))
    else
      [[ \$Dupe -eq 0 ]] && print "DEFINE CHL(\$ThisChl) CHLTYPE(CLNTCONN)
CONNAME('\$CONNAME') QMNAME(\$QMNAME) SSLCIPH('\$SSLCIPH')"
      LastChannel=\$ThisChl
      Dupe=0
    fi
  done
} | mqscx -n -t ./AMQCLCHL.TAB
```

1. Any time the channel name changes, check if we’ve seen it before.
2. If so, it’s a dupe so discard.
3. If not, format it as a DEF CHL statement and pipe it to MQSCX.

Possible CCDT Script enhancements

It’s tough to anticipate what the naming convention will be at a given shop so the proposed enhancements are discussed in general terms.

- **Filter for Dev, SIT, QA, Prod, etc. This can be by naming convention in the QMgrs or in the channels, or it can use physical grouping in the connections database/file.**
- **Use application HLQ in the channel name to filter specific applications.**
- **Add SSL, UserID or other custom data to the CCDT. The more these elements can be standardized, the easier it is to automate them.**

DOWNLOAD IT!

All scripts and supporting files from this session are available NOW at

https://t-rob.net/links

…along with lots of other Good Stuff!

(Also, full program listings below.)

Questions & Answers

KSH Script to get depth of DLQ and XMitQs

```
#!/usr/bin/ksh
cd "$(cd "$(dirname "$0")"; pwd)"
#------------------------------------------------------------------------------
# QDepth.ksh
#
# Script to get depth of any transmit queues and the DLQ
#
#
# 20150909 T.Rob  New script
#------------------------------------------------------------------------------
# Fetch vars to construct log file name
_PROG=${0##*/} && QMGR=${1}
_DATE=`date "+%y%m%d-%H%M%S"`
_LOG=$_PROG.$QMGR.$_DATE.out

# Find the mqm/bin directory
_MQMBIN="$(dspmqver -f 128 -b)/bin"

# Execute a new shell with redirection of the log file
/usr/bin/ksh > $_LOG 2>&1 << EOF

# Make sure the QMgr is there and running before continuing
[[ \$(${_MQMBIN}/dspmq -m $QMGR | grep '(Running)' | wc -l | tr -d " ") != 1 ]] && print
"\n${0##*/}: Queue Manager '$QMGR' not found or not running.\n" && exit 1

DLQ=\$(echo dis qmgr deadq | runmqsc $QMGR 2>&1 | grep 'DEADQ(' | sed 's|.* DEADQ(||' | sed
's|).*||' | tr -d ' ')
if [ -z \${DLQ} ]; then
  print "QMgr $QMGR does not specify a Dead Letter Queue"
else
  DPTH=\$(echo "dis q(\$DLQ)" | runmqsc $QMGR 2>&1 | grep 'CURDEPTH(' | sed 's|.*
CURDEPTH(||' | sed 's|).*||')
  print "\$DLQ CURDEPTH(\$DPTH)\n"
fi

echo "dis ql(*) where(USAGE eq XMITQ) curdepth CLCHNAME" | runmqsc $QMGR | sed -e :a -e
'$!N;s/\n   / /;ta' -e 'P;D' | sed 's|  *| |g' | grep ^AMQ | grep -v 'CURDEPTH(0)' | grep -v
'CURDEPTH( )' | sort | {
  while read XQDetails
  do
    XMitQ=\$(echo \$XQDetails | sed 's|.* QUEUE(\([^)]*\)).*|\1|')
    Curdepth=\$(echo \$XQDetails | sed 's|.* CURDEPTH(\([^)]*\)).*|\1|')
    ClChName=\$(echo \$XQDetails | sed 's|.* CLCHNAME(\([^)]*\)).*|\1|' | tr -d ' ')
    if [ -z \${ClChName} ]; then
      for Chl in \$(echo "dis chl(*) where(XMITQ eq \$XMitQ)" | runmqsc $QMGR | grep
'CHANNEL(' | sed 's|.* CHANNEL(\([^)]*\)).*|\1|')
      do
        print "Channel=\$Chl, XMitQ=\$XMitQ(\$Curdepth)"
      done
    else
      print "Channel=\$ClChName, XMitQ=\$XMitQ(\$Curdepth)"
    fi
  done
} | sort
#-----------------------------------------------------------------
#                  E N D   O F   S C R I P T
#-----------------------------------------------------------------
EOF
cat $_LOG
```

MQSCX equivalent of Qdepth.ksh

```
#!/usr/bin/ksh
cd "$(cd "$(dirname "$0")"; pwd)"
#-------------------------------------------------------------
# QDepth.mqscx
#
# Script to get depth of any transmit queues and the DLQ
#
#
# 20150909 T.Rob  New script
#-------------------------------------------------------------
# Fetch vars to construct log file name
_PROG=${0##*/} && QMGR=${1}
_DATE=`date "+%y%m%d-%H%M%S"`
_LOG=$_PROG.$QMGR.$_DATE.out
# Find the mqm/bin directory
_MQMBIN="$(dspmqver -f 128 -b)/bin"

# Execute a new shell with redirection of the log file
/usr/bin/ksh > $_LOG 2>&1 << EOF

# Make sure the QMgr is there and running before continuing
[[ \$(${_MQMBIN}/dspmq -m $QMGR | grep '(Running)' | wc -l | tr -d " ") != 1 ]] && print
"\n${0##*/}: Queue Manager '$QMGR' not found or not running.\n" && exit 1

mqscx -w0 -m $QMGR << EOMQSCX
=echo input(no) cmds(no) lang(no) langcon(no) resp(no) blank(no)
* =echo input(setting) cmds(yes) lang(yes) langcon(yes) resp(yes) blank(yes)
```

```
DIS QMGR DEADQ
if (DEADQ)
  @dlq=DEADQ
  DIS QL(<@dlq>) CURDEPTH
  print :n:@dlq,:n:"(",:n:CURDEPTH,:n:")"
else
  print "QMgr $QMGR does not specify a Dead Letter Queue"
endif

* Get all the transmission queues with CURDEPTH > 0
foreach(DISPLAY QUEUE(*) CURDEPTH CLCHNAME =WHERE(USAGE EQ XMITQ AND CURDEPTH > 0))
  if (CLCHNAME)
    print "Channel=",:n:CLCHNAME,:n:", XMitQ=",:n:QUEUE,:n:"(",:n:CURDEPTH,:n:")"
  else
    @qname = QUEUE
    @depth = CURDEPTH
    foreach(DISPLAY CHL(*) WHERE(XMITQ EQ <@qname>))
      print "Channel=",:n:CHANNEL,:n:", XMitQ=",:n:@qname,:n:"(",:n:@depth,:n:")"
    endfor
  endif
endfor

EOMQSCX

#-----------------------------------------------------------------
#                 E N D   O F   S C R I P T
#-----------------------------------------------------------------
EOF
cat $_LOG
```

Cluster Check

```
#!/usr/bin/ksh
cd "$(cd "$(dirname "$0")"; pwd)"
#------------------------------------------------------------------------------
# clqck.mqscx
#
# Script to check the consistency of queues in a cluster
#
#
# 20150909 T.Rob  New script
#------------------------------------------------------------------------------
# Fetch vars to construct log file name
_PROG=${0##*/} && QMGR=${1}
_DATE=`date "+%y%m%d-%H%M%S"`
_LOG=$_PROG.$QMGR.$_DATE.out
_Template=$0.ctrlbrk.XXXXXXXXXX
# Find the mqm/bin directory
_MQMBIN="$(dspmqver -f 128 -b)/bin"

# Option to display singletons
[[ $# > "1" ]] &&  _Singleton=1
#print "\${#}=${#}, \$_Singleton=$_Singleton\n"
```

```
#------------------------------------------------------------------------------
# C O N T R O L   B R E A K   S U B R O U T I N E
#------------------------------------------------------------------------------
CtrlBreak=$(mktemp $_Template) || exit 1
cat >> $CtrlBreak << EODCtrlBrk
if (@idx = 1)
  if (${_Singleton:=0} > 0)
    print _nl,@queues[1,1], "is a singleton."
    print :28:"Queue Manager","  CLUSQT   DEFBIND  DEFPRTY DEFPSIST DEFPRESP   PUT"
    print :28:@queues[1, 2], :8r:@queues[1, 3], :9r:@queues[1, 4], :6r:@queues[1, 5],
:7r:@queues[1, 6], :9r:@queues[1, 7], :10r:@queues[1, 8]
  endif
else
  * Check for discrepancies and print
  @flag = 0
  @x = 2
  while(@x <= @idx)
    if (exists(@queues[@x,1]))
      if (@queues[1, 3] != @queues[@x, 3] | @queues[1, 4] != @queues[@x, 4] | @queues[1, 5]
!= @queues[@x, 5] | @queues[1, 6] != @queues[@x, 6] | @queues[1, 7] != @queues[@x, 7])
        @flag = 1
        @x = @idx
      endif
    endif
    @x = @x + 1
  endwhile
```

```
 if (@flag = 1)
    print _nl,@lastQueue
    print :28:"Queue Manager","  CLUSQT   DEFBIND  DEFPRTY DEFPSIST DEFPRESP   PUT"
    @x = 1
    while(@x <= @idx)
      if (exists(@queues[@x, 1]))
        print :28:@queues[@x, 2], :8r:@queues[@x, 3], :9r:@queues[@x, 4], :6r:@queues[@x,
5], :7r:@queues[@x, 6], :9r:@queues[@x, 7], :10r:@queues[@x, 8]
      endif
      @x = @x + 1
    endwhile
  endif
endif
EODCtrlBrk

#------------------------------------------------------------------------------
# M A I N   P R O G R A M
#------------------------------------------------------------------------------

# Execute a new shell with redirection of the log file
/usr/bin/ksh > $_LOG 2>&1 << EOF

# Make sure the QMgr is there and running before continuing
[[ \$(${_MQMBIN}/dspmq -m $QMGR | grep '(Running)' | wc -l | tr -d " ") != 1 ]] && print
"\n${0##*/}: Queue Manager '$QMGR' not found or not running.\n" && exit 1
```

```
mqscx -w0 -m $QMGR 2>&1 << EOMQSCX
* mqscx -Dt -w0 -m $QMGR 2>&1 << EOMQSCX
=echo input(setting) cmds(yes) lang(yes) langcon(yes) resp(yes) blank(yes)
=echo input(no) cmds(no) lang(no) langcon(no) resp(no) blank(no)

DIS QMGR REPOS
if (REPOS)
  @clusname=REPOS
  print "QMgr $QMGR is a repository for the",@clusname,"cluster."
else
  print "QMgr $QMGR does not appear to be a repository."
  end
endif

@idx = 0
@lastQueue = "“

foreach (DISPLAY QCLUSTER(*) cluster(*) CLUSQMGR CLUSQT DEFBIND DEFPRTY DEFPSIST DEFPRESP
PUT)
  if (@lastQueue != QUEUE AND @lastQueue != "")

    =import file($CtrlBreak)

    * Reset control break fields
    @idx=0
    delvar(@queues)
  endif
  * Otherwise, save values for queue
   @idx=@idx+1
   @lastQueue = QUEUE
   @queues[@idx, 1] =  QUEUE
   @queues[@idx, 2] =  CLUSQMGR
   @queues[@idx, 3] =  CLUSQT
   @queues[@idx, 4] =  DEFBIND
   @queues[@idx, 5] =  DEFPRTY
   @queues[@idx, 6] =  DEFPSIST
   @queues[@idx, 7] =  DEFPRESP
   @queues[@idx, 8] =  PUT
 endfor

 * Final control break
 =import file($CtrlBreak)

 EOMQSCX

 #------------------------------------------------------------------
 #                  E N D   O F   S C R I P T
 #------------------------------------------------------------------
 rm -f $CtrlBreak
 EOF
 cat $_LOG
```

CCDT Automation

```
#!/bin/ksh
cd "$(cd "$(dirname "$0")"; pwd)"
# -------------------------------------------------------------------
# ccdt.ksh
#
# Builds a CCDT file from QMgr object backups
#
# -------------------------------------------------------------------

# Where & when
_BkupRoot=/var/mqm/cfg

# Fetch vars to construct log file name
_PROG=${0##*/} && _QMgr=${1}
_DATE=`date "+%y%m%d-%H%M%S"`
_LOG=/var/mqm/log/$_PROG.$_DATE.out

# Execute a new shell with redirection of the log file
/bin/ksh -s > $_LOG 2>&1 <<EOFEXEC

export MQCERTLABL=ec2-user
export MQCHLLIB=/home/ec2-user
export MQSSLKEYR=/home/ec2-user/.ssh/key
```

```
print "
#-----------------------------------------------------------------
#      Start scanning queue manager configs at `date`
#-----------------------------------------------------------------
"
# Get the epoch time from 10 days ago
((Oldest=\$( date "+%s") - (10*24*60*60) ))

# Collect the matching SVRCONN channel defs
Cursor=0
ls -1 ${_BkupRoot} | {
  while read QMgr;do
    CONNAME=\$(grep "^\$QMgr/" ./connections.txt | wc -l | tr -d " ")
    if [[ \$CONNAME -lt 1 ]]; then
      print "\$QMgr: No connection entry found.  Skipping.\n"
      break
    fi
    print "\$QMgr: Connection entry found.  Checking for recent backups."
    Candidates=0
    GotOne=0
 ls -1r ${_BkupRoot}/\$QMgr/*.??????-??????.mqsc | {
      while read FileName;do
        BaseName=\$(basename \$FileName)
        YYMMDD=\${BaseName#*.}  # Strip QMgr name from front of string
        Age=\$(printf "%(%#)T" "\${YYMMDD:2:2}/\${YYMMDD:4:2}/\${YYMMDD:0:2}")
        [[ \$Age -lt \$Oldest ]] && break
        ((Candidates+=1))
        echo \$FileName
        GotOne=\$(grep -v '^*' \$FileName | grep 'CHLTYPE(SVRCONN' | wc -l | tr -d " ")
        [[ \$GotOne > 0 ]] && break
      done
 if [[ \$Candidates > 0 && \$GotOne == 0 ]]; then
        print "No SVRCONN channels found among most recent \$Candidates backups.
Skipping.\n"
      else
        # Save off the channels so we can check for dup names before making CCDT
        grep -v '^*' \$FileName | grep 'CHLTYPE(SVRCONN' | {
          while read Channel;do
            # print "\t\$Cursor - \$Channel"
            Channels[\$Cursor]="\$Channel QMNAME(\$QMgr)"
            ((Cursor+=1))
          done
        }
      fi
    }
    echo
  done
}

#set -v
i=-1
LastChannel="“
Dupe=0
{
  while [[ i -lt \$Cursor ]]; do
    ((i+=1))
    print \${Channels[\$i]}
  done
} | sort | {
 while read Channel;do
    ThisChl=\$(echo \${Channel} | sed "s/.* CHANNEL('\([^']*\).*/\1/")
    QMNAME=\$(echo \${Channel} | sed "s/.* QMNAME(\([^)]*\).*/\1/")
    SSLCIPH=\$(echo \${Channel} | sed "s/.* SSLCIPH('\([^']*\).*/\1/")
    CONNAME=\$(grep "^\$QMNAME/" ./connections.txt | cut -d / -f 3)
    if [[ \$ThisChl == \$LastChannel ]];then
      ((Dupe+=1))
    else
      [[ \$Dupe -eq 0 ]] && print "DEFINE CHL(\$ThisChl) CHLTYPE(CLNTCONN)
CONNAME('\$CONNAME') QMNAME(\$QMNAME) SSLCIPH('\$SSLCIPH')"
      LastChannel=\$ThisChl
      Dupe=0
    fi
  done
} | mqscx -n -t ./AMQCLCHL.TAB

print "CCDT processing complete at \$(date)

#-----------------------------------------------------------------
#                  E N D   O F   S C R I P T
#-----------------------------------------------------------------
"
EOFEXEC
cat $_LOG
```