Websphere Message Queueing System

MQ Content

Mod 1: Websphere MQ Intro
- What is message queue
- What is a message and queue
- Type of messages
- Synchronous Vs Asynchronous

Mod 2: Installation & Uninstallation Linux, AIX and windows
- Installation Overview
- Create users and Group
- Product Directory Structure
- Disk Space Requirement
- Installing Server
- Support Pass
- Uninstallation Svr
- Migration & Upgrade

Mod 3: Administration
- Creating, Start, Stop Deleting, Displaying Queue Manager
- Queue Manager Properties
- Automatic Start

Mod 4: Queue Manager objects and their properties
- Queue Types
    - Local, Remote, Transmission, Alias, Model
- Channel Type
    - Sender, Receiver, Cluster Sender & Receiver
    - Server Connection

Mod 5: Distribution Setup (Point to Point Setup)
- Why Distribution Setup
- Advantages with distribution setup
- Application Communication in IBM MQ with distribution setup - Technical Implementation

Mod 6: Triggering
- Intro
- Channel Triggering
- Initiation Queue
- Trigger events & Trigger conditions
- Application Triggering
- Trigger Monitor

- Process

Mod 7: Reason Code & Dead letter queue
- MQ Reason Codes
- Dead Letter queue & it’s advantage?
- Troubleshoot message in dead letter queue.

Mod 8: Security
- Security Objective
- Achieving Security Objective
- General Security Principals
- Channel Security with SSL
- Channel Authority Records

Mod 9: Logging, Troubleshooting, Backup and Restore
- Types of logging
- Logging Configuration
- Log Sizing
- Log Archiving
- Troubleshooting Steps
- Backing up MQ Configuration & Data
- Checkpoint Recovery

Mod 10: Clustering
- Adv of Clustering
- Cluster Repository
- Best Practices
- Cluster workload Balancing
- Sending message to a cluster Queue
- Sending Message to a cluster Queue
- Remove Cluster Queue Manager From Cluster
- High Availability Cluster

Mod 11: Client-Server Architecture
- MQ client and server
- Adv with client mode connectivity
- Client Connectivity to MQ Server

What is IBM WebSphere MQ?

IBM WebSphere MQ is messaging for applications. It sends messages across networks of diverse components. Your application connects to IBM WebSphere MQ to send or receive a message. IBM WebSphere MQ handles the different processors, operating systems, subsystems, and communication protocols it encounters in transferring messages. If a connection or a processor is temporarily unavailable, IBM WebSphere MQ queues the message and forwards it when the connection is back online.

An application has a choice of programming interfaces, and programming languages to connect to IBM WebSphere MQ.

IBM WebSphere MQ is messaging and queuing middleware, with point-to-point, publish/subscribe, and file transfer modes of operation. Applications can publish messages to many subscribers over multicast.

Installation of MQ Server can be on any OS platform.

Control Commands

```
$ dspmq -o installation -o all        << display the installation path of MQ server

$ dspmqver                            << display MQ version. Currently it is 9.2.
$ dspmq                               << display all Queue Managers

$ crtmqm <QMN>                        << create Queue Manager with a specific name.
        -q = default Queue Manager
        -ll = Liner Logging     (continuous writing restart recovery and media recovery)
        -lc = circular Logging  (As a default, restart recovery only)
        -md = Directory used to hold the log files for QM

$ dspmq -m <QMN>                      << display Queue Manager status with QM Name.
$ strmqm <QMN>                        << start Queue Manager
$ endmqm <QMN>                        << stop Queue Manager
$ amqsget <QN> <QM>                   << Consume message in queues
$ amqsput <QN> <QM>                   << send message to queue
$ amqsbcg <QN> <QM>                   << browse message from queue
$ dltmqm <QMN>                        << to delete queue Manager
```

We will discuss control commands in detail later for further usage. The above commands will be the basic steps of this document.

Queue Manager’s Objects (Simple Queues)

- **Local Queue:** Transmission, initiation, dead-letter, command, default, channel, and event queues are types of local queues.

- **Alias Queue:** The alias queue is usually used as a pointer of the local queue. And also can be used as a pointer to a topic of publish/subscriber applications.
- **Model Queue:** A model queue is a template that can be used by an application to dynamically create a real queue. These templates are often used to create a unique queue for reply messages and then the queue is automatically deleted when the application ends.
- **Remote Queue:** To a program, a queue is remote if it is owned by a different queue manager than the one to which the program is connected. This queue has the address of the destination queue to send the messages.
- **Dynamic Queue:** These queues are created by the process of opening a model queue. They are real local queues, but they cannot be explicitly created through administration interfaces.

Special Queues

- **Dead Letter Queue:** same as a backout queue. It is just a local Q, Only one dead letter Q for Queue Manager.

- **Transmission Queue:** A transmission queue is a local queue with the USAGE(XMITQ) attributes configured. Typically, there is **one transmission queue for each remote queue manager** to which the local queue manager might connect directly.

- **Initiation queue:** An initiation queue is a local queue to which **the queue manager writes a trigger message** when certain conditions are met on another local queue.

- **Event Queue:** The queue manager generates event messages when certain things happen. For example, events can occur when a queue is nearly full or when an application is not authorized to open a queue. These event messages are written to one of the predefined event queues and can then be processed by management tools. All event queues have similar names that indicate the type of event that is held there. SYSTEM.ADMIN.QMGR.EVENT or SYSTEM.ADMIN.CHANNEL.EVENT are examples of queue manager event queues.

- **Cluster Queue:** A cluster queue is a local queue that is configured to advertise its existence within a WebSphere MQ cluster. Applications can refer to this queue without needing any additional definitions, and all the queue managers in the cluster know how to route messages to it. Usually, there are multiple cluster queues of the same name within the cluster. The sending queue manager selects which cluster queue to use based on a workload balancing algorithm.

PCF(Program Command Format)

Control Command + MQSC Scripting Command = PCF Commands Eg. MQExplore

MQ Scripting Commands

```
$ runmqsc <QMN>
```

The scripting mode can manage inside of the queue manager’s objects like queues and their attributes, also their properties.

There are 18 commands inside MQ scripting mode. In this mode, the administrator has to use only uppercase to manage the queue manager’s objects. All these commands have to pass with the required parameters. The example usage will be shown in the next sections.

We can use MQSC commands to manage queue manager objects, including

- Queue managers
- clusters
- channels
- queues
- namelists
- process definitions
- authentication information objects

ALTER	<< to change or modify somethings
CLEAR	<< To clear the terminal in the scripting mode
DEFINE	<< to create new objects.
DELETE	<< to delete new objects.
DISPLAY	<< to show the status, details and changes on objects
END	<< to finish or exit from the scripting mode.
EXIT	<< to finish or exit from the scripting mode.
PING	<< to test the connection between channels.
PURGE	<< delete
QUIT	<< to quit
REFRESH	<< if the rules were changed, but the status is not running. To refresh
RESET	<< Reset the objects rules
RESOLVE	
RESUME	
SET	<< to declare the objects attributes
START	<< to start something like channel or trigger rules
STOP	<< stop the channel or queue.
SUSPEND	<< to suspend the channel or queue.

Add a Diagram for more understanding.
Example Simulation
Example Failure Simulation
Message Desc

CCID
DeadQ Handler
Troubleshooting

1. Planning and Installation
2. Local Queuing
3. Distributed Queuing
4. Multihoping
5. Triggering
6. MQ client
7. MQ Clustering
8. MQ Cluster Overlapping
9. Multi instance Queue Manager
10. Backup And Restoration
11. Troubleshooting

1. Planning and Installation

MQ Server

Licensed Product
Queue Manager
Queue Manager Objects: - Queues, Channel, Listener (port 1414) etc..

MQ Client

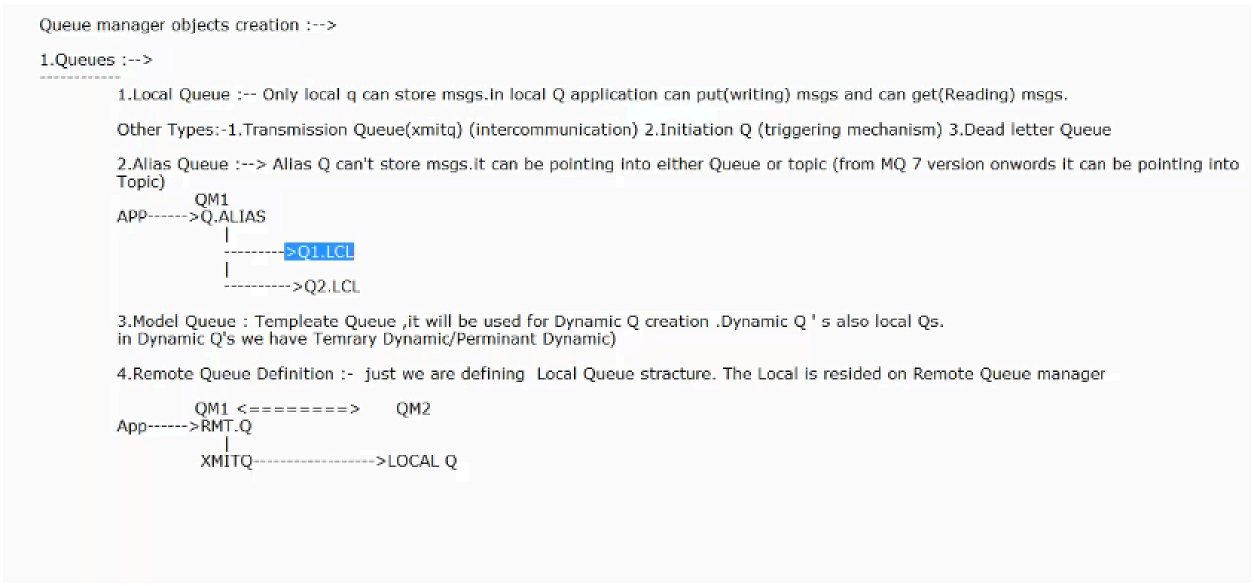
License Free, No Queue Manager, MQI (messaging Queuing Interface), failover functionality.

- MQ Client Server
- MQ Server to Server
- Hub And Spoke

Commands

1. dspmqver
2. crtmqm
3. strmqm
4. endmqm
5. dltmqm
6. dspmq
7. dspmq -o installation -m Queue Manager
8. dspmq -o all

Circular Logging >> Default , rewriting old log files, restart recovery
Linear logging >> Continuous Writing, restart recovery, media recovery



Queue manager objects creation :-->

1.Queues :-->

1.Local Queue :-- Only local q can store msgs.In local Q application can put(writing) msgs and can get(Reading) msgs.

Other Types:-1.Transmission Queue(xmitq) (intercommunication) 2.Initiation Q (triggering mechanism) 3.Dead letter Queue

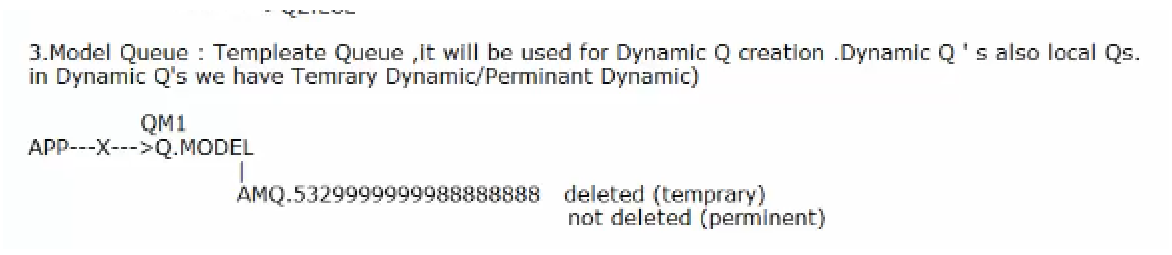
2.Alias Queue :--> Alias Q can't store msgs.it can be pointing into either Queue or topic (from MQ 7 version onwords it can be pointing into Topic)

```
          QM1
APP------>Q.ALIAS
             |
             --------->Q1.LCL
             |
             ---------->Q2.LCL
```

3.Model Queue : Templeate Queue ,it will be used for Dynamic Q creation .Dynamic Q ' s also local Qs.
in Dynamic Q's we have Temrary Dynamic/Perminant Dynamic)

4.Remote Queue Definition :- just we are defining Local Queue stracture. The Local is resided on Remote Queue manager

```
          QM1 <========>     QM2
App------>RMT.Q
             |
           XMITQ----------------->LOCAL Q
```



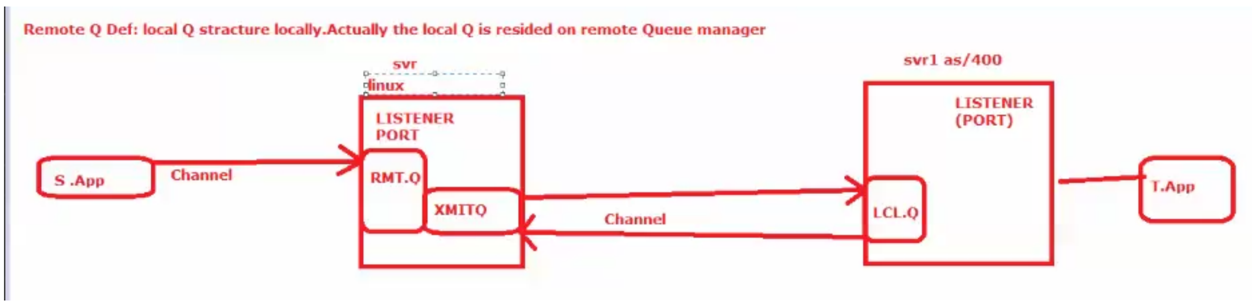
3.Model Queue : Templeate Queue ,it will be used for Dynamic Q creation .Dynamic Q ' s also local Qs.
in Dynamic Q's we have Temrary Dynamic/Perminant Dynamic)

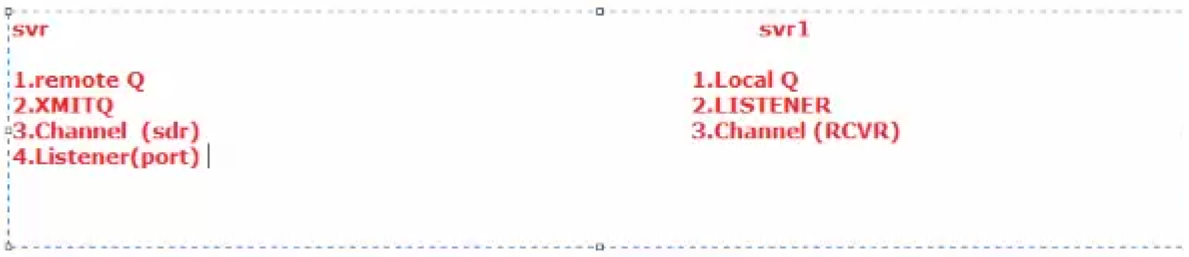
```
          QM1
APP---X--->Q.MODEL
                |
               AMQ.5329999999988888888    deleted (temprary)
                                           not deleted (perminent)
```

amqsput Q1.LCL QMGR
amqsget Q1.LCL QMGR
amqsbcg LQQName QMGR
DSP QL(QL)
DSP QSTATUS(QL)
ALTER QMGR MAXMSGL(41943040)
ALTER QLOCAL(Q1.LCL) MAXDEPTH(50000) MAXMSGL(41943040)
ALTER QLOCAL(Q1.LCL) PUT(DISABLED) GET(ENABLED)
CLEAR QLOCAL(Q1.LCL)
DELETE QLOCAL(Q1.LCL)

QALIAS

DEFINE QALIAS(Q.ALIAS) TARGTYPE(queue) TARGET(Q1.LCL)
amqsput Q.ALIAS QMGR_NAME


Remote Q Def: local Q stracture locally.Actually the local Q is resided on remote Queue manager
svr
linux
LISTENER
PORT
S .App
Channel
RMT.Q
XMITQ
Channel
svr1 as/400
LISTENER
(PORT)
LCL.Q
T.App


svr

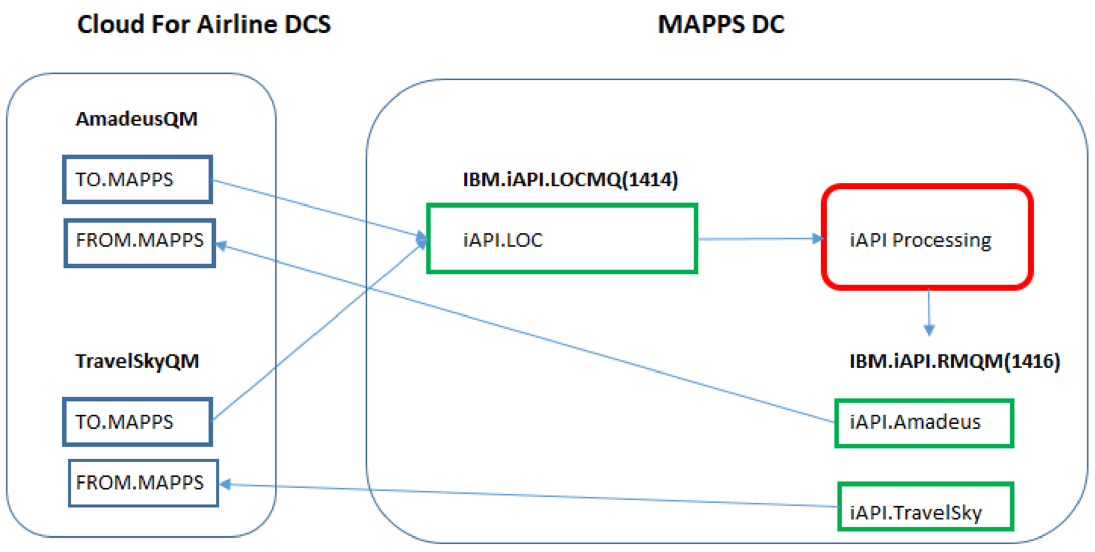
1.remote Q
2.XMITQ
3.Channel (sdr)
4.Listener(port)

svr1

1.Local Q
2.LISTENER
3.Channel (RCVR)

Channel : Logical Communication Link
MQI Channels
Bi-directional Channels
Svrconn (server connection channel)

Best Practice (TWO WAYS Communications)



Bi Directional

DCSQM (Queue Manager)

```
AMADEUS.LOC.IN
DEF QL(AMADEUS.LOC.IN) DEFPSIST(YES) MAXDEPTH(100000)
CUSTOM('CAPEXPRY(600)')
DIS QL(AMADEUS.LOC.IN)
# DEF QL(AMD.IN) DEFPSIST(YES) MAXDEPTH(100000) BOQNAME(AMD.IN.BOQ)
CUSTOM('CAPEXPRY(600)')

ALTER QL(AMADEUS.LOC.IN) STREAMQ(AMADEUS.RMT)

AMADEUS.TX
DEF QL(AMADEUS.TX) TRIGGER INITQ(SYSTEM.CHANNEL.INITQ)
TRIGDATA(AMADEUS.LOC.IN) USAGE(XMITQ) DEFPSIST(YES) MAXDEPTH(100000)

AMADEUS.RMT
DEF QREMOTE(AMADEUS.RMT) RNAME(IAPI.LOC) RQMNAME(IAPIQM)
XMITQ(AMADEUS.TX) DEFPSIST(YES)

SENDER CHANNEL (AMADEUS.TO.IAPI)
DEF CHANNEL(AMADEUS.TO.IAPI) CHLTYPE(SDR) CONNAME('10.20.1.21(1514)')
XMITQ(AMADEUS.TX)

DEFINE SERVICE(CHSERV.AMADEUS) CONTROL(QMGR) STARTCMD('runmqchl')
STARTARG('-m DCSQM -c AMADEUS.TO.IAPI') SERVTYPE(SERVER)
START SERVICE(CHSERV.AMADEUS)
DIS SVSTATUS(CHSERV.AMADEUS)

amqsput AMADEUS.LOC.IN DCSQM
amqsbcg <QN> <QMN>
----------------------------------------

TRAVELSKY.LOC.IN
DEF QL(TRAVELSKY.LOC.IN) DEFPSIST(YES) MAXDEPTH(100000)
CUSTOM('CAPEXPRY(600)')
DIS QL(AMADEUS.LOC.IN)

ALTER QL(TRAVELSKY.LOC.IN) STREAMQ(TRAVELSKY.RMT)

TRAVELSKY.TX
```

```
DEF QL(TRAVELSKY.TX) TRIGGER INITQ(SYSTEM.CHANNEL.INITQ)
TRIGDATA(TRAVELSKY.LOC.IN) USAGE(XMITQ) DEFPSIST(YES) MAXDEPTH(100000)

TRAVELSKY.RMT
DEF QREMOTE(TRAVELSKY.RMT) RNAME(IAPI.LOC) RQMNAME(IAPIQM)
XMITQ(TRAVELSKY.TX) DEFPSIST(YES)

SENDER CHANNEL (TRAVELSKY.TO.IAPI)
DEF CHANNEL(TRAVELSKY.TO.IAPI) CHLTYPE(SDR) CONNAME('10.20.1.21(1515)')
XMITQ(TRAVELSKY.TX)

DEFINE SERVICE(CHSERV.TRAVELSKY) CONTROL(QMGR) STARTCMD('runmqchl')
STARTARG('-m DCSQM -c TRAVELSKY.TO.IAPI') SERVTYPE(SERVER)
START SERVICE(CHSERV.TRAVELSKY)
DIS SVSTATUS(CHSERV.TRAVELSKY)

-------------------------------------
TRAVELSKY.LOC.OUT
DEF QL(TRAVELSKY.LOC.OUT) DEFPSIST(YES) MAXDEPTH(100000)
CUSTOM('CAPEXPRY(600)')
RECIVER CHANNEL (IAPI.TO.TRAVELSKY)
DEF CHANNEL(IAPI.TO.TRAVELSKY) CHLTYPE(RCVR)

LISTNER (Port 1414)
DEF LISTENER(LSTR.DCS) TRPTYPE(TCP) PORT(1414) CONTROL(QMGR)
START LISTENER(LSTR.DCS)

AMADEUS.LOC.OUT
DEF QL(AMADEUS.LOC.OUT) DEFPSIST(YES) MAXDEPTH(100000)
CUSTOM('CAPEXPRY(600)')
RECIVER CHANNEL (IAPI.TO.AMADEUS)
DEF CHANNEL(IAPI.TO.AMADEUS) CHLTYPE(RCVR)

---------------------------------------------------------------------------

IAPIQM (queue manager)

crtmqm IAPIQM
str IAPIQM
runmqsc IAPIQM
-----------

IAPI.LOC
DEF QL(IAPI.LOC) DEFPSIST(YES) MAXDEPTH(100000) CUSTOM('CAPEXPRY(600)')
```

LISTNER (Port 1514)
```
DEF LISTENER(LSTR.IAPI) TRPTYPE(TCP) PORT(1514) CONTROL(QMGR)
START LISTENER(LSTR.IAPI)
```

RECIVER CHANNEL
(AMADEUS.TO.IAPI)
```
DEF CHANNEL(AMADEUS.TO.IAPI) CHLTYPE(RCVR)
```
(TRAVELSKY.TO.IAPI)
```
DEF CHANNEL(TRAVELSKY.TO.IAPI) CHLTYPE(RCVR)
```

AMADEUS.LQ.DCS
AMADEUS.TX.DCS
AMADEUS.RMT.DCS

SENDER CHANNEL (IAPI.TO.AMADEUS)

AMADEUS.LQ.DCS
```
DEF QL(AMADEUS.LQ.DCS) DEFPSIST(YES) MAXDEPTH(100000)
CUSTOM('CAPEXPRY(600)')
DIS QL(AMADEUS.LQ.DCS)

ALTER QL(AMADEUS.LQ.DCS) STREAMQ(AMADEUS.RMT.DCS)
```

AMADEUS.TX.DCS
```
DEF QL(AMADEUS.TX.DCS) TRIGGER INITQ(SYSTEM.CHANNEL.INITQ)
TRIGDATA(AMADEUS.LQ.DCS) USAGE(XMITQ) DEFPSIST(YES) MAXDEPTH(100000)
```

AMADEUS.RMT.DCS
```
DEF QREMOTE(AMADEUS.RMT.DCS) RNAME(AMADEUS.LOC.OUT) RQMNAME(DCSQM)
XMITQ(AMADEUS.TX.DCS) DEFPSIST(YES)
```

SENDER CHANNEL (IAPI.TO.AMADEUS)
```
DEF CHANNEL(IAPI.TO.AMADEUS) CHLTYPE(SDR) CONNAME('10.0.0.22(1414)')
XMITQ(AMADEUS.TX.DCS)

DEFINE SERVICE(CHSERV.IAPI.AMADEUS) CONTROL(QMGR) STARTCMD('runmqchl')
STARTARG('-m IAPIQM -c IAPI.TO.AMADEUS') SERVTYPE(SERVER)
START SERVICE(CHSERV.IAPI.AMADEUS)
DIS SVSTATUS(CHSERV.IAPI.AMADEUS)
```

TRAVELSKY.LQ.DCS
TRAVELSKY.TX.DCS
TRAVELSKY.RMT.DCS
SENDER CHANNEL (IAPI.TO.TRAVELSKY)

TRAVELSKY.LQ.DCS
```
DEF QL(TRAVELSKY.LQ.DCS) DEFPSIST(YES) MAXDEPTH(100000)
CUSTOM('CAPEXPRY(600)')
DIS QL(TRAVELSKY.LQ.DCS)

ALTER QL(TRAVELSKY.LQ.DCS) STREAMQ(TRAVELSKY.RMT.DCS)
```

TRAVELSKY.TX.DCS
```
DEF QL(TRAVELSKY.TX.DCS) TRIGGER INITQ(SYSTEM.CHANNEL.INITQ)
TRIGDATA(TRAVELSKY.LQ.DCS) USAGE(XMITQ) DEFPSIST(YES) MAXDEPTH(100000)
```

TRAVELSKY.RMT.DCS
```
DEF QREMOTE(TRAVELSKY.RMT.DCS) RNAME(TRAVELSKY.LOC.OUT)
RQMNAME(DCSQM) XMITQ(TRAVELSKY.TX.DCS) DEFPSIST(YES)
```

SENDER CHANNEL (IAPI.TO.TRAVELSKY)
```
DEF CHANNEL(IAPI.TO.TRAVELSKY) CHLTYPE(SDR) CONNAME('10.0.0.22(1414)')
XMITQ(TRAVELSKY.TX.DCS)

DEFINE SERVICE(CHSERV.IAPI.TRAVELSKY) CONTROL(QMGR) STARTCMD('runmqchl')
STARTARG('-m IAPIQM -c IAPI.TO.TRAVELSKY') SERVTYPE(SERVER)
START SERVICE(CHSERV.IAPI.TRAVELSKY)
DIS SVSTATUS(CHSERV.IAPI.TRAVELSKY)
```

IAPI.RECEIVEQM

```
crtmqm IAPI.RECEIVEQM
strmqm IAPI.RECEIVEQM
runmqsc IAPI.RECEIVEQM
```

IAPI.LOC
```
DEF QL(IAPI.LOC) DEFPSIST(YES) MAXDEPTH(100000) CUSTOM('CAPEXPRY(600)')
```

LISTNER (Port 1514)
```
DEF LISTENER(LSTR.IAPI) TRPTYPE(TCP) PORT(1514) CONTROL(QMGR)
START LISTENER(LSTR.IAPI)
```

RECIVER CHANNEL
(AMADEUS.TO.IAPI)
```
DEF CHANNEL(AMADEUS.TO.IAPI) CHLTYPE(RCVR)
```
(TRAVELSKY.TO.IAPI)
```
DEF CHANNEL(TRAVELSKY.TO.IAPI) CHLTYPE(RCVR)
```