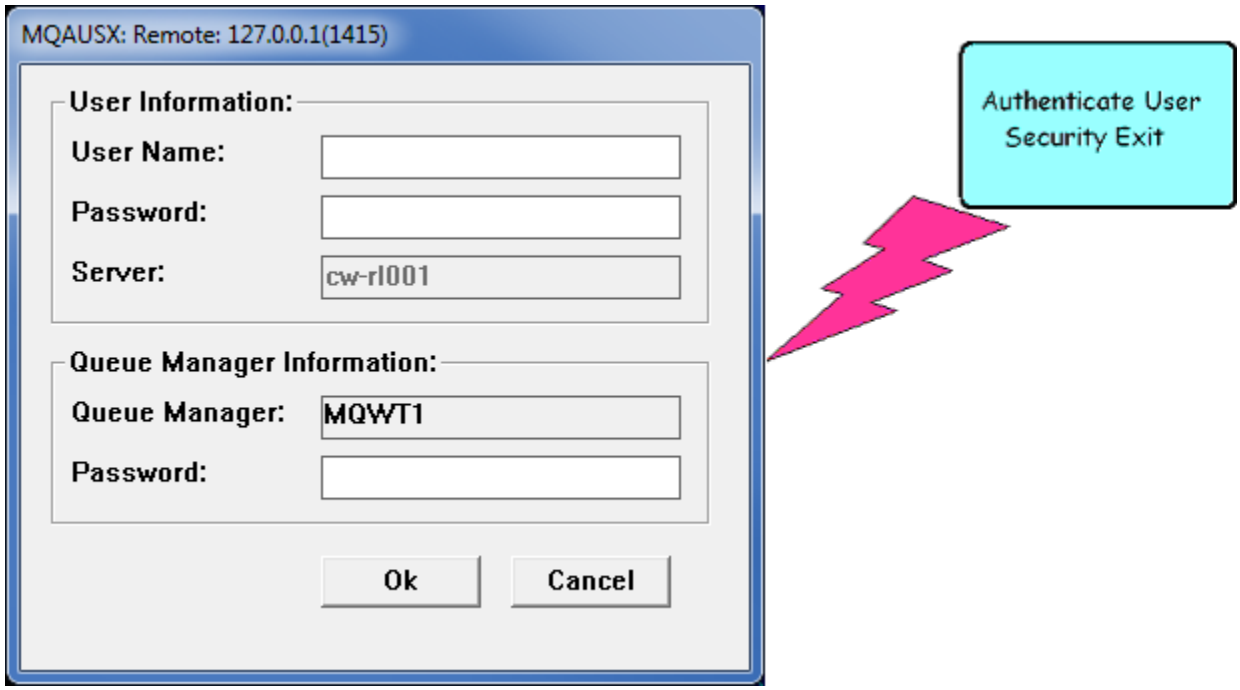
MQAUSX
Programming Guide



Capitalware Inc.
Unit 11, 1673 Richmond Street, PMB524
London, Ontario N6G2N3
Canada
sales@capitalware.com
https://www.capitalware.com

Last Updated: January 2021.

Table of Contents

1 Introduction

1.1 Overview

MQ Authenticate User Security Exit (MQAUSX) is solution that allows a company to fully authenticate a user who is accessing a IBM MQ resource. It authenticates the user's UserId and Password (and possibly Domain Name) against the server's native OS system, LDAP server, Microsoft's Active Directory, Quest Authentication Services, Centrify's DirectControl, Unix/Linux PAM (Pluggable Authentication Module) or an encrypted MQAUSX FBA file.

The security exit will operate with IBM MQ v7.0, v7.1, v7.5, v8.0, v9.0, v9.1 and v9.2 in Windows, Unix and Linux environments. It works with Server Connection, Client Connection, Sender, Receiver, Server and Requestor channels of IBM MQ queue manager.

The MQ Authenticate User Security Exit solution is comprised of 2 components: client-side security exit and server-side security exit.

1.1.1 Client-Side Security Exit

The ***client-side security exit*** first checks if the server-side exit is defined for the particular channel. The client-side exit will receive a security token to be used in the encryption process of the user's password. It will prompt the user for his / her UserId and Password (and domain name for Windows), encrypt the data and send it to the server-side security exit.

For each connection attempt, the server-side security exit will verify that it is an acceptable client exit attempting the connection. If so, then the server-side will send a unique security token. When the server-side security exit receives the encrypted data, it will decrypt the incoming data and then perform UserId and Password (and domain) authentication against the native OS system, LDAP server, Microsoft's Active Directory, Quest Authentication Services, Centrify's DirectControl, Unix/Linux PAM (Pluggable Authentication Module) or an encrypted MQAUSX FBA file. If successful, the connection will be allowed.

1.1.2 Server-Side Security Exit

The ***server-side security exit*** supports the concept of 'Proxy IDs'. After a user has been successfully authenticated against the native OS system, LDAP server, Microsoft's Active Directory, Quest Authentication Services, Centrify's DirectControl, Unix/Linux PAM (Pluggable Authentication Module) or an encrypted MQAUSX FBA file and the 'Proxy Mode' flag is set, then the server-side security exit will look up the user's UserID in the Proxy file for their Proxy ID. The Proxy ID will be used for all MQ interactions.

An MQAdmin can define a password for a queue manager. Hence, when enabled, a back-end application and/or end-user would need to not only know their UserID and Password but also the queue manager's Password to successfully log in. Defining and requiring a queue manager Password in MQAUSX is equivalent to adding perimeter security to your system.

The server-side security exit has the ability to allow or restrict users from logging in with the 'mqm' or 'MUSR_MQADMIN' or 'QMQM' UserIDs. This is controlled by the server-side security exit's property keyword 'Allowmqm'.

The server-side security exit has the capability to allow or limit the incoming channel connections according to the name of the associated Server Connection channel (SVRCONN). Each Server Connection channel can be allocated a maximum number of connections and the server-side security exit will ensure that this maximum is not exceeded.

Client connections to a queue manager are limited by either channel name or the 'DefaultMCC' property keyword in the initialization file. In today's use of J2EE applications, it is a possibility that one J2EE application could overwhelm the queue manager with client connections, thus preventing any connections being made from other applications.

The MQAdmin can enable Excessive Client Connections alerting system that counts the number of connections over a period of time (i.e. Day / Hour / Minute) and writes a message to the log when the count exceeds a particular value. If the keyword WriteToEventQueue is set to ‘Y’ then an event message is also written to an event queue. ECC feature is designed to catch applications that are poorly written, for example, applications that continuously connect and disconnect from the queue manager for every message sent or received.

The server-side security exit has the ability to allow or restrict the incoming IP address, hostname and/or SSL DN. The server-side security exit uses a regular expression parser to parse the incoming client IP address, hostname, and/or SSL DN against a predefined regular expression pattern.

The server-side security exit has the ability to allow or restrict the incoming UserID against a group. A list of groups can be queried for the incoming UserID. The groups can be in the local OS or a group file. If MQAUSX is authenticating against an LDAP server then the group querying can be against the LDAP server.

For those channels where authentication is not required, the server-side security exit can be set to not perform this function. This is controlled by the server-side security exit's property keyword 'NoAuth'.

The server-side security exit, when in non-authentication mode, has the ability to allow or restrict users from connecting with a blank UserID value. This is controlled by the server-side security exit's property keyword 'AllowBlankUserID'.

The server-side security exit, when in non-authentication mode, has the ability to allow or restrict the incoming UserID. The server-side security exit uses a regular expression parser to parse the incoming client UserID against a predefined regular expression pattern.

Note: Raspberry Pi is a Linux ARM 32-bit OS (Operating System). Hence, simply follow the Linux 32-bit instructions for installing and using the solution on a Raspberry Pi.

MQAUSX is 4 products in 1

1. If the client application is configured with the client-side security exit then the user credentials are encrypted and sent to the remote queue manager. This is the best level of security.

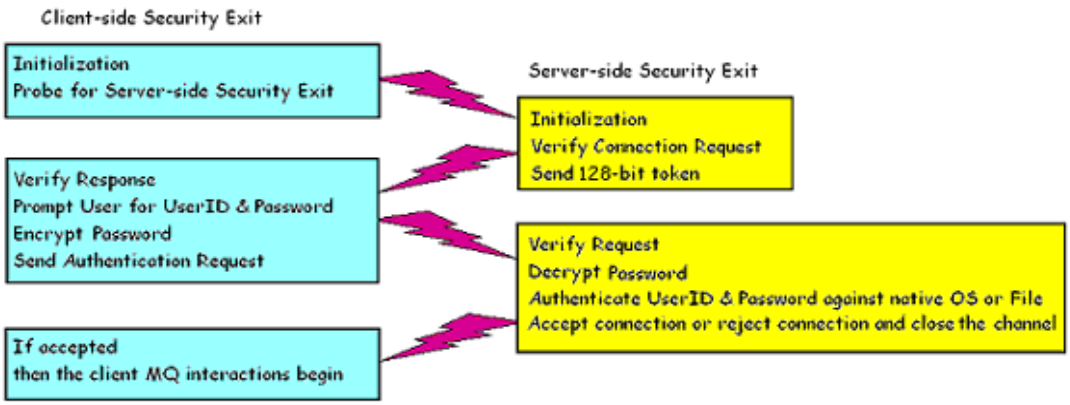
2. If the client application is not configured with the client-side security exit and the client-side **AND** server-side are at MQ V8 then MQ V8 will encrypt the user credentials as they flow from the client application to the queue manager.

3. If the client application is not configured with the client-side security exit then the user credentials are sent in plain text to the remote queue manager. This feature is available for Java/JMS, Java and C# DotNet client applications. For native applications (i.e. C/C++), then the application must use and populate the MQCSP structure with the UserID and Password.
    - Using MQAUSX with No Client-side Security Exit - Part 1 (coding examples) http://www.capitalware.com/rl_blog/?p=638
    - Using MQAUSX with No Client-side Security Exit - Part 2 (configuring tools like MQ Explorer, SupportPac MO71, MQ Visual Edit, etc..) http://www.capitalware.com/rl_blog/?p=659

4. If the MQAdmin sets the MQAUSX IniFile parameter NoAuth to Y then it functions just like MQ Standard Security Exit (MQSSX). MQSSX does not authenticate. It filters the incoming connection based on UserID, IP address, hostname and/or SSL DN.

1.2 Context Diagram (Logical View)

1.3 Security Message Flow (Logical View)



2 Procedural Languages

For Procedural Languages, C and Visual Basic, the programmer has 4 different methods to set the UserId and Password for authentication by the MQAUSX server-side security exit.

If the programmer's application uses the MQCONN API, the MQCONN API can be replaced with the CWMQCONN wrapper using the following method:

1. CWMQCONN, the wrapper for MQCONN API, will pass the UserId and Password directly to the MQAUSX client-side security exit. It is assumed that the user has previously set up an entry in a client channel table for use by their application when using CWMQCONN.

If the programmer's application uses the MQCONNX API, there are 3 choices on how to pass the UserId and Password for authentication. Two of three methods include a newly written API wrapper that replaces the MQCONN and MQCONNX API calls.

1. Use MQCONNX API to pass the UserId and Password via the SecurityUserData field.

2. CWMQCONNX, the wrapper for MQCONNX API, will pass the UserId and Password directly to the MQAUSX client-side security exit.

3. MQCONNX API using MQCSP does NOT interact with the MQAUSX client-side security but instead connects directly to the MQAUSX server-side security exit. Hence, the password will not be encrypted.

2.1 CWMQCONN - Wrapper for MQCONN

This section describes how to use the MQAUSX wrapper (CWMQCONN) for MQCONN to pass the UserId and Password to MQAUSX client-side security exit. The CWMQCONN call replaces the MQCONN API call so that the UserId and Password is passed to the MQAUSX client-side security exit and then invokes the MQCONN API for the calling application.

2.1.1 Syntax

```
CWMQCONN(UserId,
         Password,
         QMName,
         QMPassword,
         HConn,
         CompCode,
         Reason)
```

2.1.2 Parameters

The CWMQCONN call has the following parameters as described below: UserId, Password, QMName, HConn, CompCode and Reason.

- **UserId (char 32) - input**

A UserId to be authenticated by the MQAUSX server-side security exit

- **Password (char 32) - input**

The Password to be authenticated by the MQAUSX server-side security exit

- **QMName (char 48) - input**

The name of the queue manager to which the application wants to connect

- **QM Password (char 32) - input**

The Queue Manager password to be authenticated by the MQAUSX server-side security exit

- **HConn (MQHCONN) - output**

This handle represents the connection to the queue manager.

- **CompCode (MQLONG) - output**

The completion code of the MQCONN API call

- **Reason (MQLONG) - output**

The reason code of the MQCONN API call

2.1.3 Language Invocations

The CWMQCONN call is supported in the programming languages (C and Visual Basic) as shown below. It is assumed that the user has previously set up an entry in a client channel table for use by the user's application.

2.1.3.1 C Language

```
MQHCONN    HConn;
MQLONG     CompCode;
MQLONG     Reason;
char       QMName[MQ_Q_MGR_NAME_LENGTH+1];
char       QMPassword[32+1];
char       UserId[32+1];
char       Password[32+1];

CWMQCONN (UserId,
          Password,
          QMName,
          QMPassword
          &HConn,
          &CompCode,
          &Reason);
```

2.1.3.2 Visual Basic Language

```
Dim QMName As String
Dim QMPassword As String
Dim Hconn As Long
Dim CompCode As Long
Dim Reason As Long
Dim UserId As String
Dim Password As String

CWMQCONN UserId, Password, QMName, QMPassword, Hconn, CompCode, Reason
```

2.2 MQCONNX

This section describes how to use MQCONNX API to pass the UserId and Password to MQAUSX client-side security exit.

2.2.1 Syntax

MQCONNX (QMName, ConnectOptions, HConn, CompCode, Reason)

2.2.2 Parameters

The MQCONNX call has the following parameters as described below: QMName, ConnectOptions , HConn, CompCode and Reason.

- **QMName (char 48) - input**

The name of the queue manager to which the application wants to connect

- **ConnectOptions (MQHCONN) – input / output**

The ConnectOptions allows the application to specify options relating to the connection to the queue manager.

- **HConn (MQHCONN) - output**

This handle represents the connection to the queue manager.

- **CompCode (MQLONG) - output**

The completion code of the MQCONN API call

- **Reason (MQLONG) - output**

The reason code of the MQCONN API call

2.2.3 Language Invocations

The MQCONNX API call is supported in the following programming languages (C and Visual Basic) as shown below.

2.2.3.1 C Language

```
MQCNO     ConnectOptions = {MQCNO_DEFAULT};
MQCD      ClientConn = {MQCD_CLIENT_CONN_DEFAULT};
MQHCONN   HConn;
MQLONG    CompCode;
MQLONG    Reason;
char      QMName[MQ_Q_MGR_NAME_LENGTH+1];
char      channelName[MQ_CHANNEL_NAME_LENGTH+1];
char      hostname[1024];
char      exitName[1024]="C:\\Capitalware\\MQAUSX\\mqausxclnt(ClntExit)"
char      securityData[1024];
char      UserId[32+1];
char      Password[32+1];

strncpy(ClientConn.ConnectionName,
        hostname, MQ_CONN_NAME_LENGTH);

strncpy(ClientConn.ChannelName,
        channelName, MQ_CHANNEL_NAME_LENGTH);

strncpy(ClientConn.SecurityExit,
         exitName, MQ_EXIT_NAME_LENGTH);

/* Specify UserId & Password explicitly. Max of 32 chars.*/
memset(securityData, '\0', sizeof(securityData));
sprintf(securityData, "u=%s;p=%s", UserId, Password);

memcpy(ClientConn.SecurityUserData,
       securityData, MQ_EXIT_DATA_LENGTH);

ConnectOptions.ClientConnPtr = &ClientConn;
ConnectOptions.Version = MQCNO_VERSION_6;

MQCONNX (QMName,
          &ConnectOptions,
          &HConn,
          &CompCode,
          &Reason);
```

2.2.3.2 Visual Basic Language

```
Dim CNOCD As MQCNOCD
Dim QMName As String
Dim Hconn As Long
Dim CompCode As Long
Dim Reason As Long
Dim UserId As String
Dim Password As String

MQCNOCD_DEFAULTS CNOCD

CNOCD.ChannelDef.ConnectionName = GUI_hostname.Text
CNOCD.ChannelDef.ChannelName = GUI_chlName.Text
CNOCD.ChannelDef.Version = MQCD_CURRENT_VERSION
CNOCD.ChannelDef.SecurityExit = "C:\Capitalware\MQAUSX\mqausxclnt(ClntExit)"
CNOCD.ChannelDef.SecurityUserData = "u=" & UserId & ";p=" & Password

MQCONNXAny QMName, CNOCD, Hconn, CompCode, Reason
```

2.3 CWMQCONNX - Wrapper for MQCONNX

This section describes how to use the MQAUSX wrapper (CWMQCONNX) for MQCONNX to pass the UserId and Password to the MQAUSX client-side security exit. CWMQCONNX call replaces the MQCONNX API call so that the UserId and Password is passed to the MQAUSX client-side security exit to invoke the MQCONNX API for the calling application.

2.3.1 Syntax

```
CWMQCONNX(UserId,
          Password,
          QMName,
          QMPassword,
          HConn,
          CompCode,
          Reason)
```

2.3.2 Parameters

The CWMQCONNX call has the following parameters as described below: UserId, Password, QMName, HConn, CompCode and Reason.

- **UserId (char 32) - input**

A UserId to be authenticated by the MQAUSX server-side security exit

- **Password (char 32) - input**

The Password to be authenticated by the MQAUSX server-side security exit

- **QMName (char 48) - input**

The name of the queue manager to which the application wants to connect

- **QMPassword (char 32) - input**

The Queue Manager password to be authenticated by the MQAUSX server-side security exit

- **HConn (MQHCONN) - output**

This handle represents the connection to the queue manager.

- **CompCode (MQLONG) - output**

The completion code of the MQCONN API call

- **Reason (MQLONG) - output**

The reason code of the MQCONN API call

2.3.3 Language Invocations

The CWMQCONNX call is supported in the following programming languages (C and Visual Basic) as shown below.

2.3.3.1 C Language

```
MQCNO     ConnectOptions = {MQCNO_DEFAULT};
MQCD      ClientConn = {MQCD_CLIENT_CONN_DEFAULT};
MQHCONN   HConn;
MQLONG    CompCode;
MQLONG    Reason;
char      QMName[MQ_Q_MGR_NAME_LENGTH+1];
char      QMPassword[32+1];
char      channelName[MQ_CHANNEL_NAME_LENGTH+1];
char      hostname[1024];
char      exitName[1024]="C:\\Capitalware\\MQAUSX\\mqausxclnt(ClntExit)"
char      UserId[32+1];
char      Password[32+1];

strncpy(ClientConn.ConnectionName,
        hostname, MQ_CONN_NAME_LENGTH);

strncpy(ClientConn.ChannelName,
        channelName, MQ_CHANNEL_NAME_LENGTH);

strncpy(ClientConn.SecurityExit,
         exitName, MQ_EXIT_NAME_LENGTH);

ConnectOptions.ClientConnPtr = &ClientConn;
ConnectOptions.Version = MQCNO_VERSION_2;

CWMQCONNX(UserId,
          Password,
          QMName,
          QMPassword,
          &ConnectOptions,
          &HConn,
          &CompCode,
          &Reason);
```

2.3.3.2 Visual Basic Language

```
Dim CNOCD As MQCNOCD
Dim QMName As String
Dim QMPassword As String
Dim Hconn As Long
Dim CompCode As Long
Dim Reason As Long
Dim UserId As String
Dim Password As String

MQCNOCD_DEFAULTS CNOCD

CNOCD.ChannelDef.ConnectionName = GUI_hostname.Text
CNOCD.ChannelDef.ChannelName = GUI_chlName.Text
CNOCD.ChannelDef.Version = MQCD_CURRENT_VERSION
CNOCD.ChannelDef.SecurityExit = "C:\Capitalware\MQAUSX\mqausxclnt(ClntExit)"

CWMQCONNX UserId, Password, QMName, QMPassword, CNOCD, Hconn, CompCode, Reason
```

2.4 MQCONNX using MQCSP

This section describes how to use MQCONNX API with MQCSP to pass the UserId and Password to MQAUSX server-side security exit. The MQAUSX client-side security exit is not involved with this interaction; hence, the Password is not encrypted between the application and the remote queue manager.

2.4.1 Syntax

MQCONNX (QMName, ConnectOptions, HConn, CompCode, Reason)

2.4.2 Parameters

The MQCONNX call has the following parameters as described below: QMName, ConnectOptions , HConn, CompCode and Reason.

- **QMName (char 48) - input**

The name of the queue manager to which the application wants to connect

- **ConnectOptions (MQHCONN) – input / output**

The ConnectOptions allows the application to specify options relating to the connection to the queue manager.

- **HConn (MQHCONN) - output**

This handle represents the connection to the queue manager.

- **CompCode (MQLONG) - output**

The completion code of the MQCONN API call

- **Reason (MQLONG) - output**

The reason code of the MQCONN API call

2.4.3 Language Invocations

The MQCONNX API call is supported in the following programming languages (C and Visual Basic) as shown below.

2.4.3.1 C Language

```
MQCNO     ConnectOptions = {MQCNO_DEFAULT};
MQCD      ClientConn = {MQCD_CLIENT_CONN_DEFAULT};
MQCSP     mqCSP = {MQCSP_DEFAULT};
MQHCONN   HConn;
MQLONG    CompCode;
MQLONG    Reason;
char      QMName[MQ_Q_MGR_NAME_LENGTH+1];
char      channelName[MQ_CHANNEL_NAME_LENGTH+1];
char      hostname[1024];
char      UserId[32+1];
char      Password[32+1];

strncpy(ClientConn.ConnectionName,
        hostname, MQ_CONN_NAME_LENGTH);

strncpy(ClientConn.ChannelName,
        channelName, MQ_CHANNEL_NAME_LENGTH);

mqCSP.AuthenticationType = MQCSP_AUTH_USER_ID_AND_PWD;

mqCSP.CSPUserIdPtr = &UserId;
mqCSP.CSPUserIdOffset = 0;
mqCSP.CSPUserIdLength = strlen(UserId);

mqCSP.CSPPasswordPtr = &Password;
mqCSP.CSPPasswordOffset = 0;
mqCSP.CSPPasswordLength = strlen(Password);

ConnectOptions.SecurityParmsPtr = &mqCSP;
ConnectOptions.SecurityParmsOffset = 0;
ConnectOptions.ClientConnPtr = &ClientConn;
ConnectOptions.Version = MQCNO_VERSION_2;

MQCONNX (QMName,
          &ConnectOptions,
          &HConn,
          &CompCode,
          &Reason);
```

2.4.3.2 Visual Basic Language

```
Dim CNOCD As MQCNOCD
Dim CSP As MQCSP
Dim QMName As String
Dim Hconn As Long
Dim CompCode As Long
Dim Reason As Long
Dim UserId As String
Dim Password As String

MQCNOCD_DEFAULTS CNOCD
MQCSP_DEFAULTS CSP

CNOCD.ChannelDef.ConnectionName = GUI_hostname.Text
CNOCD.ChannelDef.ChannelName = GUI_chlName.Text
CNOCD.ChannelDef.Version = MQCD_CURRENT_VERSION

CSP.AuthenticationType = MQCSP_AUTH_USER_ID_AND_PWD

CSP.CSPUserIdPtr = StrPtr(UserId)
CSP.CSPUserIdOffset = 0
CSP.CSPUserIdLength = Len(UserId)

CSP.CSPPasswordPtr = StrPtr(Password)
CSP.CSPPasswordOffset = 0
CSP.CSPPasswordLength = Len(Password)

CNOCD.ConnectOpts.SecurityParmsPtr = CSP
CNOCD.ConnectOpts.SecurityParmsOffset = 0

MQCONNXAny QMName, CNOCD, Hconn, CompCode, Reason
```

3 C++ Language

For the C++ Language, the programmer has 4 different methods to set the UserId and Password for authentication by the MQAUSX server-side security exit. A new C++ class (MQAUSXClient) was written to pass the UserId and Password to the MQAUSX client-side security exit.

If the programmer's application uses the ImqQueueManager class, the MQAUSXCLient class needs to be used.

1. The MQAUSXCLient class will pass the UserId and Password directly to the MQAUSX client-side security exit.

If the programmer's application uses the ImqQueueManager and ImqChannel classes, they have the following 3 choices on how to pass the UserId and Password for authentication:

1. For the ImqQueueManager and ImqChannel classes, the UserId and Password can be passed via the SecurityUserData field to the MQAUSX client-side security exit.

2. For the ImqQueueManager and ImqChannel classes, the UserId and Password can be passed via the setUserId and setPassword methods of ImqChannel class, to the MQAUSX client-side security exit.

3. For the ImqQueueManager and ImqChannel classes using the MQCSP class, the UserId and Password can be passed directly to the MQAUSX server-side security exit. In this scenario, the application does NOT interact with the MQAUSX client-side security; hence, the password will not be encrypted.

3.1 MQAUSXClient Class

This section describes how to use the MQAUSXClient class to pass the UserId and Password to MQAUSX client-side security exit. Use the setCredentials method of the MQAUSXClient class to pass the UserId and Password to the MQAUSX client-side security exit.

3.1.1 Syntax

```
mqausx = new MQAUSXClient;
mqausx->setCredentials(UserId, Password, QMName, QMPassword);
```

3.1.2 Parameters

The MQAUSXClient class has the following parameters as described below: UserId, Password and QMName.

- **UserId (char 32) - input**

A UserId to be authenticated by the MQAUSX server-side security exit

- **Password (char 32) - input**

The Password to be authenticated by the MQAUSX server-side security exit

- **QMName (char 48) - input**

The name of the queue manager to which the application wants to connect

- **QMPassword (char 32) - input**

The Queue Manager password to be authenticated by the MQAUSX server-side security exit

3.1.3 Language Invocations

The MQAUSXClient class is supported in the following programming language (C++) as shown below. It is assumed that the user has previously set up an entry in a client channel table for use by the user's application.

3.1.3.1 C++ Language

```
ImqQueueManager mgr;
MQAUSXClient   *mqausx;
char           QMName[MQ_Q_MGR_NAME_LENGTH+1];
char           QMPassword[32+1];
char           UserId[32+1];
char           Password[32+1];

mgr.setName(QMName);
mqausx = new MQAUSXClient;

if ((mqausx->setCredentials(UserId,
                            Password,
                            QMName,
                            QMPassword)) != CW_OK)
{
   delete mqausx;
   return( 1 );
}
else
{
   if ( ! mgr.connect( ) )
   {
      delete mqausx;
      return( 1 );
   }

   delete mqausx;
}
```

3.2 ImqQueueManager and ImqChannel (MQCONNX)

This section describes how to use the ImqQueueManager and ImqChannel classes to pass the UserId and Password to MQAUSX client-side security exit.

3.2.1 Syntax

```
ImqQueueManager     mgr;
ImqChannel          *pchannel;
mgr.setName( QMName );
pchannel -> setChannelName( ChannelName );
pchannel -> setConnectionName( ConnName );
pchannel -> setSecurityExitName(ExitName);
pchannel -> setSecurityUserData(SecurityData);
```

3.2.2 Parameters

The ImqQueueManager and ImqChannel classes require the following parameters as described below: QMName, Channelname, ConnName, ExitName and SecurityData.

- **QMName (char 48) - input**

The name of the queue manager to which the application wants to connect

- **ChannelName (char 20) – input**

The name of the channel to use for the connection

- **ConnName (char 264) - input**

The ConnName is the hostname or IP address and Port Number of the remote server where the queue manager is located.

- **ExitName (char 128) – input**

The full path and name of the MQAUSX client-side security exit

- **SecurityData (char 32) – input**

The security data will contain the UserId and Password that is being passed to the MQAUSX client-side security exit.

3.2.3 Language Invocations

The ImqQueueManager and ImqChannel classes are supported in the following programming language (C++) as shown below.

3.2.3.1 C++ Language

```
ImqQueueManager  mgr;
ImqChannel      *pchannel = 0;

char      QMName[MQ_Q_MGR_NAME_LENGTH+1];
char      QMPassword[32+1];
char      channelName[MQ_CHANNEL_NAME_LENGTH+1];
char      hostname[1024];
char      exitName[1024]="C:\\Capitalware\\MQAUSX\\mqausxclnt(ClntExit)"
char      securityData[1024];
char      UserId[32+1];
char      Password[32+1];

mgr.setName(QMName);

pchannel = new ImqChannel ;
pchannel -> setHeartBeatInterval( 1 );
pchannel -> setTransportType( MQXPT_TCP );
pchannel -> setChannelName(channelName);
pchannel -> setConnectionName(hostname);
pchannel -> setSecurityExitName(exitName);
mgr.setChannelReference( pchannel );

/* Specify UserId & Password explicitly. Max of 32 chars.*/
memset(securityData, '\0', sizeof(securityData));
sprintf(securityData, "u=%s;p=%s", UserId, Password);
pchannel -> setSecurityUserData( securityData );

if ( ! mgr.connect( ) )
{
  delete pchannel;
  return( 1 );
}
```

3.3 ImqQueueManager and ImqChannel (MQCONNX)

This section describes how to use the ImqQueueManager and ImqChannel classes to pass the UserId and Password to MQAUSX client-side security exit.

3.3.1 Syntax

```
ImqQueueManager     mgr;
ImqChannel          *pchannel;
mgr.setName( QMName );
pchannel -> setChannelName( ChannelName );
pchannel -> setConnectionName( ConnName );
pchannel -> setSecurityExitName(ExitName);
pchannel -> setUserId( UserId );
pchannel -> setPassword( Password );
```

3.3.2 Parameters

The ImqQueueManager and ImqChannel classes require the following parameters as described below: QMName, ChannelName, ConnName, ExitName, UserId and Password.

- **QMName (char 48) - input**

The name of the queue manager to which the application wants to connect

- **ChannelName (char 20) – input**

The name of the channel to use for the connection

- **ConnName (char 264) - input**

The ConnName is the hostname or IP address and Port Number of the remote server where the queue manager is located.

- **ExitName (char 128) – input**

The full path and name of the MQAUSX client-side security exit

- **UserId (char 12***) - input**

A UserId to be authenticated by the MQAUSX server-side security exit

- **Password (char 12***) - input**

The Password to be authenticated by the MQAUSX server-side security exit.

*** The 12-character limit is an MQ limit and not a limit of MQAUSX. To pass a longer UserId or Password, please review the details in section 3.2 or 3.4.

3.3.3 Language Invocations

The ImqQueueManager and ImqChannel classes are supported in the following programming language (C++) as shown below.

3.3.3.1 C++ Language

```
ImqQueueManager  mgr;
ImqChannel      *pchannel = 0;

char      QMName[MQ_Q_MGR_NAME_LENGTH+1];
char      channelName[MQ_CHANNEL_NAME_LENGTH+1];
char      hostname[1024];
char      exitName[1024]="C:\\Capitalware\\MQAUSX\\mqausxclnt(ClntExit)"
char      UserId[32+1];
char      Password[32+1];

mgr.setName(QMName);

pchannel = new ImqChannel ;
pchannel -> setHeartBeatInterval( 1 );
pchannel -> setTransportType( MQXPT_TCP );
pchannel -> setChannelName(channelName);
pchannel -> setConnectionName(hostname);
pchannel -> setSecurityExitName(exitName);
mgr.setChannelReference( pchannel );

/* Specify UserId & Password via Channel class. Max of 12 chars. */
pchannel -> setUserId( myUserId );
pchannel -> setPassword( myPassword );

if ( ! mgr.connect( ) )
{
  delete pchannel;
  return( 1 );
}
```

3.4 ImqQueueManager and ImqChannel with MQCSP (MQCONNX)

This section describes how to use ImqQueueManager and ImqChannel classes with MQCSP to pass the UserId and Password to MQAUSX server-side security exit. The MQAUSX client-side security exit is not involved with this interaction; hence, the Password is not encrypted between the application and the remote queue manager.

3.4.1 Syntax

```
ImqQueueManager     mgr;
ImqChannel          *pchannel;
mgr.setName( QMName );
pchannel -> setChannelName( ChannelName );
pchannel -> setConnectionName( ConnName );
mgr.setAuthenticationType(MQCSP_AUTH_USER_ID_AND_PWD);
mgr.setUserId( UserId );
mgr.setPassword( Password );
```

3.4.2 Parameters

The ImqQueueManager and ImqChannel classes have the following parameters as described below: QMName, ChannelName, ConnName, UserId and Password.

- **QMName (char 48) - input**

The name of the queue manager to which the application wants to connect

- **ChannelName (char 20) – input**

The name of the channel to use for the connection

- **ConnName (char 264) - input**

The ConnName is the hostname or IP address and Port Number of the remote server where the queue manager is located.

- **UserId (char 32) - input**

A UserId to be authenticated by the MQAUSX server-side security exit

- **Password (char 32) - input**

The Password to be authenticated by the MQAUSX server-side security exit

3.4.3 Language Invocations

The ImqQueueManager and ImqChannel classes with MQCSP is supported in the following programming language (C++) as shown below.

3.4.3.1 C++ Language

```
ImqQueueManager  mgr;
ImqChannel      *pchannel = 0;

char      QMName[MQ_Q_MGR_NAME_LENGTH+1];
char      channelName[MQ_CHANNEL_NAME_LENGTH+1];
char      hostname[1024];
char      UserId[32+1];
char      Password[32+1];

mgr.setName(QMName);

pchannel = new ImqChannel ;
pchannel -> setHeartBeatInterval( 1 );
pchannel -> setTransportType( MQXPT_TCP );
pchannel -> setChannelName(channelName);
pchannel -> setConnectionName(hostname);
mgr.setChannelReference( pchannel );

/* Specify UserId and Password via MQCSP */
mgr.setAuthenticationType(MQCSP_AUTH_USER_ID_AND_PWD);
mgr.setUserId( UserId );
mgr.setPassword( Password );

if ( ! mgr.connect( ) )
{
  delete pchannel;
  return( 1 );
}
```

4 Java Language

There are 2 distinct approaches for Java programming for WMQ: IBM MQ base Java and IBM MQ base JMS (Java Messaging Service).

If the programmer's application uses the IBM MQ base Java, the MQAUSXJ class must be used for authentication.

- The MQAUSXJ class is the MQAUSX client-side security exit for IBM MQ base Java. The UserId and Password can be passed directly during the class instantiation.

If the programmer's application uses the IBM MQ base JMS, the MQAUSXJ2EE class must be used for authentication.

- The MQAUSXJ2EE class is the MQAUSX client-side security exit for MQ base JMS. The UserId and Password can be passed directly via the createQueueConnection method of the QueueConnection class.

4.1 IBM MQ base Java

This section describes how to instantiate MQAUSXJ base Java. There are three ways to instantiate the MQAUSXJ base Java client-side security exit.

4.1.1 Syntax

```
new MQAUSXJ();
new MQAUSXJ(filename);
new MQAUSXJ(userId, password);
new MQAUSXJ(userId, password, qmPassword);
```

4.1.2 Parameters

The MQAUSXJ base Java instantiation can include the following parameters as described below: none or filename or UserId and Password.

4.1.2.1 Filename (String) – input

The filename represents the name of the property file (IniFile) that contains the UserId and Password values.

4.1.2.2 UserId (String) - input

A UserId to be authenticated by the MQAUSX server-side security exit.

4.1.2.3 Password (String) - input

The Password to be authenticated by the MQAUSX server-side security exit.

4.1.2.4 qmPassword (String) - input

The Queue Manager password to be authenticated by the MQAUSX server-side security exit.

4.1.3 Exceptions

The following exceptions may be encountered:

- IllegalArgumentException
  Invalid / illegal value supplied as an argument to the call.

- FileNotFoundException
  The specified property file (IniFile) could not be found at the location given.

4.1.4 Language Invocations

The MQAUSXJ base Java only supports the Java programming language.

4.1.4.1 Java Language

Sample #1 does not pass an IniFile or UserId & Password to the MQAUSXJ client-side security exit; hence the exit will display a log on pop-up to the end-user.

```
String qManager;
MQEnvironment.hostname = "10.10.10.10(1414)";
MQEnvironment.channel = "TEST.CHL";
MQEnvironment.securityExit = new MQAUSXJ();

MQQueueManager _qMgr = new MQQueueManager(qManager);
```

Sample #2 passes an IniFile to the MQAUSXJ class. The IniFile contains the UserId and Password that will be used by the MQAUSXJ client-side security exit.

```
String qManager;
MQEnvironment.hostname = "10.10.10.10(1414)";
MQEnvironment.channel = "TEST.CHL";
MQEnvironment.securityExit = new MQAUSXJ("C:\\Capitalware\\MQAUSX\\clnt.ini");

MQQueueManager _qMgr = new MQQueueManager(qManager);
```

Sample #3 passes the UserId and Password directly to the MQAUSXJ client-side security exit.

```
String qManager;
String userID;
String password;
MQEnvironment.hostname = "10.10.10.10(1414)";
MQEnvironment.channel = "TEST.CHL";
MQEnvironment.securityExit = new MQAUSXJ(userId, password);

MQQueueManager _qMgr = new MQQueueManager(qManager);
```

Sample #4 passes the UserId and Password indirectly via the MQEnvironment class to the MQAUSXJ client-side security exit. Note: The UserId and Password cannot be longer than 12 characters; otherwise, MQ will truncate them.

```
String qManager;
String userID;
String password;
MQEnvironment.hostname = "10.10.10.10(1414)";
MQEnvironment.channel = "TEST.CHL";
/* Old MQ syle */
MQEnvironment.userID = userID;
MQEnvironment.password = password;
MQEnvironment.securityExit = new MQAUSXJ();

MQQueueManager _qMgr = new MQQueueManager(qManager);
```

4.2 IBM MQ base JMS

This section describes how to use the createQueueConnection method of the QueueConnection class to pass the UserId and Password to the MQAUSX client-side security exit.

4.2.1 Syntax

```
createQueueConnection(userID, password);
```

4.2.2 Parameters

The createQueueConnection method of the QueueConnection class can include the following parameters as described below: UserId and Password.

4.2.2.1 UserId (String) - input

A UserId to be authenticated by the MQAUSX server-side security exit.

4.2.2.2 Password (String) - input

The Password to be authenticated by the MQAUSX server-side security exit.

4.2.3 Exceptions

The following exceptions may be encountered:

- javax.jms.SecurityException
  The supplied UserId and/or Password is invalid.

4.2.4 Language Invocations

The MQAUSXJ2EE for JMS only supports the Java/JMS programming language.

4.2.4.1 Java/JMS Language

Sample #1 uses a QCF via an MQJNDI entry. The QCF entry includes the definition for the MQAUSXJ2EE client-side security exit. The JMS layer passes the UserId and Password to the MQAUSXJ2EE client-side security exit via the createQueueConnection method of the QueueConnectionFactory.

```
QueueConnectionFactory qcf;
QueueConnection connection;
String userID;
String password;
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, JNDI_CONTEXT);
env.put(Context.PROVIDER_URL, "file:/C:\JNDI\test\mqjndi");

Context ctx = new InitialContext(env);
qcf = (QueueConnectionFactory) ctx.lookup(myQCF);
connection = qcf.createQueueConnection(userID, password);
```

Sample #2 uses a dynamically created QCF. The programmer must explicitly set the MQAUSXJ2EE client-side security exit via the setSecurityExit method of the QCF. The JMS layer passes the UserId and Password to the MQAUSXJ2EE client-side security exit via the createQueueConnection method of the QueueConnectionFactory.

```
MQQueueConnectionFactory mqQCF;
QueueConnection connection;
String qManager;
String userID;
String password;

mqQCF = new MQQueueConnectionFactory();
mqQCF.setQueueManager(qManager);
mqQCF.setHostName("10.10.10.10(1414)");
mqQCF.setChannel("TEST.CHL");
mqQCF.setTransportType(JMSC.MQJMS_TP_CLIENT_MQ_TCPIP);
mqQCF.setSecurityExit("biz.capitalware.mqausx.MQAUSXJ2EE");

connection = mqQCF.createQueueConnection(userID, password);
```

5 .NET C-Sharp Language

For the .NET C-Sharp Language, the programmer has 3 different methods to set the UserId and Password for authentication by the MQAUSX server-side security exit. Two methods use the new MQAUSXDN .NET class under a managed .NET environment and the other method uses the native mqausx.dll under an unmanaged .NET environment.

If the programmer's application uses a managed .NET environment, the MQAUSXDN class must be used for authentication.

- The MQAUSXDN class is the MQAUSX client-side security exit for a managed .NET environment. The UserId and Password can be passed directly during the class using the MQEnvironment class.

If the programmer's application uses an unmanaged .NET environment, the native mqausx.dll must be used for authentication.

- The mqausxclnt.dll is the native MQAUSX client-side security exit for an unmanaged .NET environment. The UserId and Password can be passed directly via the SecurityUserData field.

5.1 Managed .NET Environment

This section describes how to instantiate MQAUSXDN class.

5.1.1 Syntax

```
MQEnvironment.SecurityExit="C:\\Capitalware\\MQAUSX\\mqausxdn.dll(Capitalware.MQAUSXDN)";
```

5.1.2 Parameters

There are no parameters for the MQAUSXDN class.

5.1.3 Exceptions

There are no MQAUSXDN exceptions.

5.1.4 Language Invocations

The MQAUSXDN class supports any managed .NET language (e.g. C-Sharp .NET and VB.NET).

5.1.4.1 C-Sharp Language

```
String qManager;
MQEnvironment.Hostname = "10.10.10.10(1414)";
MQEnvironment.Channel = "TEST.CHL";
MQEnvironment.SecurityExit="C:\\Capitalware\\MQAUSX\\
mqausxdn.dll(Capitalware.MQAUSXDN)";

MQEnvironment.UserId = "userID";
MQEnvironment.Password = "password";

MQQueueManager _qMgr = new MQQueueManager(qManager);
```

6 Appendix A – Sample Client Channel Table

The following are sample Client Channel Table entries that can be used with the sample code for MQCONN (ImqQueueManager), CWMQCONN or MQQueueManager (see Appendix C for sample code).

6.1 Windows

```
DEFINE CHANNEL('TEST.CHL') CHLTYPE(CLNTCONN) +
   TRPTYPE(TCP) CONNAME('10.10.10.10(1414)') QMNAME('MQA1') +
   SCYDATA(' ') SCYEXIT('C:\Capitalware\MQAUSX\mqausxclnt(ClntExit)')
```

6.2 Unix and Linux for IBM MQ 32-bit

```
DEFINE CHANNEL('TEST.CHL') CHLTYPE(CLNTCONN) +
   TRPTYPE(TCP) CONNAME('10.10.10.10(1414)') QMNAME('MQA1') +
   SCYDATA(' ') SCYEXIT('/var/mqm/exits/mqausxclnt(ClntExit)')
```

6.3 Unix and Linux for IBM MQ 64-bit

```
DEFINE CHANNEL('TEST.CHL') CHLTYPE(CLNTCONN) +
   TRPTYPE(TCP) CONNAME('10.10.10.10(1414)') QMNAME('MQA1') +
   SCYDATA(' ') SCYEXIT('/var/mqm/exits64/mqausxclnt(ClntExit)')
```

6.4 Java Applications

```
DEFINE CHANNEL('TEST.CHL') CHLTYPE(CLNTCONN) +
   TRPTYPE(TCP) CONNAME('10.10.10.10(1414)') QMNAME('MQA1') +
   SCYDATA(' ') SCYEXIT('biz.capitalware.mqausx.MQAUSXJE6')
```

7 Appendix B – Sample MQJNDI

The following are sample MQJNDI entries that can be used by the Java/JMS code samples (see Appendix C for sample code):

7.1 JMS Queue Connection Factory (QCF) Sample:

```
DEFINE QCF(myQCF) QMANAGER(MQA1) CHANNEL(TEST.CHL)
       HOSTNAME(10.10.10.10) PORT(1414)
       SECEXIT(biz.capitalware.mqausx.MQAUSXJ2EE)
       FAILIFQUIESCE(YES) TRANSPORT(CLIENT)
```

7.2 JMS Queue Sample:

```
DEFINE Q(mqs.test.q) QUEUE(TEST.Q1) QMANAGER(MQA1)
       TARGCLIENT(JMS) FAILIFQUIESCE(YES)
```

8 Appendix C – MQAUSX Language Files

The following is the directory structure layout followed by the Language files:

Windows Directory Structure	Unix Directory Structure
C: +--Capitalware +--MQAUSX <- Install Directory +--samples +--c +--cpp +--cs +--java +--vb	<Install_Directory> +--Capitalware +--MQAUSX +--samples +--c +--cpp +--java

8.1 MQAUSX C Sample Files

The MQAUSX C sample files are installed in the following directories:

Platform	Directory
Linux / Unix	<Install_Directory>/samples/c/
Windows	C:\Capitalware\MQAUSX\samples\c\

8.1.1 List of C sample files

Filename	Description
MQTest01.c	Demonstrates how to use the CWMQCONN wrapper to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager.
MQTest02.c	Demonstrates how to use the CWMQCONN wrapper to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager.
MQTest11.c	Demonstrates how to use the MQCONNX API with the MQAUSX client-side security exit to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set via the SecurityUserData field..*
MQTest12.c	Demonstrates how to use the MQCONNX API with the MQAUSX client-side security exit to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set via the SecurityUserData field.*
MQTest21.c	Demonstrates how to use the CWMQCONNX wrapper to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager.
MQTest22.c	Demonstrates how to use the CWMQCONNX wrapper to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager.
MQTest31.c	Demonstrates how to use the MQCONNX API and MQCSP structure to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager.
MQTest32.c	Demonstrates how to use the MQCONNX API and MQCSP structure to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager.

8.2 MQAUSX C++ Sample Files

The MQAUSX C sample files are installed in the following directories:

Platform	Directory
Linux / Unix	<Install_Directory>/samples/cpp/
Windows	C:\Capitalware\MQAUSX\samples\cpp\

8.2.1 List of C++ sample files

Filename	Description
MQTest01.cpp	Demonstrates how to use the ImqQueueManager class with the MQAUSXClient class to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager.
MQTest02.cpp	Demonstrates how to use the ImqQueueManager class with the MQAUSXClient class to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager.
MQTest11.cpp	Demonstrates how to use the ImqQueueManager and ImqChannel classes with the MQAUSX client-side security exit to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set via the setSecurityUserData method of the ImqChannel class.*
MQTest12.cpp	Demonstrates how to use the ImqQueueManager and ImqChannel classes with the MQAUSX client-side security exit to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set via the setSecurityUserData method of the ImqChannel class.*
MQTest21.cpp	Demonstrates how to use the ImqQueueManager and ImqChannel classes with the MQAUSX client-side security exit to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set via the setUserId and SetPassword methods of the ImqChannel class.*
MQTest22.cpp	Demonstrates how to use the ImqQueueManager and ImqChannel classes with the MQAUSX client-side security exit to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set via the setUserId and SetPassword methods of the ImqChannel class.*
MQTest31.cpp	Demonstrates how to use the ImqQueueManager and ImqChannel classes along with the MQCSP class to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager.
MQTest32.cpp	Demonstrates how to use the ImqQueueManager and ImqChannel classes along with the MQCSP class to connect to a queue manager then how to open a queue, ***get*** a message to a queue, close the queue and disconnect from a queue manager.

8.3 MQAUSX base Java & JMS Sample Files

The MQAUSX base Java and JMS sample files are installed in the following directories:

Platform	Directory
Linux / Unix	<Install_Directory>/samples/java/
Windows	C:\Capitalware\MQAUSX\samples\java\

8.3.1 List of Java sample files

Filename	Description
MQTest01.java	Demonstrates how to use the MQQueueManager, MQEnvironment and MQAUSXJ class to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager.
MQTest02.java	Demonstrates how to use the MQQueueManager, MQEnvironment and MQAUSXJ class to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager.
MQTest11.java	Demonstrates how to use the MQQueueManager, HashTable and MQAUSXJ class to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager.
MQTest12.java	Demonstrates how to use the MQQueueManager, HashTable and MQAUSXJ class to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager.
MQTest21.java	Demonstrates how to use the MQQueueManager, MQEnvironment and MQAUSXJ class to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set via the MQEnvironment class.*
MQTest22.java	Demonstrates how to use the MQQueueManager, MQEnvironment and MQAUSXJ class to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set via the MQEnvironment class.*
MQTest41.java	Demonstrates how to use the MQQueueManager class with a Client Channel Table to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set in the security exit data field of the Client Channel Table entry.*
MQTest42.java	Demonstrates how to use the MQQueueManager class with a Client Channel Table to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set in the security exit data field of the Client Channel Table entry.*

8.3.2 List of Java/JMS sample files

Filename	Description
MQTestJMS01.java	Demonstrates how to use the QueueConnectionFactory (QCF) via MQJNDI and MQAUSXJ2EE class to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager.
MQTestJMS02.java	Demonstrates how to use the QueueConnectionFactory (QCF) via MQJNDI and MQAUSXJ2EE class to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager.
MQTestJMS11.java	Demonstrates how to use the QueueConnectionFactory (QCF) and MQAUSXJ2EE class to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager.
MQTestJMS12.java	Demonstrates how to use the QueueConnectionFactory (QCF) and MQAUSXJ2EE class to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager.

8.4 .NET C-Sharp Sample Files

The MQAUSX .NET C-Sharp sample files are installed in the following directories:

Platform	Directory
Windows	C:\Capitalware\MQAUSX\samples\cs\

8.4.1 List of .NET C-Sharp sample files

Filename	Description
MQTest01.cs	Demonstrates how to use the MQQueueManager, MQEnvironment and MQAUSXDN class to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set via the MQEnvironment class.*
MQTest02.cs	Demonstrates how to use the MQQueueManager, MQEnvironment and MQAUSXDN class to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set via the MQEnvironment class.*
MQTest11.cs	Demonstrates how to use the MQQueueManager, MQEnvironment and MQAUSXDN class to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set via the MQEnvironment class.*
MQTest12.cs	Demonstrates how to use the MQQueueManager, MQEnvironment and MQAUSXDN class to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set via the MQEnvironment clas.*
MQTest31.cs	Demonstrates how to use the MQQueueManager, MQCSP and MQEnvironment class to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set via the MQEnvironment class.*
MQTest32.cs	Demonstrates how to use the MQQueueManager, MQCSP and MQEnvironment class to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set via the MQEnvironment class.*
MQTest41.cs	Demonstrates how to use the MQQueueManager class (unmanaged .NET) with a Client Channel Table to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set in the security exit data field of the Client Channel Table entry.*
MQTest42.cs	Demonstrates how to use the MQQueueManager class (unmanaged .NET) with a Client Channel Table to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager. *The UserId and Password are set in the security exit data field of the Client Channel Table entry.*

8.5 MQAUSX Visual Basic Sample Files

The MQAUSX Visual Basic sample files are installed in the following directories:

Platform	Directory
Windows	C:\Capitalware\MQAUSX\samples\vb\

8.5.1 List of Visual Basic sample files

Filename	Description
MQTest01.frm	Demonstrates how to use the CWMQCONN wrapper to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager.
MQTest02.frm	Demonstrates how to use the CWMQCONN wrapper to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager.
MQTest11.frm	Demonstrates how to use the MQCONNX API with the MQAUSX client-side security exit to connect to a queue manager then how to open a queue, ***put*** a message to a queue, close the queue and disconnect from a queue manager.
MQTest12.frm	Demonstrates how to use the MQCONNX API with the MQAUSX client-side security exit to connect to a queue manager then how to open a queue, ***get*** a message from a queue, close the queue and disconnect from a queue manager.

9 Appendix D – License Agreement

This is a legal agreement between you (either an individual or an entity) and Capitalware Inc. By opening the sealed software packages (if appropriate) and/or by using the SOFTWARE, you agree to be bound by the terms of this Agreement. If you do not agree to the terms of this Agreement, promptly return the disk package and accompanying items for a full refund.
SOFTWARE LICENSE

1. GRANT OF LICENSE. This License Agreement (License) permits you to use one copy of the software product identified above, which may include user documentation provided in on-line or electronic form (SOFTWARE). The SOFTWARE is licensed as a single product, to an individual user, or group of users for Muliple User Licenses and Site Licenses. This Agreement requires that each user of the SOFTWARE be Licensed, either individually, or as part of a group. A Multi-User License provides for a specified number of users to use this SOFTWARE at any time. This does not provide for concurrent user Licensing. Each user of this SOFTWARE must be covered either individually, or as part of a group Multi-User License. The SOFTWARE is in use on a computer when it is loaded into the temporary memory (i.e. RAM) or installed into the permanent memory (e.g. hard disk) of that computer. This software may be installed on a network provided that appropriate restrictions are in place limiting the use to registered users only.

2. COPYRIGHT. The SOFTWARE is owned by Capitalware Inc. and is protected by United States Of America and Canada copyright laws and international treaty provisions. You may not copy the printed materials accompanying the SOFTWARE (if any), nor print copies of any user documentation provided in on-line or electronic form. You must not redistribute the registration codes provided, either on paper, electronically, or as stored in the files MQAUSX.ini or any other form.

3. OTHER RESTRICTIONS. The registration notification provided, showing your authorization code and this License is your proof of license to exercise the rights granted herein and must be retained by you. You may not rent or lease the SOFTWARE, but you may transfer your rights under this License on a permanent basis, provided you transfer this License, the SOFTWARE and all accompanying printed materials, retain no copies, and the recipient agrees to the terms of this License. You may not reverse engineer, decompile, or disassemble the SOFTWARE, except to the extent the foregoing restriction is expressly prohibited by applicable law.

LIMITED WARRANTY

LIMITED WARRANTY. Capitalware Inc. warrants that the SOFTWARE will perform substantially in accordance with the accompanying printed material (if any) and on-line documentation for a period of 365 days from the date of receipt.

CUSTOMER REMEDIES. Capitalware Inc. entire liability and your exclusive remedy shall be, at Capitalware Inc. option, either (a) return of the price paid or (b) repair or replacement of the SOFTWARE that does not meet this Limited Warranty and that is returned to Capitalware Inc. with a copy of your receipt. This Limited Warranty is void if failure of the SOFTWARE has resulted from accident, abuse, or misapplication. Any replacement SOFTWARE will be

warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer.

NO OTHER WARRANTIES. To the maximum extent permitted by applicable law, Capitalware Inc. disclaims all other warranties, either express or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to the SOFTWARE and any accompanying written materials.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES. To the maximum extent permitted by applicable law, in no event shall Capitalware Inc. be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use or inability to use the SOFTWARE, even if Capitalware Inc. has been advised of the possibility of such damages.

10 Appendix E – Notices

Trademarks:

AIX, IBM, MQSeries, OS/2 Warp, OS/400, IBM i, MVS, OS/390, WebSphere, IBM MQ and z/OS are trademarks of International Business Machines Corporation.

HP-UX is a trademark of Hewlett-Packard Company.

Intel is a registered trademark of Intel Corporation.

Java, J2SE, J2EE, Sun and Solaris are trademarks of Sun Microsystems Inc.

Linux is a trademark of Linus Torvalds.

Mac OS X is a trademark of Apple Computer Inc.

Microsoft, Visual Basic, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation.

UNIX is a registered trademark of the Open Group.

WebLogic is a trademark of BEA Systems Inc.