Nasdaq

CTCI WebSphere MQ V1.1
Subscriber Intercommunication Specification

Version 1.2

July 2002

Confidentiality/Disclaimer

This Specification is being forwarded to you strictly for informational purposes solely for the purpose of developing or operating systems for your use that interact with systems of The Nasdaq Stock Market, Inc. (Nasdaq) and its affiliates (collectively, the Corporations). This specification is proprietary to Nasdaq. Nasdaq reserves the right to withdraw, modify, or replace the specification at any time, without notice. No obligation is made by Nasdaq regarding the level, scope, or timing of Nasdaq's implementation of the functions or features discussed in this specification. The specification is "AS IS", "WITH ALL FAULTS" and Nasdaq makes no warranties, and disclaims all warranties, express, implied, or statutory related to the specifications. THE CORPORATIONS ARE NOT LIABLE FOR ANY INCOMPLETENESS OR INACCURACIES. THE CORPORATIONS ARE NOT LIABLE FOR ANY CONSEQUENTIAL, INCIDENTAL, OR INDIRECT DAMAGES RELATING TO THE SPECIFICATIONS OR THEIR USE. It is further agreed by you by using this specification, that you agree not to copy, reproduce, or permit access to the information contained in, the specification except to those with a need-to-know for the purpose noted above.

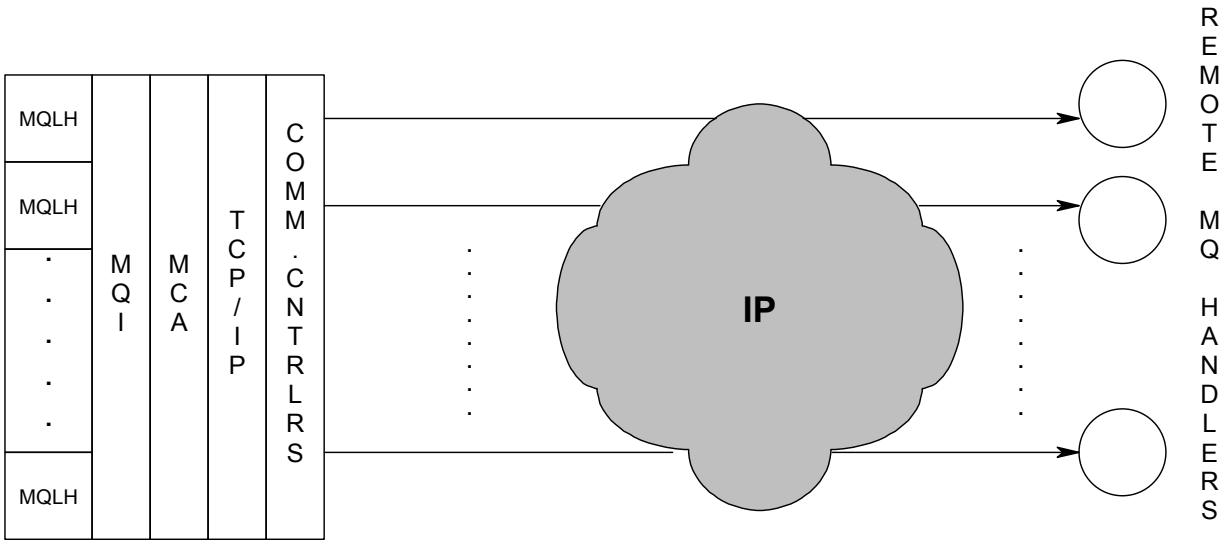
TABLE OF CONTENTS

Purpose of This Document

This document describes how a subscriber can submit and receive Computer to Computer Interface (CTCI) messages utilizing the NASDMS (Switch) through IBM WebSphere MQ (formerly MQ Series) Middleware using WebSphere MQ API calls over TCP/IP protocol. This document also describes the required CTCI-MQ intercommunication specifics for a subscriber.

The CTCI-MQ Interface discussed in this document employs the WebSphere MQ Distributed Queuing technique.

CTCI-MQ Interface Connection Topology



Nasdaq Tandem CTCI Node(s)

MCA- Message Channel Agent
MQI – Message Queuing Interface
TCP/IP – TCP/IP stack
MQLH – WebSphere MQ Line Handler at Nasdaq

MQLH is the line handler at Nasdaq that communicates with the remote line handler at the remote subscriber. This MQLH communicates with the remote MQ line interfaces through a remote/local queue pair. There is a one-to-one correspondence between the remote/local queue pair and the origin/destination station-id (device-id).

CTCI-MQ Interface Setup

Before attempting to establish a connection over WebSphere MQ, the subscriber must contact Nasdaq to obtain the four relevant IP addresses: two for the Primary and Alternate links and two for the disaster recovery site. Each client will also be assigned a fixed port number in the range of 40500 – 40700. A client profile must be established for each IP address assigned to a subscriber. This involves assigning as many remote/local queue pairs as the subscriber will use to exchange CTCI Messages with Nasdaq.

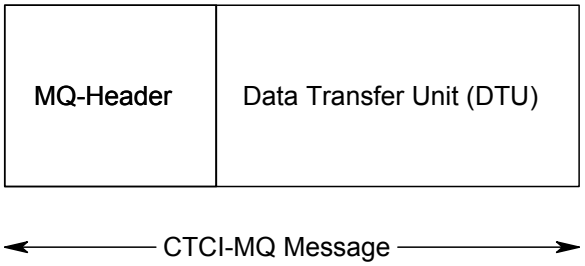
Over one active MQ remote/local queue pair, a subscriber can submit and receive CTCI Messages on behalf of one station/device location. For each of the stations or device locations that the subscriber will be submitting and receiving CTCI Messages on behalf of, the subscriber and Nasdaq will assign remote/local queue pairs. Every station has its own unique application sequence numbering scheme (refer to “Subscriber Requirements for Computer to Computer Interface Utilizing the NASDMS Switch” documentation).

Detailed MQ Configuration of WebSphere MQ entities are discussed in the CTCI WebSphere MQ Configuration Management Section. Note that a Nasdaq-supplied MQ Form helps the subscriber to define the MQ entities required for this interface. (See Appendix B of this document.)

CTCI-MQ Interface Message Format and Flow

CTCI-MQ Message Format

The CTCI-MQ Message is the complete MQ message flowing between Nasdaq and the subscriber. The CTCI-MQ Message has an MQ header followed by a Data Transfer Unit (DTU).



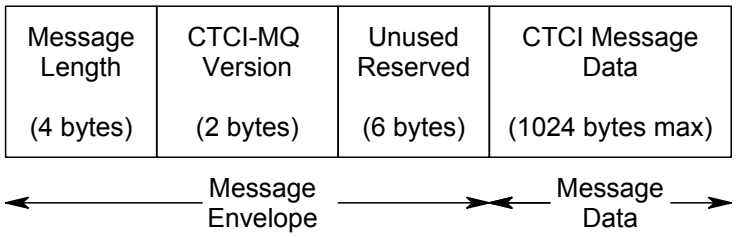
The underlying MQ API call builds the MQ Header. However, the various entities of the MQ Header have to be set programmatically prior to every MQPUT and MQGET. (See CTCI-MQ Programming Management section.)

The DTU is the data that would be built, sent, and received by the MQLH-Nasdaq MQ Line Handler. (See Data Transfer Unit section.)

CTCI-MQ Interface Message Format and Flow

DTU Format

The DTU sends and receives CMS Messages through the MQ CHANNEL pair. The DTU consists of a message “envelope” and CTCI message data.



Message Length is a four-byte field that contains the total length of the DTU, including the length of the *Message Length* field at the beginning. Currently, the largest CTCI Message Data that can be sent by the subscriber is 1024 bytes. Hence, the maximum value of message length would be 1036.

Version is the two-byte CTCI-MQ implementation version number. Currently it is version 11 (1.1). Note that this is different from the WebSphere MQ version defined in the MQ Header.

If the DTU data length exceeds 1036, the data will be truncated and returned as an application error to the originator.

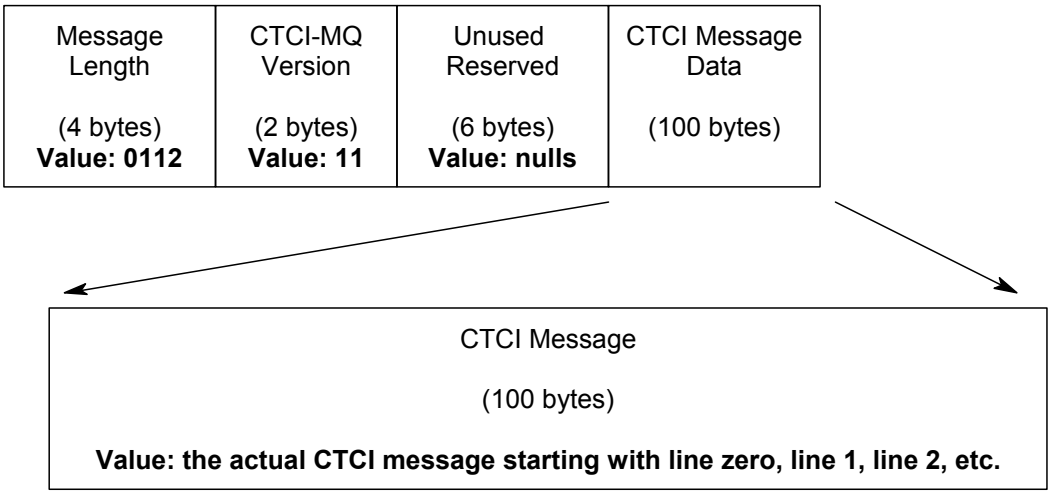
All fields defined here will be in ASCII. WebSphere MQ data conversion – MQGMO_CONVERT will be used. (See page 17.)

CTCI-MQ Interface Message Format and Flow

Sending a CTCI Message

CTCI Messages are formatted as usual (refer to the “Subscriber Requirements for Computer to Computer Interface Utilizing the NASDMS Switch” documentation), but when delivered, they must be imbedded in a CTCI Message “envelope.”

Here is an example of a 100-byte long CTCI Message sent across a particular remote/local queue pair for a origin/destination station pair:



The *Unused* field is an ASCII field that should always be filled with ASCII spaces (blanks).

CTCI Message is the CTCI Message Data, formatted as usual to start at the beginning of line zero, as described in the “Subscriber Requirements for Computer to Computer Interface Utilizing the NASDMS Switch” documentation.

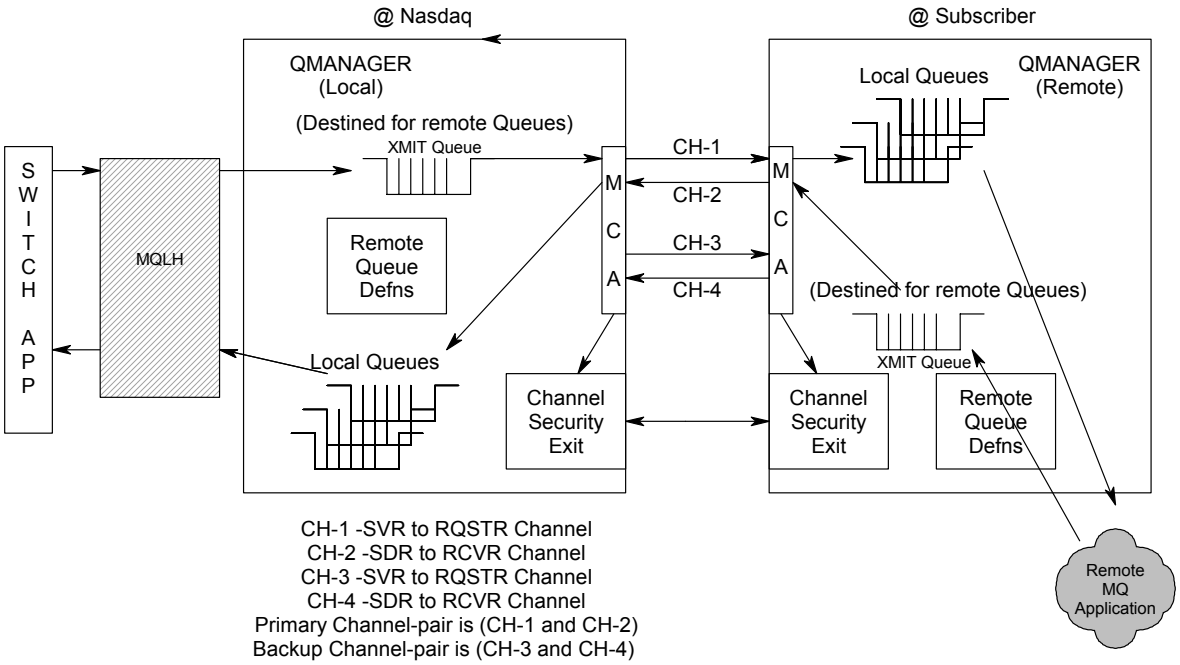
CTCI-MQ Interface Message Format and Flow

CTCI Message Sequence Verification

It is the responsibility of the subscriber to detect and recover lost data by implementing CTCI Message sequence number checking and message retrieval processing. Every station has its own sequence numbering scheme (0001 to 9999 and wrapping back to 0001 after 9999). Refer to the "Standard Input" and "Standard Output" links located on the CTCI Specifications page at http://www.nasdaqtrader.com/asp/ctcidisclaim4.asp for a detailed description of these procedures.

CTCI WebSphere MQ Management

MQ Configuration



All Channels are over TCP-IP links. The directed lines on CH-1, 2, 3 and 4 show the direction of the data-flow.

The CTCI-MQ connection between the Nasdaq CTCI switch and the remote subscriber is through an MQ line handler for a remote/local queue pair. The Message queuing interface between the Nasdaq CTCI Switch and the subscriber is through a list of remote/local queue pairs defined at the Subscriber's Queue Manager.

Two pairs of channels are defined for the Primary and Alternate link. Each pair has one Server/Requester and one Receiver/Sender channel defined, a sender and requester channel defined at the Subscriber Queue Manager, and a receiver and server channel defined at the Nasdaq Queue Manager (see the CTCI-MQ Configuration Management section).

The subscriber starts both the primary channels (sender and requester defined at the subscriber).

In the event of a Primary IP-link failure, the SDR/SVR channels will try after *short-timer (2)* seconds for *short-retry (5)* times. After (2X5 =) 10 seconds, the subscriber should call its own procedure to switch to the Alternate link and connect and bring up the alternate channels (CH-3 and CH-4). Apart from establishing the link and re-authenticating, this procedure should also reset and re-synchronize the sequence numbers, pick up the messages from the subscriber's transmit queue and recommence transmission of messages in doubt and thereafter. It is the responsibility of the subscriber to develop this procedure.

CTCI WebSphere MQ Management

MQ Configuration (Contd.)

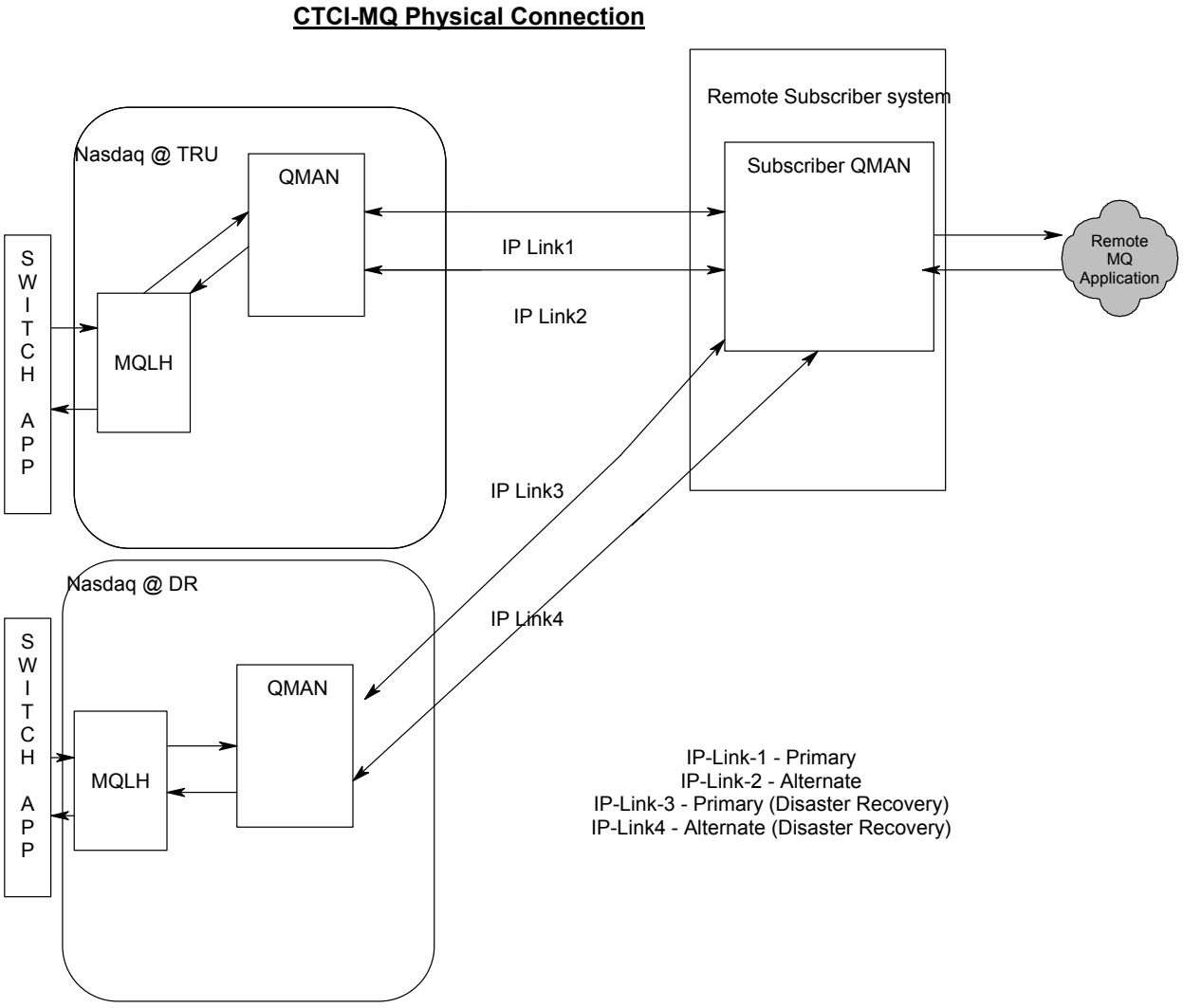
An error queue should be defined at the subscriber end by the name ERROR.<SUBS>, where SUBS is the MMID (Market Maker Identifier) of the subscriber. In the event of a CTCI-MQ application error, the messages are passed back to this error queue along with a nine-byte CTCI-MQ error code (format: CTCI-MQnn) at the top of the message delivered. These CTCI-MQ error codes are listed in Appendix E of this document.

Channel Security exits are executed immediately after the channel connection is established to authenticate the subscriber's identity through MCAUSER and security data information. If a security breach should occur, the channel suppresses data transfers. No data would flow through the defined channel thereafter (see Authentication section), and a source code for security exits would be provided (see Appendix C of the document).

The Remote/Local Queues, SVR, RQSTR, SDR, and RCVR Channel Definitions are documented in Appendix A of this document.

CTCI WebSphere MQ Management

CTCI Physical Connectivity



For every (SDR, RQSTR) channel pair at the subscriber, there exists an IP link. Two physical links (Primary and Alternate) from the subscriber to Nasdaq production and two physical links (Primary and Alternate) from the subscriber to the Nasdaq Disaster Recovery site must be defined. All configurations at the disaster recovery site would be similar to the one at the production environment except for the IP addresses and channel names.

Note that, with the exception of the channels, the DR site is always ready. In the instance of a disaster, Nasdaq will notify the remote subscriber should to connect to the Primary DR IP-Link (IP-link 3).

At both Nasdaq Production and the Disaster Recovery site, the alternate link is a hot standby with a listener process up and running all the time.

CTCI WebSphere MQ Management

Connection Establishment and Recovery

Establishment

The subscriber establishes two TCP/IP connections with the server and receiver channels at Nasdaq using the IP address and the port number assigned during the MQ primary channel setup.

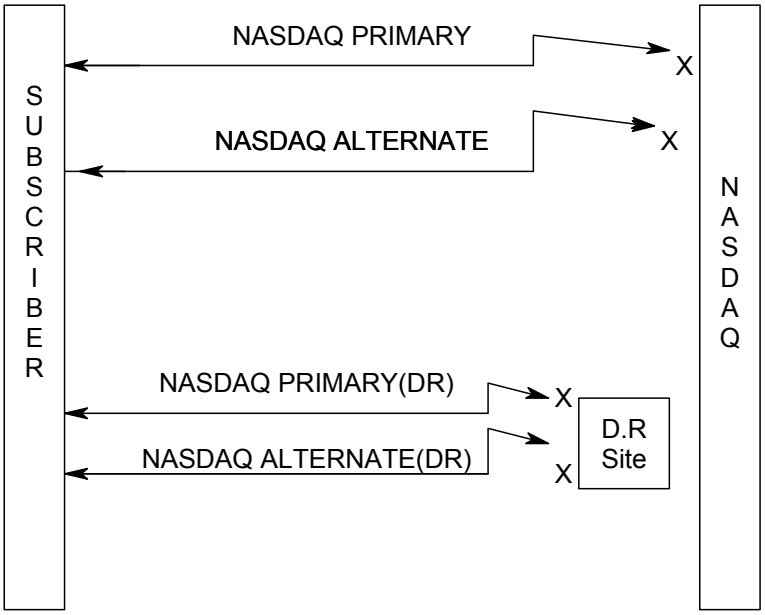
The two primary channels (SDR and RQSTR at the subscriber end) should be established well before the market open.

Note that the primary and alternate listener processes for the assigned port number would be functioning before the start of every market day at Nasdaq. Once the primary connection is established from the subscriber, the “Channel-Level Security Exit” would be invoked; the Security Exit at Nasdaq would authenticate the subscriber and, therefore, the data flow. In the event that the authentication fails, no data flows between the subscriber and Nasdaq.

CTCI WebSphere MQ Management

Connection Establishment and Recovery (Contd.)

Recovery



X =Subscriber Sender/Requester Channel Disconnect

Initially, the connection to the subscriber is through the Nasdaq PRIMARY link. If the link fails, the system should retry for 10 seconds before moving to the Nasdaq ALTERNATE link and connecting to the alternate channel pair. The data flow then resumes from where it stopped. If the Nasdaq ALTERNATE link fails to connect for another 10 seconds, the connection should revert back to the PRIMARY link. This back-and-forth cycle should continue for a time until a manual interruption takes it to the disaster recovery mode and connects it to the PRIMARY link of the DR site. This takes effect if disaster hits the Nasdaq production site, and Nasdaq identifies it as such and subsequently informs the subscriber. The listener processes for both PRIMARY and ALTERNATE links of the Nasdaq DR site are always ready for a connect from the remote subscriber. Therefore, when the subscriber connects to the Nasdaq DR PRIMARY link, the transmission would restart.

The subscriber should implement for the DR site an identical retry logic to the one implemented at the Nasdaq production site between the PRIMARY and ALTERNATE links.

CTCI WebSphere MQ Management

CTCI-MQ Configuration Management

The Transmit Queue with the same name as the remote Nasdaq queue manager should be defined at the subscriber end.

A sender and a requester type channel, one for the primary and the other for the alternate link, has to be defined at the subscriber end.

A list of local queues and remote queues for every station has to be defined at the subscriber end. (See Appendix A.)

Channel security exits, compiled and bound in the MQ API source code, handle MCA User-level Security. (See the section on Authentication.)

The following parameters would be set at the channel definitions:

BATCHSZ as 50. (at both SDR and RQSTR Channel definitions)
HBINT as 10. (at both SDR and RQSTR Channel definitions)
SHORTTMR as 2. (at SDR Channel definitions)
SHORTRTY as 5. (at SDR Channel definitions)

No LONGRTY and LONGTMR would be implemented. MQ-Sequence numbers will be reset through a RESET CHANNEL command at the start of every market day. Also, the ‘‘Sequence number wrap” will be set to 999999999.

An electronic MQ Form will be supplied to every subscriber for creates/updates of the MQ configuration at the subscribing firm. (See Appendix B.) This facilitates Nasdaq and the subscriber in coding the MQ definitions appropriately.

CTCI WebSphere MQ Management

CTCI-MQ Programming Management

MQ Message Delivery:
All messages will be defined to be PERSISTENT across the board.
Only STANDARD BINDINGS will be used.
Message delivery sequence will be defined as FIFO.

MQMD *MessDesc* – is the Message Descriptor Structure (At MQPUT and MQGET)

MessDesc.Expiry = MQEI_UNLIMITED
MessDesc.Persistence = MQPER_PERSISTENT
MessDesc.PutTime = time in ”HHMMSSCC” format
MessDesc.Encoding = MQENC_NATIVE
MessDesc.MsgId set to MQMI_NONE
MessDesc.Format set to MQFMT_STRING **#**

MQFMT_STRING should be used by the subscriber and by Nasdaq to implement GMO_CONVERT for translation.

At MQCONN or MQCONNX:
MQCNO_STANDARD_BINDINGS would be set at Connect *Options*.

At MQOPEN:

While Opening an Output Queue:
Output *Options* set to MQOO_OUTPUT + MQOO_FAIL_IF_QUIESCING

While Opening an Input Queue:
Input *Options* set to MQOO_INPUT_AS_Q_DEF + MQOO_FAIL_IF_QUIESCING

At MQGET and MQPUT:
MQGMO *gmo* – is the Get Message Option Structure
MQGMO_CONVERT (required)
MQPMO *pmo* – is the Put Message Option Structure
The subscriber is NOT limited by the usage of any other *pmo* or *gmo* options.

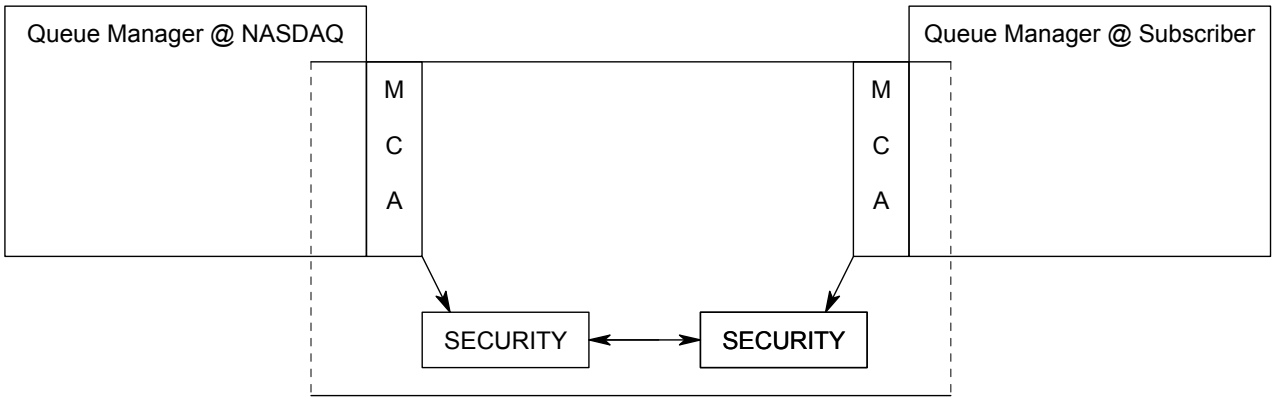
Note that the **CCSID** used at Nasdaq is **819**.

CTCI WebSphere MQ Management

Authentication

Authentication of subscribers at Nasdaq is achieved by implementing MCAUSER identifier at the subscriber's sender and Nasdaq's receiver channels in conjunction with Security Exit Routines at the MCA-level. (See Appendix A.)

The channel exits provide for queue manager authentication via an exchange protocol between partner message channel agents. This assures that they are allowed to enter into a session with Nasdaq CTCI Switch. The function is implemented in the channel's security exit**.**



Channel security exits are executed immediately after the channel connection is established to authenticate the subscriber's identity through MCAUSER and security data information. In the event of a security breach, the channel suppresses any function to the subscriber, thereby disabling message transfer.

The programming logic of the security exit at both Nasdaq and the subscriber are explained on the following page.

CTCI WebSphere MQ Management

Authentication (Contd.)

The receiving channel initiates the handshaking procedure. The security exit is invoked by the following events:

1. First, both channel security exits get called with MQXR_INIT. Both return MQXCC_OK to continue the communication.
2. Next, the sender's channel security exit gets called with MQXR_INIT_SEC. It builds the NAM and initiates the transmission of the NAM to the receiver's channel security exit by returning MQXCC_SEND_SEC_MSG. It also requests a reply to be returned by the sender's channel security exit.
3. The receiver' s channel security exit gets invoked with MQXR_SEC_MSG. It receives the NAM and checks its validity. If the NAM is not valid, the receiver's channel security exit returns MQXX_SUPRESS_FUNCTION, which stops the communication. If the NAM is valid, the receiver's channel security exit builds a NAM as an answer. The answer must contain the message content of the NAM, just received. It returns MQXCC_SEND_SEC_MSG, which initiates the transmission of the NAM to the sender's channel security exit.
4. Finally, the sender gets invoked with MQXR_SEC_MSG. It checks the NAM, and, if the NAM is valid, it returns MQXCC_OK, and the security environment is established. Now message transmission can begin. If the NAM was not valid, MQXCC_SUPRESS_FUNCTION will be returned to close the channel.

   NAM – Nasdaq Authentication Message

   In this authentication flow, Nasdaq will act as the receiver and the subscriber as the sender**.**

   Structure of NAM:

```
struct {
        char mca_name[12];
        char password[10];
} NAM;
```

   See Appendix C for subscriber sample channel security exit program.

CTCI WebSphere MQ Management

Authentication (Contd.)

Exit @ Subscriber (When in Agreement)	**Exit @ NASDAQ (When in Agreement)**
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG Sends the NAM Message---------->	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG <----------Sends the NAM back
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK Message Transfer Begins	 Message Transfer Begins
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

Exit @ Subscriber (When NOT in Agreement)	**Exit @ NASDAQ (When NOT in Agreement)**
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG Sends NAM Message---------->	Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION Channel Closes
Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION Channel Closes	

CTCI WebSphere MQ Management

Naming Conventions

All WebSphere MQ entities of CTCI-MQ Configuration follow WebSphere MQ naming standards.

The Channel Naming convention:
Production Channels:
NASD.SUBS.PRI
SUBS.NASD.PRI
NASD.SUBS.ALT
SUBS.NASD.ALT

Disaster Recovery Channels:
NASD.SUBS.DR.PRI
SUBS.NASD.DR.PRI
NASD.SUBS.DR.ALT
SUBS.NASD.DR.ALT

Queue naming convention:
SUBS.<Station_ID> for Queues local to subscriber for the Station_ID identified at the subscriber.
NASD.<Station_ID> for Queues local to Nasdaq for the Station_ID identified at Nasdaq.
Station_ID is a Station name or Device name (six characters maximum).

The Error Queue Name ERRORQ.<SUBS> should be defined as a local queue at the subscriber end.

Queue Manager Naming convention:
NSSUBS (Nasdaq QMAN), SUBSNS (Subscriber QMAN). – [for platforms other than IBM]

If the subscribers use shared Queue Managers, and when QMANs are limited to four-character names, the subscriber should specify this in the “Nasdaq MQ Definitions Form for Subscribers.” In the case of a duplicate, Nasdaq shall notify the subscriber.

Transmit Queue Name:
The Transmit Queue Name should be the same as the Remote Queue Manager Name.

<SUBS> – 4 Character subscriber code – the MMID (Market Maker ID).

The Nasdaq MQ Definitions Form for Subscribers, shown in Appendix B of this document, helps Nasdaq Configuration Management to avoid duplicates.

Testing Procedure

The testing procedure for every subscriber coming in through CTCI-MQ interface is as follows.

- Fill in the Nasdaq CTCI-MQ Form for the test environment.
- Set up MQ configuration definitions and remote MQ-API calls and authentication channel exit routines.
- Establish a point-to-point link with Nasdaq, activate Queue Managers and Channels, and establish channel links.
- Test Authentication procedures and data-flow through just one link.
- Establish the complete CTCI-MQ environment topology and test the recovery procedures.
- Test the CTCI-MQ switch application in its entirety.
- Daily and periodic maintenance procedures have to be tested as well.

Maintenance of CTCI-MQ Entities

- Nasdaq will bring Channels down at the end of every market day.
- All local and transmit queues should be cleared at the end of every day at both the Nasdaq and subscriber end.
- It is preferable to end (endmqm) both the queue managers of the MQ topology and restart them (strmqm) at the beginning of every market day.

Since each subscriber MQ application resides on different platforms, the established maintenance procedures have to be tested as part of the testing procedure.

Appendix A

Sample MQ Definitions at Nasdaq and Remote Subscriber

@NASDAQ (NPSUBS)

```
DEFINE QREMOTE(SUBS.CON001)  +
     RNAME(SUBS.CON001) +
     RQMNAME(SUBSNS) +
     XMITQ(SUBSNS)
            :
            :

DEFINE QREMOTE(ERRORP.SUBS)  +
     RNAME(ERRORP.SUBS) +
     RQMNAME(SUBSNS) +
     XMITQ(SUBSNS)

DEFINE QLOCAL(SUBSNS) +
     DESCR('TO.SUBS') +
     PUT(ENABLED) +
     GET(ENABLED) +
     NOTRIGGER +
     MAXDEPTH(9999999) +
     USAGE(XMITQ)

DEFINE CHANNEL(NASD.SUBS.PRI) +
     CHLTYPE(SVR) +
     XMITQ(SUBSNS) +
     BATCHSZ(50) +
     HBINT(10) +
     SHORTTMR(2) +
     SHORTRTY(5) +
     DISCINT(80000) +
     SEQWRAP(999999999) +
     TRPTYPE(TCP)

DEFINE CHANNEL(SUBS.NASD.PRI) +
     CHLTYPE(RCVR) +
     BATCHSZ(50) +
     SEQWRAP(999999999) +
     HBINT(10) +
     SCYEXIT(AUTH('CHANNELEXIT')) +
     MCAUSER('melon:harbor') +
     TRPTYPE(TCP)
```

@SUBSCRIBER (SUBSNS)

```
DEFINE QREMOTE(NASD.STN001)  +
     RNAME(NASD.STN001) +
     RQMNAME(NSSUBS) +
     XMITQ(NSSUBS)
            :
            :

DEFINE QLOCAL(NSSUBS) +
     DESCR('TO.NASDAQ') +
     PUT(ENABLED) +
     GET(ENABLED) +
     NOTRIGGER +
     MAXDEPTH(9999999) +
     USAGE(XMITQ)

DEFINE CHANNEL(SUBS.NASD.PRI) +
     CHLTYPE(SDR) +
     CONNAME('ppp.ppp.ppp.ppp(40500)') +
     XMITQ(NSSUBS) +
     DISCINT(80000) +
     BATCHSZ(50) +
     HBINT(10) +
     SHORTTMR(2) +
     SHORTRTY(5) +
     SEQWRAP(999999999) +
     SCYEXIT(AUTH('CHANNELEXIT')) +
     MCAUSER('melon:harbor') +
     TRPTYPE(TCP)

DEFINE CHANNEL(NASD.SUBS.PRI) +
     CHLTYPE(RQSTR) +
     CONNAME('ppp.ppp.ppp.ppp(40500)') +
     BATCHSZ(50) +
     SEQWRAP(999999999) +
     HBINT(10) +
     TRPTYPE(TCP)
```

AUTH – Name of Security Program

Continued

@ NASDAQ(NSSUBS Contd.)

```
DEFINE CHANNEL(NASD.SUBS.ALT) +
      CHLTYPE(SVR) +
      XMITQ(SUBSNS) +
      BATCHSZ(50) +
      HBINT(10) +
      SHORTTMR(2) +
      SHORTRTY(5) +
      DISCINT(80000) +
      SEQWRAP(999999999) +
      TRPTYPE(TCP)

DEFINE CHANNEL(SUBS.NASD.ALT) +
      CHLTYPE(RCVR) +
      BATCHSZ(50) +
      SEQWRAP(999999999) +
      HBINT(10) +
      SCYEXIT(AUTH('CHANNELEXIT')) +
      MCAUSER('melon:harbor') +
      TRPTYPE(TCP)

DEFINE QLOCAL(NASD.STN001) +
      MAXDEPTH(250000)
                  :
                  :
                  :
```

Note: A similar set of definitions must be created at the Disaster Recovery site Queue Manager for all but the channel names that would be different. (See Naming conventions section).

@SUBSCRIBER (SUBSNS Contd.)

```
DEFINE CHANNEL(SUBS.NASD.ALT) +
      CHLTYPE(SDR) +
      CONNAME('qqq.qqq.qqq.qqq(40500)') +
      XMITQ(NSSUBS) +
      DISCINT(80000) +
      BATCHSZ(50) +
      HBINT(10) +
      SHORTTMR(2) +
      SHORTRTY(5) +
      SEQWRAP(999999999) +
      SCYEXIT(AUTH('CHANNELEXIT')) +
      MCAUSER('melon:harbor') +
      TRPTYPE(TCP)

 DEFINE CHANNEL(NASD.SUBS.ALT) +
      CHLTYPE(RQSTR) +
      CONNAME('qqq.qqq.qqq.qqq(40500))') +
      BATCHSZ(50) +
      SEQWRAP(999999999) +
      HBINT(10) +
      TRPTYPE(TCP)

DEFINE QLOCAL(ERRORP.SUBS) +
      MAXDEPTH(250000)

 DEFINE QLOCAL(SUBS.CON001) +
      MAXDEPTH(250000)

                  :
                  :
                  :
```

Four more channel definitions similar to the four mentioned above, with the exception of the names and IP addresses, have to defined for links to Nasdaq's disaster recovery site. All defined at the same Queue Manager.

Appendix B
Nasdaq MQ Definitions Form for Subscribers

CTC-MQM RESOURCE DEFINITION:

Firm Name :
MMID (Subscriber ID) :
Contact: Name:_____
Tel:_____
e-mail:_____

Environment

Test/Production/Disaster Recovery(T/P/D):

NASDAQ Defines

Primary TCPIP address(Port #) :
Alternate TCPIP address(Port #):

Queue Manager Name :

Channel definitions:

Primary Server Channel Name :
Primary Receiver Channel Name:
Alternate Server Channel Name:
Alternate Receiver Channel Name :

(List all local queue names here)

Contd.

Firm Defines

Queue Manager Name :

Primary TCPIP address(Port #) :
Alternate TCPIP address(Port #) :

Channel definitions:

Primary Sender Channel Name:
Primary Requester Channel Name:
Alternate Sender Channel Name:
Alternate Requester Channel Name:

(List all local queue names here)

Appendix C
Sample Subscriber Channel Security Exit Source Code

```
 /*********************************************************************/
/* Function: SMQAUTC         Nasdaq Stock Market Inc                 */
/* SMQAUTC is a Channel exit controlling function (Sender)           */
/* Code used at the Subscriber - DEAN (DEANWITTER)                   */
/*********************************************************************/
#include "cmqc.h"
#include "cmqxc.h"
#include "mqsvmht.h"

char SEC_CTCI[11];
char MCA_USER[5];

struct {
 char ctci_mca_name[12];
 char ctci_password[10];
} CTCI_NAM;

MQ_CHANNEL_EXIT CHANNELEXIT;

/* MQStart() */

short
Read_Answer_NAM(char *Grecv, char *Hrecv, long Hlen)
{
 memcpy(Grecv, Hrecv, Hlen);
 strcpy(SEC_CTCI, "_PASSWORD_\0");
 strcpy(MCA_USER, "NASD\0");
 if (strncmp(Grecv, MCA_USER, 4) == 0) {
   if (strncmp(Grecv+12, SEC_CTCI, 10) == 0) return(0);
   else return(1);
 }
 else return(1);
}

void
CTCISecurityExit(PMQVOID pChannelExitParms,
                 PMQVOID pChannelDefinition,
                 PMQLONG pDataLength,
                 PMQLONG pAgentBufferLength,
                 PMQVOID pAgentBuffer,
                 PMQLONG pExitBufferLength,
                 PMQPTR pExitBufferAddr)
{
 char pBuff[22];
 char *pagBuf = ( char * ) pAgentBuffer;
 PMQCXP pChlExParms = ( PMQCXP ) pChannelExitParms;
```

```
PMQCD pChDef = ( PMQCD ) pChannelDefinition;
 short buflen = 22;
 short stat;


 switch ( pChlExParms->ExitReason )
 {
  case MQXR_INIT:
    pChlExParms->ExitResponse2 = 0L;
    pChlExParms->Feedback      = 0L;
    pChlExParms->ExitResponse  = MQXCC_OK;
    break;

  case MQXR_TERM:
    pChlExParms->ExitResponse  = MQXCC_OK;
    break;

  case MQXR_INIT_SEC:
    switch ( pChDef->ChannelType )
    {
      case MQCHT_SENDER:
       /* Write the sending NAM message */
       *pExitBufferAddr = pBuff;
       *pExitBufferLength = buflen;
       sprintf(CTCI_NAM.ctci_mca_name, "MMID\0";
       strcpy(SEC_CTCI, "_PASSWORD_\0");
       memcpy(CTCI_NAM.ctci_password, SEC_CTCI, 10);
       memcpy(pBuff, (char *)&CTCI_NAM, buflen);
       *pDataLength = buflen;
       pChlExParms->ExitResponse  = MQXCC_SEND_SEC_MSG;
       pChlExParms->ExitResponse2 = MQXR2_USE_EXIT_BUFFER;
      break;

      default:
      break;
    }
    break;

  case MQXR_SEC_MSG:
     switch ( pChDef->ChannelType )
        {
        case MQCHT_SENDER:
          /* Write your Read_Answer_NAM */
          stat = Read_Answer_NAM((char *)&CTCI_NAM, pagBuf, buflen);
          if (stat != 0) {
             pChlExParms->ExitResponse = MQXCC_CLOSE_CHANNEL;
             pChlExParms->ExitResponse2 = MQXR2_USE_AGENT_BUFFER;

           }  else {
                pChlExParms->ExitResponse = MQXCC_OK;
                pChlExParms->ExitResponse2 = 0L;
```

```
        }
       break;

       default:
       break;
     }
   break;

   default:
   break;
 }
}

void
MQENTRY CHANNELEXIT(
PMQVOID pChannelExitParms,    /* Channel exit parameter block */
PMQVOID pChannelDefinition,   /* Channel definition           */
PMQLONG pDataLength,          /* Length of data               */
PMQLONG pAgentBufferLength,   /* Length of agent buffer       */
PMQVOID pAgentBuffer,         /* Agent buffer                 */
PMQLONG pExitBufferLength,    /* Length of exit buffer        */
PMQPTR pExitBufferAddr)       /* Address of exit buffer       */
{
PMQCXP pCEP = pChannelExitParms;
PMQCD pCD = pChannelDefinition;
MQLONG ExitId = pCEP->ExitId;
/* MQLONG ExitReason = pCEP->ExitReason; */
pCEP->ExitResponse = MQXCC_OK ;

/* Call the handling function according to the ExitId  */
/* By default, there are no exits. If there are, then  */
/* this function will have been replaced by a bind     */

switch (ExitId)
{
case MQXT_CHANNEL_SEC_EXIT:
if (strlen(pCD->SecurityExit) == 0)
{
 pCEP->ExitResponse = MQXCC_SUPPRESS_FUNCTION;
}
else
{
 /* Call the security exit function here */
    CTCISecurityExit(pChannelExitParms,


                     pChannelDefinition,
                     pDataLength,
                     pAgentBufferLength,
                     pAgentBuffer,
                     pExitBufferLength,
```

```
                    pExitBufferAddr);
}
break;

default:
/* if the exit isn't recognized, stop it from being called again */
pCEP->ExitResponse = MQXCC_SUPPRESS_EXIT ;
} /* switch */
return;
}

/*****************************************************************/
/* END OF smqautc                                                */
/*****************************************************************/
```

Appendix D
References

Computer-to-Computer Interface Utilizing the NASDMS Switch

Appendix E
CTCI-MQ Errors

These error messages are CTCI-MQ line-handler application-specific error messages returned from Nasdaq to the ERRORQ.<SUBS> queue at the subscriber. The format is usually a nine-byte code followed by a 55-byte decode string explaining the cause or reason for the error. If some data has to be returned to the subscriber, this data will follow the 64-byte header. The maximum size of this message is 1100 bytes (64+1036).

CTCI-MQ01 – CTCI-MQ Version Error
CTCI-MQ02 – Message Length Mismatch

Appendix F
Glossary

API: WebSphere MQ Application Programming Interface.

API calls: The WebSphere MQ connect, disconnect, open, put and get calls used by the API.

Channel Security Exit: A user-written program that can be entered from one of a defined number of places during channel security operation.

Channel Specific Entities:
BATCHSZ – Batch Size; the maximum number of messages to be sent before a syncpoint is taken. The batch size does not affect the way the channel transfers messages.
MCAUSER – MCA user identifier.
HBINT – Heartbeat Interval; the time in seconds that is to elapse between heartbeat flows.
SHORTTMR - The approximate interval in seconds that the channel is to wait before trying to re-establish connection during the short retry mode.
SHORTRTY - Specifies the maximum number of times that the channel is to try allocating a session to its partner during the short retry mode.
LONGTMR - The approximate interval in seconds that the channel is to wait before retrying to establish connection during the long retry mode.
LONGRTY - Specifies the maximum number of times that the channel is to try allocating a session to its partner during the long retry mode.

Connection: Transmission path (including all equipment) between a sender and receiver.

CSA: Channel Security Authenticator.

CCSID: The Coded Character Set ID (CCSID), also called a Code Page, is a number assigned to a particular method of representing data.

CCSID 819: This is the ISO 8859-1 standard Western European ASCII code page.

CTCI: Computer to Computer Interface; the facility that allows subscribers to send and receive Nasdaq securities transactions from/to a subscriber host computer.

DR: Disaster Recovery.

FIFO: First In First Out – no special priorities. A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time.

HHMMSSCC: Method of formatting the time day in hours, minutes, seconds and hundredths of seconds. HH = Hours in military time (00-23), MM = Minutes (00-59), SS = Seconds (00-59) CC = hundredths of seconds (00-99). For example: 9 a.m. is 09000000, 1:35PM is 13350000, 35, and 98/100 seconds past midnight is 00003598.

IP Address: The IP address along with the Well Known Port is used to establish a connection to

Nasdaq in order to send and receive CTCI MQ Messages. Nasdaq assigns the IP address.

Listener: In WebSphere MQ distributed queuing, a program that monitors for incoming network connections.

Local Queue: A queue that belongs to the local queue manager. A local queue can contain a list of messages waiting to be processed. Contrast with remote queue.

MCA: Message channel agent. A program that transmits prepared messages from a transmission queue to a communication link, or from a communication link to a destination queue.

MCAUSER: MCA user identifier defined at the channel definitions.

Message Descriptor: Control information describing the message format and presentation that is carried as part of a WebSphere MQ message. The format of the message descriptor is defined by the MQMD structure.

MMID: Market Maker Identifier also referred to here as the subscriber ID.

NAM: Nasdaq Authentication Message – Used between the Channel Security Authenticators at Nasdaq and the subscriber to avoid security breach.

Nulls: The value of the lowest occurrence in the ASCII character set (Binary zero).

Persistence: A mechanism by which messages survive a restart of the queue manager.

Queue Manager: A system program that provides queuing services to applications. It provides an application programming interface so that programs can access messages on the queues that the queue manager owns.

Remote Queue: A queue belonging to a remote queue manager. Programs can put messages on remote queues, but they cannot get messages from remote queues. Contrast with local queue.

STANDARD BINDINGS: The Option in MQ connect that guarantees queue manager integrity. MQCONNX (like MQCONN) implies two logical threads where the WebSphere MQ application and the local queue manager agent run in separate processes. The WebSphere MQ application performs the WebSphere MQ operation and the local queue manager agent performs the application operation. This is defined by the MQCNO_STANDARD_BINDING option on the MQCONNX call. This default maintains the integrity of the queue manager (that is, it makes the queue manager immune to errant programs).

TCP/IP: Transmission Control Process/Internet Protocol, a method that allows communications to take place between heterogeneous systems in a multi-network environment (Internet).

Transmission Queue: A local queue on which prepared messages destined for a remote queue manager are temporarily stored.

WebSphere MQ: A family of IBM-licensed programs that provides message queuing services.

Well Known Port: Identifier (5 bytes) combined with an IP Address to form a socket (connection) name. Will be assigned by Nasdaq.