MQtt_X ‘C’ Language API

User's Manual

[Version 1.01]

Warranty

All products manufactured by ICP DAS Inc. are warranted against defective materials for a period of one year from the date of delivery to the original purchaser.

Warning

ICP DAS Inc. assumes no liability for damages consequent to the use of this product. ICP DAS Inc. reserves the right to change this manual at any time without notice. The information furnished by ICP DAS Inc. is believed to be accurate and reliable. However, no responsibility is assumed by ICP DAS Inc. for its use, or for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Trademark

The names used for identification only maybe registered trademarks of their respective companies.

License

The user can use, modify and backup this software on a single machine. The user may not reproduce, transfer or distribute this software, or any copy, in whole or in part.

Table of Contents

1 Summary of Amendments

Date	Changes
25 December 2009	Initial release • Version 1.01

2 Preface

This SDK provides a C language implementation of the MQtt_X protocol. The code is supplied pre-built for Borland C++ on MiniOS7 and is supplied with MQtt_X.lib and MQtt_X.h to compile the code on uPAC-7186.

MQtt_X – MQ Telemetry Transport for uPAC-7186

3 C language of MQtt_X API and programming model

The MQtt_X protocol is built into a shared library on the MiniOS7 and X-Server platforms (7186EL.lib and TCP_DM32.lib), although the source may be compiled and linked as appropriate for the development platform.

The API provides functions communicating with IBM micro broker, such as connecting, disconnecting, publishing, subscribing, unsubscribing, receiving publications and some additional helper functions. The API is designed to be non-blocking, so functions will return before an operation, such as publish or subscribe has completed. The status of these operations can be queried using the message identifier returned by the API.

3.1 Programming model

The MQtt_X C source code is compiled in a single thread of execution. The single threaded implementation allows the code to be quickly compiled for evaluation on a platform. **Ethernet_Init** and **MQtt_Init** must be used in uPAC-7186 programming without X-Server architecture. **Ethernet_Init** can be ignored in uPAC-7186 programming with X-Server architecture.

3.1.1 Connecting and disconnecting

When **MQtt_MQIsdpConnect** returns MQISDP_OK this indicates that a connect message has been successfully built ready to send to the MQtt micro broker. The protocol is in a state of CONNECTING.
The status of the connection between the device and the WMQTT broker, which can be:

- MQISDP_CONNECTING - a connection with the broker is being requested, but no response has been received yet.
- MQISDP_CONNECTED – a response to a connect request has been received, so the protocol is now connected and ready to send data to the broker.
- MQISDP_DISCONNECTED – a TCP/IP error has occurred and the protocol is trying to reconnect to the broker.
- MQISDP_CONNECTION_BROKEN – the protocol has been unable to connect to the broker and all retries have been exhausted, as determined by the RetryInt parameter of **MQtt_MQIsdpconnect**.

MQtt_MQIsdpDisconnect must be called to disconnect the application, even if the connection between the device and the broker is in state MQISDP_CONNECTION_BROKEN. **MQtt_MQIsdpDisconnect** frees up resources as well as closing the TCP/IP connection.

3.1.2 Sending data

To send data to the broker the application must use **MQtt_MQIsdpPublish**. Every piece of data published must be associated with a topic.

Data can be published no matter what state the connection to the broker is in, but

applications need to be aware that if the protocol fails to reconnect to the broker after a connection error then the messages will not get delivered. In the event of an error applications can use **MQtt_MQIsdp_getMsgStatus** to find out what messages have been delivered.

3.1.3Receiving data

To receive data an application must first tell the broker what data it is interested in receiving. This can be done using **MQtt_MQIsdpSubscribe** to specify all topics that the application is interested in.

MQtt_MQIsdpRcvPub can be used to receive data. A timeout can be specified, so that the API blocks until a message arrives, or the timeout expires. **MQtt_MQIsdpRcvPub** may return:

- MQISDP_NO_PUBS_AVAILABLE – if there are no publications to receive.
- MQISDP_PUBS_AVAILABLE – if a publication is successfully received and there are more publications available
- MQISDP_OK – if a publication is successfully received and there are no more publications available.
- MQISDP_DATA_TRUNCATED – if there is a message to receive, but the buffer supplied by the application is not large enough.

When an application is no longer interested in receiving data for certain topics it can call **MQtt_MQIsdpUnsubscribe** specifying all topics for which it no longer wishes to receive data.

The MQISDP_CLEAN_START flag has an affect on subscriptions active within the broker.

If the flag is not specified when connecting then the application must explicitly unsubscribe from all topics, otherwise subscriptions will remain active within the broker even after the application has disconnected. Data will be queued up to send to the application next time it connects.

If the flag is specified then the micro broker will remove any active subscriptions and outstanding messages when the application disconnects (cleanly or otherwise e.g. a TCP/IP error).

4 MQtt_X 'C' language API

4.1 Ethernet Init

int Ethernet_Init(void)

Returns:

0: ok.
-1: function "llip" error
-2: function "Ninit" error
-3: function "Portinit" error

4.2 MQtt Init

int MQtt_Init(int Broker_ID)

Inputs :

Broker_ID; 0 ~ 3

Returns:

0: ok.
1: Initiation error
2: Certification error

4.3 MQtt Close

int MQtt_Close(int Broker_ID)

Inputs :

Broker_ID; 0 ~ 3

Returns:

- Return code:
  MQISDP_OK

4.4 Connect

```
int MQtt_MQIsdpConnect (int Broker_ID,
                        PUBPARMS *ppp,
                        MQISDPTI *pApiTaskInfo,
                        int RetryInt,
                        int KeepAlive,
                        int CleanStart)
```

Inputs:

Broker_ID – Broker ID; 0 ~ 3
ppp – Address of a PUBPARMS
pApiTaskInfo – Address of task information
RetryInt – retry times
KeepAlive – Keep alive time
CleanStart – Whether clean start; TRUE = 1; FALSE = 0

Returns:

- Return code:

MQISDP_OK
MQISDP_NO_WILL_TOPIC
MQISDP_ALREADY_CONNECTED
MQISDP_HOSTNAME_NOT_FOUND
MQISDP_PERSISTENCE_FAILED
MQISDP_DATA_TOO_BIG

.

If return code is MQISDP_OK, a valid connection handle is returned otherwise connection handle is set to MQISDP_INV_CONN_HANDLE

PUBPARMS:

Field	Data Type	Usage
clientId	char[24]	A NULL terminated string up to MQISDP_CLIENT_ID_LENGTH (23) characters in length uniquely identifying the application to the MQIsdp broker.
pBroker	char[32]	The hostname or dotted decimal IP address of the broker.
port	int	The TCP/IP port number of the broker.
topic	char[100]	The topic to be associated with the data being published
qos	int	Optional parameter. The Quality of Service at which to deliver the publication – 0, 1 or 2 • A QoS of 0 denotes that the publisher and broker attempt one-time delivery of the message but do not take steps above and beyond those provided by TCP/IP to ensure message delivery. This level is sometimes called fire and forget because the message is sent to its destination without verification of receipt. • A QoS setting of 1 specifies that the message is ensured to be delivered to the broker; however, it may be delivered more than once. • A QoS value of 2 instructs MQtt to deliver the message once and only once.
retain	int	Optional parameter. Should the publication be retained by the broker – 1(yes) or 0(no).
lwtTopic	char[32]	The last Will topic name
lwtQos	int	The last Will topic QoS • A QoS of 0 denotes that the publisher and broker attempt one-time delivery of the message but do not take steps above and beyond those provided by TCP/IP to ensure message delivery. This level is sometimes called fire and forget because the message is sent to its destination without verification of receipt. • A QoS setting of 1 specifies that the

		message is ensured to be delivered to the broker; however, it may be delivered more than once. • A QoS value of 2 instructs MQtt to deliver the message once and only once.
lwtRetain	int	The last Will topic retain – 1(yes) or 0(no).
lwtData	char[32]	The last Will topic data
debug	int	Debug mode – always 0(no).
dataArg	int	Data Arguement – always 0(no).
hConn	MQISDPCH	A valid connection handle
lastSentMsg	MQISDPMH	Address of last sent message handle

4.5 Disconnect

int MQtt_MQIsdpDisconnect(int Broker_ID,
PUBPARMS *ppp)

Inputs:

Broker_ID - Broker ID; 0 ~ 3
ppp - Address of a PUBPARMS

Returns:

- Return code:
  MQISDP_OK
  MQISDP_PERSISTENCE_FAILED
  MQISDP_CONN_HANDLE_ERROR

PUBPARMS:

Field	Data Type	Usage
clientId	char[24]	A NULL terminated string up to MQISDP_CLIENT_ID_LENGTH (23) characters in length uniquely identifying the application to the MQIsdp broker.
pBroker	char[32]	The hostname or dotted decimal IP address of the broker.
port	int	The TCP/IP port number of the broker.
topic	char[100]	The topic to be associated with the data being published
qos	int	Optional parameter. The Quality of Service at which to deliver the publication – 0, 1 or 2 • A QoS of 0 denotes that the publisher and broker attempt one-time delivery of the message but do not take steps above and beyond those provided by TCP/IP to ensure message delivery. This level is sometimes called fire and forget because the message is sent to its destination without verification of receipt. • A QoS setting of 1 specifies that the message is ensured to be delivered to the broker; however, it may be delivered more

		than once. • A QoS value of 2 instructs MQtt to deliver the message once and only once.
retain	int	Optional parameter. Should the publication be retained by the broker – 1(yes) or 0(no).
lwtTopic	char[32]	The last Will topic name
lwtQos	int	The last Will topic QoS • A QoS of 0 denotes that the publisher and broker attempt one-time delivery of the message but do not take steps above and beyond those provided by TCP/IP to ensure message delivery. This level is sometimes called fire and forget because the message is sent to its destination without verification of receipt. • A QoS setting of 1 specifies that the message is ensured to be delivered to the broker; however, it may be delivered more than once. • A QoS value of 2 instructs MQtt to deliver the message once and only once.
lwtRetain	int	The last Will topic retain – 1(yes) or 0(no).
lwtData	char[32]	The last Will topic data
debug	int	Debug mode – always 0(no).
dataArg	int	Data Arguement – always 0(no).
hConn	MQISDPCH	A valid connection handle
lastSentMsg	MQISDPMH	Address of last sent message handle

4.6 Publish

```
int MQtt_MQIsdpPublish(int Broker_ID,
                       PUBPARMS *ppp,
                       char *pData,
                       int dataLength )
```

Inputs:

Broker_ID - Broker ID; 0 ~ 3
ppp - Address of a PUBPARMS
pData – Address of publish data
dataLength – Length of publish data

Returns:

- Return code:
  MQISDP_OK
  MQISDP_CONN_HANDLE_ERROR
  MQISDP_Q_FULL
  MQISDP_PERSISTENCE_FAILED
  MQISDP_DATA_TOO_BIG
  MQISDP_CONNECTION_BROKEN

MQISDP_INVALID_STRUC_LENGTH

PUBPARMS:

Field	Data Type	Usage
clientId	char[24]	A NULL terminated string up to MQISDP_CLIENT_ID_LENGTH (23) characters in length uniquely identifying the application to the MQIsdp broker.
pBroker	char[32]	The hostname or dotted decimal IP address of the broker.
port	int	The TCP/IP port number of the broker.
topic	char[100]	The topic to be associated with the data being published
qos	int	Optional parameter. The Quality of Service at which to deliver the publication – 0, 1 or 2 • A QoS of 0 denotes that the publisher and broker attempt one-time delivery of the message but do not take steps above and beyond those provided by TCP/IP to ensure message delivery. This level is sometimes called fire and forget because the message is sent to its destination without verification of receipt. • A QoS setting of 1 specifies that the message is ensured to be delivered to the broker; however, it may be delivered more than once. • A QoS value of 2 instructs MQtt to deliver the message once and only once.
retain	int	Optional parameter. Should the publication be retained by the broker – 1(yes) or 0(no).
lwtTopic	char[32]	The last Will topic name
lwtQos	int	The last Will topic QoS • A QoS of 0 denotes that the publisher and broker attempt one-time delivery of the message but do not take steps above and beyond those provided by TCP/IP to ensure message delivery. This level is sometimes called fire and forget because the message is sent to its destination without verification of receipt. • A QoS setting of 1 specifies that the message is ensured to be delivered to the broker; however, it may be delivered more than once. • A QoS value of 2 instructs MQtt to deliver the message once and only once.
lwtRetain	int	The last Will topic retain – 1(yes) or 0(no).

lwtData	char[32]	The last Will topic data
debug	int	Debug mode – always 0(no).
dataArg	int	Data Arguement – always 0(no).
hConn	MQISDPCH	A valid connection handle
lastSentMsg	MQISDPMH	Address of last sent message handle

4.7 Subscribe

int MQtt_MQIsdpSubscribe(int Broker_ID,
SUBPARMS *ppp)

Inputs:

Broker_ID - Broker ID; 0 ~ 3
ppp - Address of a SUBPARMS

Returns:

- Return code:
  MQISDP_CONN_HANDLE_ERROR
  MQISDP_Q_FULL
  MQISDP_PERSISTENCE_FAILED
  MQISDP_DATA_TOO_BIG
  MQISDP_CONNECTION_BROKEN
  MQISDP_INVALID_STRUC_LENGTH

SUBPARMS:

Field	Data Type	Usage
clientId	char[24]	A NULL terminated string up to MQISDP_CLIENT_ID_LENGTH (23) characters in length uniquely identifying the application to the MQIsdp broker.
pBroker	char[32]	The hostname or dotted decimal IP address of the broker.
Port	int	The TCP/IP port number of the broker.
topic	char[100]	The topic to be associated with the data being published
qos	int	Optional parameter. The Quality of Service at which to receive the publication – 0, 1 or 2 • A QoS of 0 denotes that the publisher and broker attempt one-time delivery of the message but do not take steps above and beyond those provided by TCP/IP to ensure message delivery. This level is sometimes called fire and forget because the message is sent to its destination without verification of receipt. • A QoS setting of 1 specifies that the message is ensured to be received from the broker; however, it may be received more than once. • A QoS value of 2 instructs MQtt to receive the message once and only once.

retain	int	Optional parameter. Should the publication be retained by the broker – 1(yes) or 0(no).
lwtTopic	char[32]	The last Will topic name
lwtQos	int	The last Will topic QoS • A QoS of 0 denotes that the publisher and broker attempt one-time delivery of the message but do not take steps above and beyond those provided by TCP/IP to ensure message delivery. This level is sometimes called fire and forget because the message is sent to its destination without verification of receipt. • A QoS setting of 1 specifies that the message is ensured to be received from the broker; however, it may be received more than once. • A QoS value of 2 instructs MQtt to receive the message once and only once.
lwtRetain	int	The last Will topic retain – 1(yes) or 0(no).
lwtData	char[32]	The last Will topic data
debug	int	Debug mode – always 0(no).
dataArg	int	Data Arguement – always 0(no).
hConn	MQISDPCH	A valid connection handle
lastSentMsg	MQISDPMH	Address of last sent message handle

4.8 Unsubscribe

int MQtt_MQIsdpUnsubscribe(int Broker_ID,
SUBPARMS *ppp)

Inputs:

Broker_ID - Broker ID; 0 ~ 3
ppp - Address of a SUBPARMS

Returns:

- Return code:
  MQISDP_CONN_HANDLE_ERROR
  MQISDP_Q_FULL
  MQISDP_PERSISTENCE_FAILED
  MQISDP_DATA_TOO_BIG
  MQISDP_CONNECTION_BROKEN
  MQISDP_INVALID_STRUC_LENGTH

SUBPARMS:

Field	Data Type	Usage
clientId	char[24]	A NULL terminated string up to MQISDP_CLIENT_ID_LENGTH (23) characters in length uniquely identifying the application to the MQIsdp broker.
pBroker	char[32]	The hostname or dotted decimal IP address of the broker.

Port	int	The TCP/IP port number of the broker.
topic	char[100]	The topic to be associated with the data being published
qos	int	Optional parameter. The Quality of Service at which to receive the publication – 0, 1 or 2 • A QoS of 0 denotes that the publisher and broker attempt one-time delivery of the message but do not take steps above and beyond those provided by TCP/IP to ensure message delivery. This level is sometimes called fire and forget because the message is sent to its destination without verification of receipt. • A QoS setting of 1 specifies that the message is ensured to be received from the broker; however, it may be received more than once. • A QoS value of 2 instructs MQtt to receive the message once and only once.
retain	int	Optional parameter. Should the publication be retained by the broker – 1(yes) or 0(no).
lwtTopic	char[32]	The last Will topic name
lwtQos	int	The last Will topic QoS • A QoS of 0 denotes that the publisher and broker attempt one-time delivery of the message but do not take steps above and beyond those provided by TCP/IP to ensure message delivery. This level is sometimes called fire and forget because the message is sent to its destination without verification of receipt. • A QoS setting of 1 specifies that the message is ensured to be received from the broker; however, it may be received more than once. • A QoS value of 2 instructs MQtt to receive the message once and only once.
lwtRetain	int	The last Will topic retain – 1(yes) or 0(no).
lwtData	char[32]	The last Will topic data
debug	int	Debug mode – always 0(no).
dataArg	int	Data Arguement – always 0(no).
hConn	MQISDPCH	A valid connection handle
lastSentMsg	MQISDPMH	Address of last sent message handle

4.9 Get Message Status

int MQtt_MQIsdp_getMsgStatus(int Broker_ID,
MQISDPCH hConn,
MQISDPMH hMsg)

Inputs:

Broker_ID - Broker ID; 0 ~ 3
hConn - A valid connection handle
hMsg - A valid message handle

Returns:

- Return code:
  MQISDP_CONN_HANDLE_ERROR
  MQISDP_MSG_HANDLE_ERROR
  MQISDP_DELIVERED
  MQISDP_RETRYING
  MQISDP_IN_PROGRESS

MQISDP_DELIVERED is the final state than a message can get into. A message is delivered once all the Quality of Service MQtt protocol flows are complete.

Messages with a QoS of 0 will be discarded if the TCP/IP connection is down. The application cannot query the state of a publication sent at QoS 0 because the protocol does not know if delivery is successful or not.

MQISDP_MSG_HANDLE_ERROR is returned if an invalid message handle is supplied.

4.10 Receive Publication

int MQtt_MQIsdpRcvPub(int Broker_ID,
SUBPARMS *ppp,
char *pMatchData,
long *topicLength,
long *dataLength)

Inputs:

Broker_ID - Broker ID; 0 ~ 3
ppp - Address of a SUBPARMS

Returns:

- Return code:
  MQISDP_CONN_HANDLE_ERROR
  MQISDP_MSG_HANDLE_ERROR
  MQISDP_DELIVERED
  MQISDP_RETRYING
  MQISDP_IN_PROGRESS
- pMatchData - The first topicLength bytes of this buffer contain the topic, which is followed by dataLength bytes of message data.
- topicLength - The length in bytes of the topic
- dataLength - The length in bytes of the data associated with the topic

SUBPARMS:

Field	Data Type	Usage
clientId	char[24]	A NULL terminated string up to MQISDP_CLIENT_ID_LENGTH (23) characters in length uniquely identifying the application to the MQIsdp broker.
pBroker	char[32]	The hostname or dotted decimal IP address of the broker.
Port	int	The TCP/IP port number of the broker.
topic	char[100]	The topic to be associated with the data being published
qos	int	Optional parameter. The Quality of Service at which to receive the publication – 0, 1 or 2 • A QoS of 0 denotes that the publisher and broker attempt one-time delivery of the message but do not take steps above and beyond those provided by TCP/IP to ensure message delivery. This level is sometimes called fire and forget because the message is sent to its destination without verification of receipt. • A QoS setting of 1 specifies that the message is ensured to be received from the broker; however, it may be received more than once. • A QoS value of 2 instructs MQtt to receive the message once and only once.
retain	int	Optional parameter. Should the publication be retained by the broker – 1(yes) or 0(no).
lwtTopic	char[32]	The last Will topic name
lwtQos	int	The last Will topic QoS • A QoS of 0 denotes that the publisher and broker attempt one-time delivery of the message but do not take steps above and beyond those provided by TCP/IP to ensure message delivery. This level is sometimes called fire and forget because the message is sent to its destination without verification of receipt. • A QoS setting of 1 specifies that the message is ensured to be received from the broker; however, it may be received more than once. • A QoS value of 2 instructs MQtt to receive the message once and only once.
lwtRetain	int	The last Will topic retain – 1(yes) or 0(no).
lwtData	char[32]	The last Will topic data

debug	int	Debug mode – always 0(no).
dataArg	int	Data Arguement – always 0(no).
hConn	MQISDPCH	A valid connection handle
lastSentMsg	MQISDPMH	Address of last sent message handle

4.11 Version

int MQtt_MQIsdp_version(int Broker_ID)

Inputs:

Broker_ID - Broker ID; 0 ~ 3

Returns:

- Return code:
  Version number

4.12 Return Codes

Return Code	Value	Explanation
MQISDP_OK	0	Success
MQISDP_PROTOCOL_VERSION_ERROR	1001	The WMQTT broker does not support this version of the WMQTT protocol
MQISDP_HOSTNAME_NOT_FOUND	1002	If a hostname is used in the connection parameters then this indicates that DNS resolution of the hostname failed.
MQISDP_Q_FULL	1003	The limit on the amount of data in the process of being delivered has been reached. Space will be freed up as messages are delivered or discarded.
MQISDP_FAILED	1004	Failure
MQISDP_PUBS_AVAILABLE	1005	Publications are available to be received.
MQISDP_NO_PUBS_AVAILABLE	1006	No publications are available to be received.
MQISDP_PERSISTENCE_FAILED	1007	When connecting or sending data the persistence implementation reported an error. Investigate the persistence implementation to resolve the problem.
MQISDP_CONN_HANDLE_ERROR	1008	An invalid connection handle has been specified.
MQISDP_NO_WILL_TOPIC	1010	Option MQISDP_WILL has been supplied on MQIsdp_connect, but there is no Will topic.
MQISDP_INVALID_STRUC_LENGTH	1011	An incorrect length supplied in a structure causes the send task to attempt to read beyond the end of the structure.
MQISDP_DATA_LENGTH_ERROR	1012	The data length parameter of MQIsdp_publish is less than zero.
MQISDP_DATA_TOO_BIG	1013	The data supplied is bigger than

		the WMQTT protocol can handle
MQISDP_ALREADY_CONNECTED	1014	MQIsdp_connect has been called when a connection already exists for the application.
MQISDP_CONNECTION_BROKEN	1017	All attempts by the WMQTT client to establish a connection with the WMQTT broker have been exhausted. MQIsdp_getMsgStatus, MQIsdp_status can be used to find what messages have been delivered and why the connection failed. MQIsdp_receivePub can receive waiting publications. The application must disconnect before it is able to send any more data.
MQISDP_DATA_TRUNCATED	1018	The receive buffer supplied for MQIsdp_receivePub is not big enough for the data.
MQISDP_CLIENT_ID_ERROR	1019	The WMQTT broker refused the connection attempt because of a problem with the client identifier.
MQISDP_BROKER_UNAVAILABLE	1020	The WMQTT broker has refused the connection attempt.
MQISDP_SOCKET_CLOSED	1021	The remote socket was closed unexpectedly terminating communications.
MQISDP_OUT_OF_MEMORY	1022	No more memory can be allocated for handling the API call.
	1031	Certification error.

5 Sample applications

5.1 One broker communication without X-Server

This demo shows how to use MQtt_X library in 7186.

Step 1: Initiate the controller.

```
//Step1. Initiate the controller.
   InitLib();
   InstallCom1(115200, 8, 0, 1);
```

Step 2: Initiate the Ethernet adapter.

```
//Step2. Initiate the Ethernet adapter.
   iRet=Ethernet_Init();
   if(iRet==NoError)
      printCom1("Inint Ethernet ok.\n\r");
   else
      printCom1("Inint Ethernet error.\n\r");
```

Step 3: Initiate MQtt client.

```
//Step3. Initiate MQtt client.
         iRet = MQtt_Init(0);
   if(iRet!=0)
   {
      // Initial MQtt_X library error.
                printCom1("Initial MQtt_X library error.\n\r");
   }
   else
   {
      // Initial MQtt_X library ok.
                printCom1("Initial MQtt_X library OK.\n\r");
   }
```

Step 4: Connect MQtt client to miro broker(IP:192.168.1.91).

```
//Step4. Connect MQtt client to miro broker(IP:192.168.1.91).
         pubParms.port = 1883;
         subParms.port = 1888;
         sprintf(pubParms.pBroker,"%s","192.168.1.91");
         sprintf(subParms.pBroker,"%s","192.168.1.91");
                                          :
                                          :
         iRet = MQtt_MQIsdpConnect(0, &pubParms, pApiTaskParms, 2, 5, 1);
                                          :
```

Step 5: Subscribe Topic

```
//Step5. Subscribe Topic.
        subParms.hConn = pubParms.hConn;
        subParms.qos = 1;
        subParms.timeout = 50;
        //Sub topic 1
        sprintf(subParms.topic,"%s","uPAC101/XBoard/DO/ch0");
        iRet = MQtt_MQIsdpSubscribe(0, &subParms );
        if(iRet!=0)
        {
    // MQtt_MQIsdpSubscribe 1 error.
            printCom1("MQtt_MQIsdpSubscribe 1 error.\n\r");
        }
        else
        {
    // MQtt_MQIsdpSubscribe 1 OK.
            printCom1("MQtt_MQIsdpSubscribe 1 OK.\n\r");
        }
                                    :
                                    :
```

Step 6: It is a loop function which receive data published from other MQtt clients and publish its data to other MQtt clients.

```
//Step6. Begin the loop function
   for(;;)
   {
      if((GetTimeTicks()-lStart_TimeTick)>50)
      {
         lStart_TimeTick=GetTimeTicks();
//Step6.1 Recieve Publish
                                    :
                                    :
                                    :
//Step6.2 Publish
                                    :
                                    :
                                    :
        }
    }
```

Full project could be seen in **MQttX_1B.PRJ**.

5.2 One broker communication with X-Server

This demo shows how to use MQtt_X library in 7186.

Step 1: Initiate the controller.

```
//Step1. Initiate the controller.
   InitLib();
   InstallCom1(115200, 8, 0, 1);
```

Step 2: Initiate MQtt client.

```
//Step2. Initiate MQtt client.
   //======= Initiate MQtt client =======
   iRet = MQtt_Init(0);
   if(iRet!=0)
   {
      // Initial MQtt_X library error.
                printCom1("Initial MQtt_X library error.\n\r");
   }
   else
   {
      // Initial MQtt_X library ok.
                printCom1("Initial MQtt_X library OK.\n\r");
   }
   //=======End Initiate MOtt client =======
```

Step 3: Connect MQtt client to miro broker(IP:192.168.1.91).

```
//Step3. Connect MQtt client to miro broker(IP:192.168.1.91).
        pubParms.port = 1883;
        subParms.port = 1888;
        sprintf(pubParms.pBroker,"%s","192.168.1.91");
        sprintf(subParms.pBroker,"%s","192.168.1.91");
                                        :
                                        :
        iRet = MQtt_MQIsdpConnect(0, &pubParms, pApiTaskParms, 2, 5, 1);
                                        :
```

Step 4: Subscribe Topic

```
//Step4. Subscribe Topic.
        subParms.hConn = pubParms.hConn;
        subParms.qos = 1;
        subParms.timeout = 50;
        //Sub topic 1
        sprintf(subParms.topic,"%s","uPAC101/XBoard/DO/ch0");
        iRet = MQtt_MQIsdpSubscribe(0, &subParms );
                                        :
                                        :
                                        :
                                        :
```

Step 5: It is a loop function which receive data published from other MQtt clients and publish its data to other MQtt clients.

```
   if((GetTimeTicks()-lStart_TimeTick)>50)
   {
         lStart_TimeTick=GetTimeTicks();
//Step5.1 Recieve Publish
                                          :
                                          :
                                          :
                                          :
//Step5.2 Publish
                                          :
                                          :
                                          :
                                          :
   }
```

Full project could be seen in **MQttX_1X.PRJ**.

5.3 Dual brokers communication without X-Server

This demo shows how to use MQtt_X library in 7186(connect to two micro brokers).
Step 1: Initiate the controller.

```
//Step1. Initiate the controller.
   InitLib();
   InstallCom1(115200, 8, 0, 1);
```

Step 2: Initiate the Ethernet adapter.

```
//Step2. Initiate the Ethernet adapter.
   iRet=Ethernet_Init();
   if(iRet==NoError)
      printCom1("Inint Ethernet ok.\n\r");
   else
      printCom1("Inint Ethernet error.\n\r");
```

Step 3-1: Initiate MQtt client 1.

```
//Step3-1. Initiate MQtt client 1.
         iRet = MQtt_Init(0);
   if(iRet!=0)
   {
      // Initial MQtt_X library error.
                printCom1("Initial MQtt_X library error.\n\r");
   }
   else
   {
      // Initial MQtt_X library ok.
                printCom1("Initial MQtt_X library OK.\n\r");
   }
```

Step 3-2: Initiate MQtt client 2.

```
//Step3-2. Initiate MQtt client 2.
        iRet = MQtt_Init(1);
   if(iRet!=0)
   {
      // Initial MQtt_X library error.
                printCom1("Initial MQtt_X library error.\n\r");
   }
   else
   {
      // Initial MQtt_X library ok.
                printCom1("Initial MQtt_X library OK.\n\r");
   }
```

Step 4-1: Connect MQtt client to miro broker 1(IP:192.168.1.91).

```
//Step4-1. Connect MQtt client to miro broker 1(IP:192.168.1.91).
        pubParms_1.port = 1883;
        subParms_1.port = 1888;
        sprintf(pubParms_1.pBroker,"%s","192.168.1.91");
        sprintf(subParms_1.pBroker,"%s","192.168.1.91");
                                        :
                                        :
        iRet = MQtt_MQIsdpConnect(0, &pubParms_1, pApiTaskParms, 2, 5, 1);
                                        :
```

Step 4-2: Connect MQtt client to miro broker 2(IP:192.168.1.94).

```
//Step4-2. Connect MQtt client to miro broker 2(IP:192.168.1.94).
        pubParms_2.port = 1883;
        subParms_2.port = 1888;
        sprintf(pubParms_2.pBroker,"%s","192.168.1.94");
        sprintf(subParms_2.pBroker,"%s","192.168.1.94");
                                        :
                                        :
        iRet = MQtt_MQIsdpConnect(1, &pubParms_2, pApiTaskParms, 2, 5, 1);
                                        :
```

Step 5-1: Subscribe Topic to micro broker 1.

```
//Step5-1. Subscribe Topic to micro broker 1.
        subParms_1.hConn = pubParms_1.hConn;
        subParms_1.qos = 1;
        subParms_1.timeout = 50;
        //Sub topic 1
        sprintf(subParms_1.topic,"%s","uPAC101/XBoard/DO/ch0");
        iRet = MQtt_MQIsdpSubscribe(0, &subParms_1 );
                                        :
                                        :
                                        :
                                        :
```

Step 5-2: Subscribe Topic to micro broker 2.

```
//Step5-2. Subscribe Topic to micro broker 2.
        subParms_2.hConn = pubParms_2.hConn;
        subParms_2.qos = 1;
        subParms_2.timeout = 50;
        //Sub topic 1
        sprintf(subParms_2.topic,"%s","uPAC101/XBoard/DO/ch0");
        iRet = MQtt_MQIsdpSubscribe(1, &subParms_2 );
                                        :
                                        :
                                        :
                                        :
```

Step 6: It is a loop function which receive data published from other MQtt clients and publish its data to other MQtt clients via micro broker 1 and micro broker 2.

```
//Step6. Begin the loop function
   for(;;)
   {
      if((GetTimeTicks()-lStart_TimeTick)>50)
      {
         lStart_TimeTick=GetTimeTicks();
//Step6.1 Recieve Publish from micro broker 1
                                        :
                                        :
//Step6.2 Publish to micro broker 1
                                        :
                                        :
//Step6.3 Recieve Publish from micro broker 2
                                        :
                                        :
//Step6.4 Publish to micro broker 2
                                        :
                                        :
```

Full project could be seen in **MQttX_2B.PRJ**.

5.4 Dual brokers communication with X-Server

This demo shows how to use MQtt_X library in 7186.

Step 1: Initiate the controller.

```
//Step1. Initiate the controller.
   InitLib();
   InstallCom1(115200, 8, 0, 1);
```

Step 2-1: Initiate MQtt client 1.

```
//Step2-1. Initiate MQtt client 1.
   //======= Initiate MQtt client =======
   iRet = MQtt_Init(0);
                                        :
                                        :
```

Step 2-2: Initiate MQtt client 2.

```
//Step2-2. Initiate MQtt client 2.
   //======= Initiate MQtt client =======
   iRet = MQtt_Init(1);
                                        :
                                        :
```

Step 3-1: Connect MQtt client to miro broker 1(IP:192.168.1.91).

```
//Step3-1. Connect MQtt client to miro broker 1(IP:192.168.1.91).
        pubParms_1.port = 1883;
        subParms_1.port = 1888;
        sprintf(pubParms_1.pBroker,"%s","192.168.1.91");
        sprintf(subParms_1.pBroker,"%s","192.168.1.91");
                                        :
                                        :
        iRet = MQtt_MQIsdpConnect(0, &pubParms_1, pApiTaskParms, 2, 5, 1);
                                        :
```

Step 3-2: Connect MQtt client to miro broker 2(IP:192.168.1.94).

```
//Step3-2. Connect MQtt client to miro broker 1(IP:192.168.1.94).
        pubParms_2.port = 1883;
        subParms_2.port = 1888;
        sprintf(pubParms_2.pBroker,"%s","192.168.1.94");
        sprintf(subParms_2.pBroker,"%s","192.168.1.94");
                                        :
                                        :
        iRet = MQtt_MQIsdpConnect(1, &pubParms_2, pApiTaskParms, 2, 5, 1);
                                        :
```

Step 4-1: Subscribe Topic to micro broker 1.

```
//Step4-1. Subscribe Topic to micro broker 1.
        subParms_1.hConn = pubParms_1.hConn;
        subParms_1.qos = 1;
        subParms_1.timeout = 50;
        //Sub topic 1
        sprintf(subParms_1.topic,"%s","uPAC101/XBoard/DO/ch0");
        iRet = MQtt_MQIsdpSubscribe(0, &subParms_1 );
                                        :
                                        :
```

Step 4-2: Subscribe Topic to micro broker 2.

```
//Step4-2. Subscribe Topic to micro broker 2.
        subParms_2.hConn = pubParms_2.hConn;
        subParms_2.qos = 1;
        subParms_2.timeout = 50;
        //Sub topic 1
        sprintf(subParms_2.topic,"%s","uPAC101/XBoard/DO/ch0");
        iRet = MQtt_MQIsdpSubscribe(1, &subParms_2 );
                                        :
                                        :
```

Step 5: It is a loop function which receive data published from other MQtt clients and publish its data to other MQtt clients via micro broker 1 and micro broker 2.

```
if((GetTimeTicks()-lStart_TimeTick)>50)
  {
        lStart_TimeTick=GetTimeTicks();
//Step5.1 Recieve Publish from micro broker 1
                                        :
                                        :
//Step5.2 Publish to micro broker 1
                                        :
                                        :
//Step5.3 Recieve Publish from micro broker 2
                                        :
                                        :
//Step5.4 Publish to micro broker 2
                                        :
                                        :
   }
```

Full project could be seen in **MQttX_2X.PRJ**.