DOWNTOWN LINE

Contract C955

InteGRATED SUPERVISORY

CONTROL SYSTEM (iscS)

mysql advance queue library design spec

**AMENDMENT RECORDS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Rev*** | ***Date*** | ***Descriptions*** | ***Writer*** | ***Function*** |
| *1* | *17/04/2012* | First version | Ouyang Zhilin | Module design note |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Reviewed by** |  | **Approval by** |  |
| **Entity / Function** |  | **Entity / Function** |  |
| **Date & Sign.** |  | **Date & Sign.** |  |
| **Reserved to the Engineer** |  |  |  |

**Table of Contents**

[1 introduction 5](#_Toc328055288)

[1.1 General 5](#_Toc328055289)

[1.1.1 Purpose 5](#_Toc328055290)

[1.1.2 Version Control 5](#_Toc328055291)

[1.1.3 Definitions,Acronyms and Abbreviations 5](#_Toc328055292)

[1.1.4 References 5](#_Toc328055293)

[2 Motivation 5](#_Toc328055294)

[2.1 Background 5](#_Toc328055295)

[2.2 Requirements 6](#_Toc328055296)

[2.2.1 Formal Requirements 6](#_Toc328055297)

[2.2.2 Derived Requirements 6](#_Toc328055298)

[3 constraints 6](#_Toc328055299)

[3.1 Environment Constraints 6](#_Toc328055300)

[3.2 System Requirement Constraints 6](#_Toc328055301)

[4 system overview 7](#_Toc328055302)

[5 system architecture 7](#_Toc328055303)

[5.1 System Architecture Design 7](#_Toc328055304)

[5.2 Module Decomposition Design 9](#_Toc328055305)

[5.3 Design Rational 10](#_Toc328055306)

[5.4 Related cots 10](#_Toc328055307)

[5.5 Detail Design 10](#_Toc328055308)

[5.5.1 Sequence Diagram 10](#_Toc328055309)

[5.5.2 MAQ System Table Structure 12](#_Toc328055310)

[5.5.3 Interface 15](#_Toc328055311)

[5.5.4 Class Diagram 15](#_Toc328055312)

[5.5.5 Componment Diagram 15](#_Toc328055313)

[5.5.6 Dependencies 15](#_Toc328055314)

[5.5.7 Configuration Information 15](#_Toc328055315)

[5.6 Testability 15](#_Toc328055316)

[5.7 Maintainability 15](#_Toc328055317)

[5.8 Impact Analysis 16](#_Toc328055318)

[5.9 Deployment Considerations 16](#_Toc328055319)

[5.10 Flow-on Effects 16](#_Toc328055320)

[5.11 Extensibility and Reuse 16](#_Toc328055321)

[6 gui-component design information 16](#_Toc328055322)

[Appendix A: Software module design review 17](#_Toc328055323)

# introduction

## General

### Purpose

This document has been written as an overall architecture of the library for the MySQL message advance queue. It provides detail design and architecture of the message queue and enables MySQL database features the functionalities like Oracle AQ.

### Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Changes / Reason for Issue |
| 0.01 | 11 June 2011 | Ouyang.Zhilin | First Draft |
| 1.00 |  |  | Issued for Release |
|  |  |  |  |

### Definitions,Acronyms and Abbreviations

|  |  |
| --- | --- |
| Name | Description |
| AQ | Advance Queue |
| MAQ | MySQL Advance Queue |
| MAQL | MySQL Advance Queue Library |
|  |  |

### References

|  |  |  |
| --- | --- | --- |
|  | Reference | Title |
|  | [9999-P01-01-0001](I:\\STE\\Doc\\Project Documents\\C955_docs\\Database\\DbSynchDesign\\P-Product\\P01-Common\\P0101-General\\9999P01010001-System_Glossary.doc) | System Glossary |
|  |  | Software requirement |
|  |  | Software ICD |
|  |  | Software Architecture Specification |
|  |  | IEC62279 Phases |

# Motivation

This section describes the motivation of the MAQ plug-in including the background and major requirements of the sub-system.

## Background

Due to the differences of the Database Deployment between project C955 and C830, the data replication mechnism need to revise based on the new requirements.

In the existing project C830 both the local and central databases are Oracle, and therefore the data replication mechnism used Oracle built-in AQ funcation. The new project C955, however, the type of database servers will be defined as below:

* Central Database – residing at a central location accessible across a WAN. For C955 this is the OCCDB using Sun Oracle Enterprise database server.
* Local Database – associated with any physical location accessible from other local servers and central server across a WAN. For C955 this database will use MySQL database and will be considered as the Local Database for agents running on the location for the purpose of this design.

Based on the above design, MySQL database does not have the message queue feature, therefore an extension or plug-in library is needed to enable MySQL database features this capability.

## Requirements

This section is to outline the requirements for the MySQL Advance Queue (MAQ) that can be use as a basis for developing possible solution using appropriate technologies.

### Formal Requirements

The requirements of message advance queue library (MAQL) for MySQL database should follow the same capabilities of Oracle AQ such as:

1. Single-consumer & multiple-consumer Messaging Queue support
2. Message En-queue
3. Message De-queue and
4. Message Listen capabilities
5. SQL Level Access Control
6. Application Programming Interface Access like using C/C++

### Derived Requirements

# constraints

This section describes the constraints including environment and system requirement constraints .

## Environment Constraints

The library currently can only work with MySQL database system.

## System Requirement Constraints

.

# system overview

This section describes the overview of the MAQL. The library enable producer applications en-queue messages and consumer applications listen & de-queue messages. It provides database-integrated message queuing functionality, leverages the functions of the MySQL database so that messages can be stored persistently, propagated between queues on different machines and databases, and transmitted using TCP communication.

**MySQL PL/SQL browser clients**

**Producer/consumer application clients**

Producer/consumer application clients

MySQL

Advance queues





**Browse**r

**Enq**

**Deq**

Propagation

Propagation

MySQL Advance queues

Oracle Advance queues

Other database systems

Figure 1: MySQL Advance Queue System

## Message Queuing Knowledge

The section give some general knowledge about message queuing system including what’s queue, message, payload etc..

Queue

A queue is a storage space for message. There are two different types of queues in MAQ system: user queues, also known as normal queues, and exception queues. The user queue is employed for standard message processing. The exception queue is used to hold messages if attempts to retrieve the message through a dequeue options fail or if the message is not dequeued before its expiration time. You will used the MAQ SQL administrative interface to create, start, stop and drop queues.

Message

The smallest unit of work in the queue, consisting of information about how the message is to be treated(metadata) and the payload(the data suppiled by the user). The metadata or control information is used by MAQ to manage the messages. The payload information is stored in the queue and is transparent to MAQ (this is, MAQ does not try to interpret that information in any way). A message resides in only one queue.

Queue table

A queue table is a database table that holds one queue; this queue table is created when you create the queue.

Agent

An agent is a user of a queue. There are two types of agents: *producer*, who place messages in a queue (enqueuing); and *consumer*, who retrieve messages (dequeuing). Any number of producers and consumers may be accessing the queue at a given time. An agent’s name and address identify that agent. The address field is a character field of up to 128 bytes that is interpreted by the propagation module. The address will therefore have this form,

queue\_name@IP:PORT

Where *queue\_name* is the target queue which need do operations (enqueue or dequeue) on. IP:PORT is used to identify the address of the target queue.

Agents insert messages into a queue and retrieve messages from the queue by using the MAQ SQL or C/C++ operational interfaces offered by MAQL.

Recipient list

A list of one or more agents that you can construct to send/receive a message (through enqueue/dequeue operation). For message only need for local dequeue in which case leave the address to empty; for message need to propagate to recipients in which case the address should set. These addresses are of the form*[schema].queue\_name@ip:port*. If the schema is not specified, then the schema of the current user is used. The rule of the recipient lists as below:

for only one recipient: *name,address*

for more than one recipients*: name1,address1;name2,address2;....nameN,addressN*

Producer

An agent that places messages into a queue with the enqueue operation.

Consumer

An agent that retrieves messages from a queue with the dequeue operation.

Message UUID

The unique handle for a message. This value is generated by MAQ. You can use it to retrieve a specific from a queue (bypassing the UUID of dequeuing associated with the queue). You can also use this message UUID to find out about the status of a message from the underlying MAQ system tables. These message UUIDs are long, complex values like *f00530cb-c976-11e1-8791-890065fab497*

Message sequence ID

The sequences handle for a message. This value is a 64-bit unsigned integer and generated by MAQ sequence mechanism. These IDs represent the time ordering of the messages during enqueue operation as MySQL doesn’t has the facility to record the time of messages with microsecond as Oracle do. Currently for clustering nodes, the sequence IDs should be generated at each node on its own and the replication between these clustering nodes should based on SQL statement.

# system architecture

The section details the design of message advance queue for MySQL database.

## System Architecture Design

As the library is an extension for MySQL database and all the functionalities is imitated from the existing Oracle advance queue mechanism. Applications or other database system can interact with each other to exchange (Enq/Deq) messages through PL/SQL or MAQ APIs. The following figure illustrated the architecture of the library for MAQ.

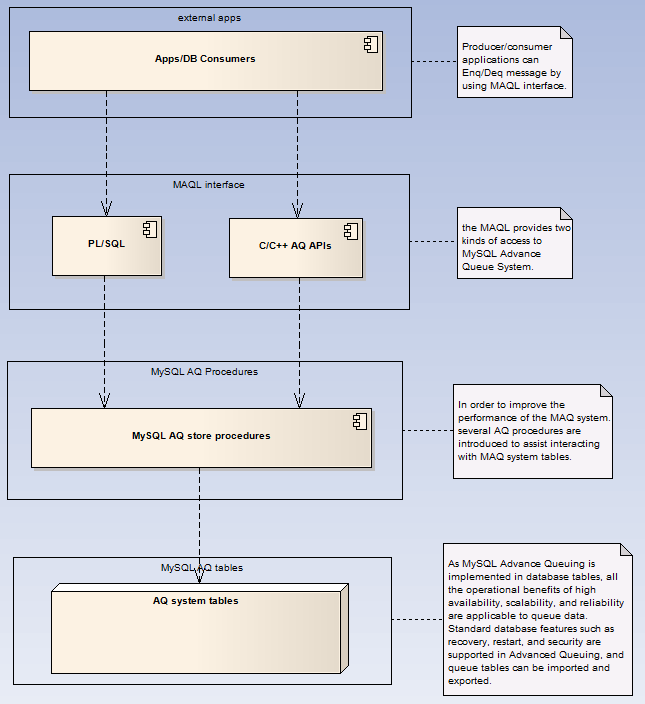


Figure 2: MySQL Advance Queue library architecture

## Module Decomposition Design

This section provides the decomposition structure of the MAQL for MySQL database system. As the library provides two kinds of access to the MAQ such as PL/SQL & MAQ APIs for C/C++, store procedures and library implementations using C/C++ needed to support this functionalities. The logic of the MAQ in both interfaces presents almost the same. Below shown the details:

1. MAQ PL/SQL and APIs, it decomposite the modules below:

▪ Enqueue

▪ Dequeue

▪ Listen

The following sections elaborate all the features of the MAQ library

* Enqueue Features

**Message enq ID** --- used for ordering of messages, each message will be given a sequence ID at the enq-time. So all the messages in the queue maintains ASC or DESC order and easy the dequeue operation later for FIFO or LIFO manner. It uses 64-bit to generate the Enq ID to avoid easily reach the maxmum value.

**Message UUID Identifier** --- the unique ID of each message for later deq specified by the message ID. It generated by the MySQL database built-in function UUID().

**Message recipient list** --- A single message can be designed to be consumed by multiple consumers. Different queues can have different recipients, and a consumer program can be a subscriber to more than one queue. You can design a single message for consumption by multiple consumers in a number of different ways. The consumers who are allowed to retrieve the message are specified as explicit recipients of the message by the user or application that enqueues the message. Every explicit recipient is an agent identified by name and address. A recipient may be specified only by its name, in which case the recipient must dequeue the message from the queue in which the message was enqueued. It may be specified by its name and an address. The address should be the name of another queue in the same database or another consumer database system (currently support Oracle & MySQL, identified by the address), in which case the message is propagated to the specified queue or database system by using the system propagation module.

* Dequeue Features

Dequeue Mode --- used for the mode how to handling the message while dequeue message from MAQ system. Three mode currently supported such as remove immediately; remove delay and remove no data.

Dequeue visibility --- used for the visibility of the message while dequeue from MAQ system. Two options currently supported such as: immediately and on commit.

Dequeue message navigation --- used for how to dequeue message from the MAQ system. Two options currently supported such as: first message and next message.

* Listen capability

Currently only the C/C++ support this listen capability.

1. MAQ system management, it decomposite the modules below:

▪ Create queue

▪ Drop queue

▪ Start queue

▪ Stop queue

▪ Add Subscriber

1. Some system tables are also involved and defined to support the MySQL advance queuing system, detailes shown below:

For each queue, there are three tables which enable the MAQ functionalities. Details below:

**Queue table** --- the main queue data table, the name convention is the queue name subfix \_T. For instance if the queue name is *AUDIT\_DATA\_QUEUE*, then the queue table name is: *AUDIT\_DATA\_QUEUE\_T*

**Dispatch table** --- the message recipient information, all the messages which ready to propagate is related to this table. The name convention of this table is prefix AQ\_ and subfix \_I of the queue name. For instance if the queue name is *AUDIT\_DATA\_QUEUE*, then the dispatch table name is: *AQ\_AUDIT\_DATA\_QUEUE\_T\_I.*

**Subscriber table** --- contains all the subscribers of the related queue. The name convention of this table is prefix AQ\_ and subfix \_S of the queue name. For instance if the queue name is *AUDIT\_DATA\_QUEUE*, then the subscriber table name is: *AQ\_AUDIT\_DATA\_QUEUE\_T\_S.*

Three MAQ system tables are introduced to manager the queuing system. Shown below:

**Exception queue table** --- contains all the exception messages encountered during all the message advance queues for processing data. The name of this table is: *AQ\_SYS\_QUEUE\_E*.

**Queue management table** --- contains information of all the queues in the database system. The name of this table is: *AQ\_SYS\_QUEUE\_M*.

**Sequence table** --- this table is designed to generate the sequence for each en-queue message and make the message in an ordering manner for ASC or DESC. It uses 64-bit for the sequence ID to avoid easily reaching the maxmum value in years. The name of this table is: ADM\_SEQUENCE.

## Design Rational

The MySQL Advance Queuing system is based on the Oracle Advance Queuing technologies.

## Related cots

The related cots of the library is only for the MAQ APIs, the cots libraries which involved to implement the MAQ APIs listed below:

mcl --- mysql connection library for interacting with MySQL database.

Ocilib --- oracle connection library for interacting with Oracle database.

## Detail Design

### Sequence Diagram

This section elaborates the sequences of the MAQ library logic including enqueue, dequeue and listen features.

● **sequence of enqueue**:

1. message producers/apps create connection to MySQL database.
2. set message queue name and message recipient list.
3. invoke *MAQEnq()* API to enqueue message.
4. call store procedure *prc\_enqueue()* to enqueue message.
5. generate the message sequence ID by using *func\_seq\_next\_number()*. Each queue has his own sequence ID.
6. get the message UUID by using system function UUID().
7. enqueue MAQ system tables such as dispatch & subscriber table.
8. enqueue to MAQ queue table for permanent storage.

Step 5 to 8 are in one transaction, any error or exception occured during these steps will cause all operations rollback and raise error message to message producer.

For the message recipient list, the following rules will be applied.

* Recipient agent with only name, and address is null, then the message is not interpreted by the system propagation. It can be dequeued by specifying the consumer name.
* Recipient agent with name & address (identfied by IP:Port), then the message is interpreted by the system propagation. Third-party consumers can not dequeued this kind of message.

The following diagram illustrated the enqueue flow:

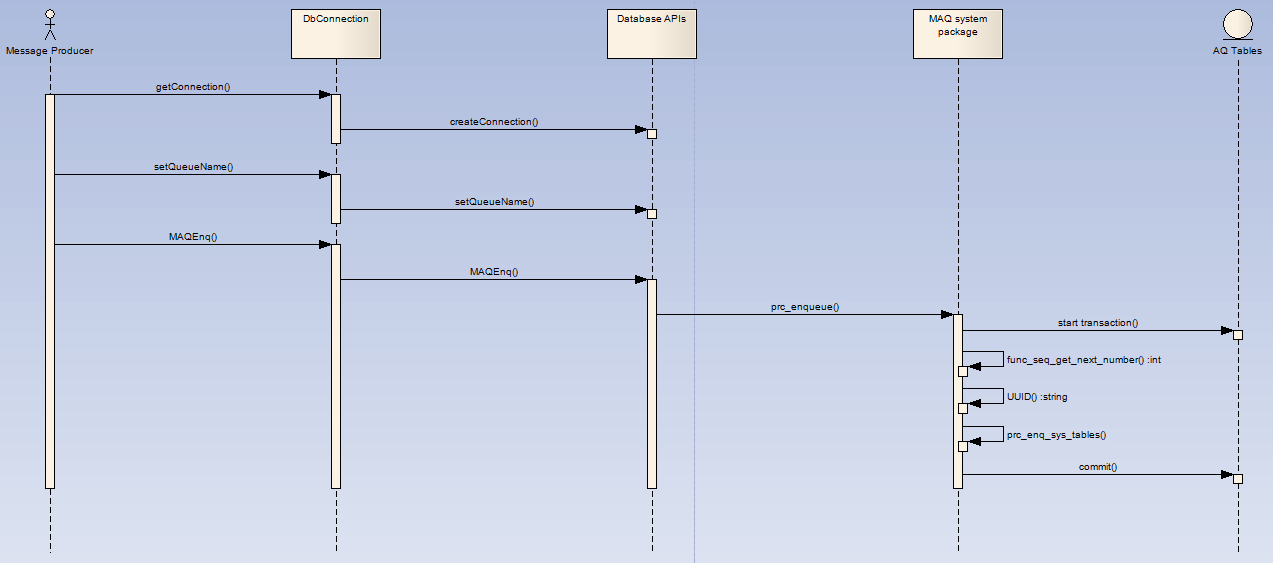


Figure 3: Enqueue sequence

● **sequence of dequeue**:

1. Message consumers/apps create connection to MySQL database.
2. set dequeue options such as message queue name, consumer name, dequeue mode, dequeue visibility and navigation.
3. invoke *MAQDeq()* or *MAQDeqArray()* API to enqueue message. As the MySQL database characteristic limitation for procedure does not support output parameter for result cursor. Currently only through C/C++ APIs supports dequeue batch of messages for one consumber. PL/SQL MAQ interface only supports dequeue message one by one.
4. call store procedure *prc\_dequeue()* to dequeue message from queue table for dequeue only one message at one time; call *MAQDeqArray()* to dequeue message by batch through C/C++ API.
5. Query data from database using SQL statement. For the detail of the SQL, please refer to the description of store procedure *prc\_dequeue()*.
6. Start a new transaction and notate the received message for removing based on the dequeue options.
7. Commit after received the messages or rollback if error/exception encountered.

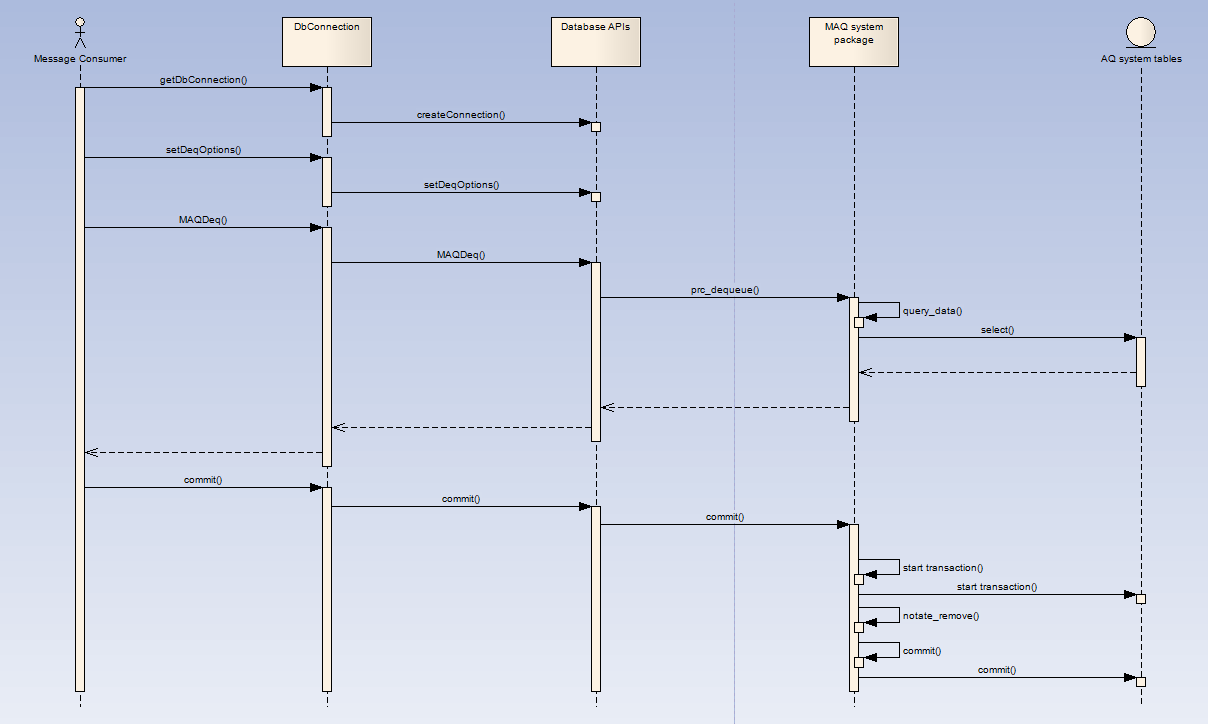


Figure 4: Dequeue seqence

### MAQ System Table Structure

This section describes the MAQ system tables which needed to support the implementation of MySQL Advance Queue. Details shown below:

1. For the MySQL Advance Queue System, each new created queue has the following tables to meet the queue requirements. Details shown below:

The following table is the main data table of the queue system, contains the queue message for further dequeue usage. The table structure shown below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Data Length | Nullable | Default value | Comment |
| Q\_NAME | VARCHAR | 30Bytes | Y | NULL | The name of the queue. |
| MSG\_SID | BIGINT | 8Bytes | N | 0 | The message sequence id. |
| MSGID | VARCHAR | 36Bytes | N | UUID() | The unique ID of the message. |
| PRIORITY | TINYINT | 1Bytes | Y |  | The priority of the message. |
| STATE | TINYINT | 1Bytes | N | 0 | 1 – ready; 2 – processing;  3 – remove delay; 4 - done |
| SUB\_CNT | TINYINT | 1Bytes | N | 0 | 0 – local; >0 --- others |
| EXPIRATION | DATETIME |  | Y |  | The message expiration time. |
| ENQ\_TIME | DATETIME |  | N | sysdate() | The enqueue time of the message. |
| ENQ\_UID | VARCHAR | 30Bytes | Y |  | The enqueue user of the message. |
| DEQ\_TIME | DATETIME |  | Y |  | The dequeue time of the message. |
| RETRY\_COUNT | INT | 4Bytes | Y |  | The retry count of the message. |
| SENDER\_MSGID | VARCHAR | 36Bytes | Y |  | The original ID of the message. |
| SENDER\_NAME | VARCHAR | 30Bytes | Y |  | The sender name of the message. |
| SENDER\_ADDR | VARCHAR | 20Bytes | Y |  | The address of the sender. |
| USER\_DATA\_1 | VARCHAR | 4000Bytes | Y |  | The message data for qualify |
| USER\_DATA\_2 | VARCHAR | 4000Bytes | Y |  | The message data for Oracle |
| USER\_DATA\_3 | VARCHAR | 4000Bytes | Y |  | The message data for MySQL |

Table 1: queue table

The following table is the dispatch table of the queue system, contains the message dispatch information. The table structure shown below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Data Length | Nullable | Default value | Comment |
| SUB\_ID | INT | 4Bytes | N |  | The id of the message subscriber. 0 – means the message is local message, not interpreted by MAQ propagation. |
| SUB\_NAME | VARCHAR | 30Bytes | N |  | The name of the subscriber. |
| MSG\_SID | BIGINT | 8Bytes | N | 0 | The sequence id. It’s same with the queue table. |
| MSGID | VARCHAR | 36Bytes | N |  | The unique ID of the message. |
| STATE | TINYINT | 1Bytes | N |  | The unique ID of the message. |

Table 2: dispatch table

The following table is the subscriber table of the queue system, contains all the recipient information of the queue, currently the maxmum number of recipients of one queue is 512. The table structure shown below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Data Length | Nullable | Default value | Comment |
| SUB\_ID | INT | 4Bytes | N |  | The id of the message subscriber. 0 – means the message is local message, not interpreted by MAQ propagation. |
| Q\_NAME | VARCHAR | 30Bytes | N |  | The name of the queue for the subscriber. |
| SUB\_NAME | VARCHAR | 80Bytes | Y |  | The name of the subscriber. |
| SUB\_ADDRESS | VARCHAR | 128Bytes | Y | Null | The address of the subscriber. Address should be unique. |

Table 3: subscriber table

1. For better management of the MySQL Advance Queue, three management tables are design to manage all queues in the database system. Details shown below:

The following table is the exception queue table for storage of messages which encountered errors or exceptions during propagation. The mechanism of handling exception messages are being described in the design of propagation. The structure of exception queue table shown below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Data Length | Nullable | Default value | Comment |
| Q\_NAME | VARCHAR | 30Bytes | Y | NULL | The name of the queue. |
| MSG\_SID | INT | 8Bytes | N | 0 | The message sequence id. |
| MSGID | VARCHAR | 36Bytes | N |  | The unique ID of the message. |
| PRIORITY | TINYINT | 1Bytes | Y |  | The priority of the message. |
| EXPIRATION | DATETIME |  | Y |  | The message expiration time. |
| ENQ\_TIME | DATETIME |  | N | sysdate() | The enqueue time of the message. |
| ENQ\_UID | VARCHAR | 30Bytes | Y |  | The enqueue user of the message. |
| DEQ\_TIME | DATETIME |  | Y |  | The dequeue time of the message. |
| RETRY\_COUNT | INT | 4Bytes | Y |  | The retry count of the message. |
| SENDER\_MSGID | VARCHAR | 36Bytes | Y |  | The original ID of the message. |
| SENDER\_NAME | VARCHAR | 30Bytes | Y |  | The sender name of the message. |
| SENDER\_ADDR | VARCHAR | 20Bytes | Y |  | The address of the sender. |
| USER\_DATA\_1 | VARCHAR | 4000Bytes | Y |  | The message data for qualify. |
| USER\_DATA\_2 | VARCHAR | 4000Bytes | Y |  | The message data for Oracle. |
| USER\_DATA\_3 | VARCHAR | 4000Bytes | Y |  | The message data for MySQL. |

Table 4: AQ\_SYS\_QUEUE\_E

The following table is the management table for MAQ system, contains all the queue list of the database system and the status of the each queue. The structure of the table shown below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Data Length | Nullable | Default value | Comment |
| PKEY | INT | 4Bytes | N | 0 | The primary key of the record. |
| Q\_NAME | VARCHAR | 30Bytes | N |  | The queue name. |
| Q\_TYPE | TINYINT | 1Bytes | N | 0 | Type of the queue.  0 – multi-consumer; 1 -- single |
| SUB\_COUNT | INT | 4Bytes | N | 0 | The number of subscribers of the MAQ system. |
| ENQ\_STATE | TINYINT | 1Bytes | N | 0 | The state of the en-queue of the queue. 0 – enable 1 -- disable |
| DEQ\_STATE | TINYINT | 1 Bytes | N | 0 | The state of the de-queue of the queue. 0 – enable 1 -- disable. |
| PAYLOAD\_TYPE | TINYINT | 1 Bytes | N | 0 | The payload type for the queue.  0 – object type; 1 – raw type |
| RETENTION | INT | 4Bytes | Y | 0 | The time of message which remain in queue when done. |
| REMARK | VARCHAR | 500Bytes | Y | Null | Useful comments of the queue. |
| ENQ\_MODE | INT | 4Bytes | N | 1 | The enqueue mode of the queue.  1. AQ\_API  2. SQL\_PRC\_NoPrepare  3. SQL\_PRC\_PrepareOne  4. SQL\_PRC\_PrepareBatch |

Table 5: AQ\_SYS\_QUEUE\_M

The following table is the seqence table for MAQ system to generate seqence IDs, contains all the sequence identifier and sequence value of the database system. The structure of the table shown below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Column Name | Data Type | Data Length | Nullable | Default value | Comment |
| PKEY | INT | 4Bytes | N | 0 | The primary key of the record. |
| NAME | VARCHAR | 30Bytes | N |  | The sequence identifier name. |
| INIT\_NUMBER | INT | 4Bytes | N | 0 | The initial value of the sequence. |
| LAST\_NUMBER | INT | 4Bytes | N | 0 | The last generated sequence key. |
| MAX\_NUMBER | INT | 4Bytes | N | 0 | The max value of the sequence. |
| INCREMENT\_BY | INT | 4Bytes | N | 1 | The sequence increase step. |
| CYCLE\_FLAG | VARCHAR | 1Bytes | N | Y | The flag for rotate of the value. |

Table 6: ADM\_SEQUENCE

### Interface

This section lists all the interfaces of PL/SQL and C/C++ API. There are two kinds of interface in MAQ library system such as public and private. For PL/SQL store procedures, it uses the following format to depict:

* Purpose of the procedure
* Procedure prototype
* Parameter explanation
* Algorithm/routine

For C/C++ API, it uses the following format to depict:

* Class name
* Purpose of the function
* Function prototype
* Function parameter explanation
* Algorithm/routine

1. **PL/SQL interface**

**Public interface:**

***prc\_enqueue()***

**Purpose**

This call is used for a MySQL Advanced Queuing enqueue.

**Prototype**

PROCEDURE prc\_enqueue(

IN p\_queue VARCHAR(80),

IN p\_payload\_1 VARCHAR(4000),

IN p\_payload\_2 VARCHAR(4000),

IN p\_payload\_3 VARCHAR(4000),

IN p\_msgid VARCHAR(38),

IN p\_sub\_cnt TINYINT(1),

IN p\_recipient VARCHAR(4000) )

**Parameters**

**p\_queue(IN)**

the name of the queue which the message will be enqueue to.

**p\_payload\_1(IN)**

the data buffer of the message. Currently, the maxmum size of the buffer is 4000Bytes. This parameter is for the qualifier attribute.

**p\_payload\_2(IN)**

the data buffer of the message. Currently, the maxmum size of the buffer is 4000Bytes. This parameter is for the oracle SQL statement.

**p\_payload\_3(IN)**

the data buffer of the message. Currently, the maxmum size of the buffer is 4000Bytes. This parameter is for the MySQL SQL statement.

**p\_msgid(IN)**

the sender message id of the received message. The message can be received from remote location and the message id is generated by remote, however, if the message is generated by local, then leave this parameter to NULL.

**p\_sub\_cnt(IN)**

the tag to indicate the message is for local, or need propagate to other consumers by using the built-in propagation mechnism or consums directly by other consumers.

0 – the message is for local; !0 -- the message need propagate to others.

**p\_recipient(IN)**

the recipient lists of the message. One message can be consumed by one or more than one consumers. For message only need for local dequeue in which case leave the address to empty; for message need to propagate to recipients in which case the address should set. The rule of the recipient lists as below:

for only one recipient: *name,address*

for more than one recipients*: name1,address1;name2,address2;....nameN,addressN*

**Algorithm/Routine**

The routine of the enqueue as below:

1. Start a new transaction;
2. Check if the enqueue option of the queue enable or not from the queue management table, if not exists, goto step 8;
3. Check the validity of the message recipient list, if the recipient list is null, then goto step 8;
4. Generate the message sequence ID by using function *func\_seq\_get\_next\_number()*, a 64bit sequence ID is introduced for not easily reach the maxmum value and it serves as sequence of the message enqueue into the queue. Therefore enable the message can maintain as ASC or DESC order. For the detail of how to get the sequence ID, please refer to function *func\_seq\_get\_next\_number().*
5. Get the message UUID by using MySQL system function *UUID()*.
6. Invoke the *prc\_enq\_others()* to insert data into dispatch and subscriber AQ system tables based on the recipient list. For single-consumer queue, if producer voilent the rule, then error will be raised to the caller;
7. Insert the message into the queue table;
8. Commit all the above operations or rollback the operations errors/exceptions occurred and raise the error to the caller.

***prc\_dequeue()***

**Purpose**

This call is used for a MySQL Advanced Queuing dequeue.

**Prototype**

PROCEDURE prc\_dequeue(

IN p\_queue VARCHAR(80),

IN p\_consumer VARCHAR(80),

IN p\_deqmode TINYINT(1),

IN p\_deqvisibility TINYINT(1),

IN p\_deqnavigation TINYINT(1),

IN/OUT p\_msgid VARCHAR(38),

OUT p\_payload VARCHAR(8050) )

**Parameters**

**p\_queue(IN)**

the name of the queue which the message will be dequeued from.

**p\_consumer(IN)**

the message consumer which specified to dequeue from the message queue.

**p\_deqmode(IN)**

the dequeue mode for the dequeue operation. Three modes being supported.

1 – remove immediate; 2 – remove delay; 3 – remove nodata

**p\_deqvisibility(IN)**

the dequeue visibility for the dequeue operation. Two options currently being supported.

1 – immediate; 2 – on commit

**p\_deqnavigation(IN)**

the message navigation for the dequeue operation. Two options currently being supported.

1 – first message; 2 – next message

**p\_msgid(IN/OUT)**

the message id which specified to dequeue from the queue. For IN parameter and not null, only the specified message id will be dequeue from the queue and the navigation dequeue options will be disabled automatically. If the parameter is null, then the out parameter is the message id which being dequeued from the queue based on the dequeue options.

**p\_payload(OUT)**

the output message data which being dequeued from the queue. The message length currently is less than 8000Bytes.

**Algorithm/Routine**

The routine of the dequeue as below:

1. Check the input parameter p\_msgid, whether is null or not, if null, then goto step 3;
2. Retrieve the message payload based only on the passed p\_msgid, then goto step 4;

*SELECT Q\_NAME, ENQ\_HID, ENQ\_LID, MSGID, PRIORITY,USER\_DATA FROM QUEUE\_TABLE WHERE MSGID = (SELECT MSGID FROM DISPATCH\_TABLE WHERE SUB\_NAME = UPPER(‘p\_consumer’) AND MSGID = ‘p\_msgid’ )*

1. Retrieve the message payload based on the dequeue navigation option;

*SELECT Q\_NAME, ENQ\_HID, ENQ\_LID, MSGID, PRIORITY,USER\_DATA FROM QUEUE\_TABLE WHERE MSGID =* *(SELECT MSGID FROM DISPATCH\_TABLE WHERE SUB\_NAME = UPPER(‘p\_consumer’)AND STATE = 1 ORDER BY ENQ\_HID, ENQ\_LID NAV\_OPT LIMIT 1*

1. Check the input parameter p\_deqvisibility, whether is immediate or not, if yes, then goto step 6;
2. Start a new transaction;
3. Check the input parameter p\_deqmode, if remove immediate, then goto step 9;
4. If remove nodata, then goto step 10;
5. Update the message state to delay remove based on the retrieved msgid, and invoke *prc\_removedelay\_check()* then goto step 11;

*UPDATE DISPATCH\_TABLE SET STATE = 3 WHERE MSGID = p\_msgid;*

*UPDATE QUEUE\_TABLE SET STATE = 3 WHERE MSGID = p\_msgid;*

1. Delete the retrieved message immediately, then goto step 11;

*DELETE FROM DISPATCH\_TABLE WHERE MSGID = p\_msgid;*

*DELETE FROM QUEUE\_TABLE WHERE MSGID = p\_msgid;*

1. Update the message state to done based on the retrieved msgid;

*UPDATE DISPATCH\_TABLE SET STATE = 4 WHERE MSGID = p\_msgid;*

*UPDATE QUEUE\_TABLE SET STATE = 4 WHERE MSGID = p\_msgid;*

1. Exit the procedure.

***prc\_create\_queue()***

**Purpose**

This call is used for a MySQL Advanced Queuing to create a new message queue.

**Prototype**

PROCEDURE prc\_create\_queue(

IN p\_queue VARCHAR(80),

IN p\_qtype INT(1),

IN p\_payload\_type TINYINT(1),

IN p\_retention INT(5)

)

**Parameters**

**p\_queue (IN)**

the name of the queue which will be created in the MAQ system.

**p\_qtype(IN)**

the type of the queue wihch will be created.

0 --- multiple-consumer queue; 1 --- single consumer queue

**Algorithm/Routine**

The routine of the create queue as below:

1. Create the queue table, the table name is add subfix *\_T* based on the queue name.
2. Create the dispatch table, the table name is add prefix *AQ\_* & subfix *\_I* based on the queue table.
3. Create the subscriber table, the table name is add prefix *AQ\_* & subfix *\_S* based on the queue table.
4. Add one entry to the queue management table *AQ\_SYS\_QUEUE\_M.*
5. Add one entries to the sequence table *ADM\_SEQUENCE* for sequence ID for the new created queue. The identifier are add prefix *AQ\_* & subfix *\_SEQ* for message sequence based on the queue name.

***prc\_drop\_queue()***

**Purpose**

This call is used for a MySQL Advanced Queuing to drop an existing message queue.

**Prototype**

PROCEDURE prc\_drop\_queue(

IN p\_queue VARCHAR(80) )

**Parameters**

**p\_queue (IN)**

the name of the queue which will be removed from the MAQ system.

**Algorithm/Routine**

The routine of the drop queue as below:

1. Remove the two seqence IDs entries from the sequence table *ADM\_SEQUENCE;*
2. Remove the entry from the queue management table *AQ\_SYS\_QUEUE\_M;*
3. Drop the subscriber table;
4. Drop the dispatch table;
5. Drop the queue table**.**

***prc\_start\_queue()***

**Purpose**

This call is used for a MySQL Advanced Queuing to start specified queue.

**Prototype**

PROCEDURE prc\_start\_queue(

IN p\_queue VARCHAR(80),

IN p\_queue\_option TINYINT(1) )

**Parameters**

**p\_queue (IN)**

the name of the queue which will be enable its functionalities in the MAQ system.

**p\_queue\_option(IN)**

the functionalities option of the queue for starting.

0 – enable enqueue; 1—enable dequeue; 2 – enable both enqueue & dequeue.

**Algorithm/Routine**

The routine of the start queue as below:

1. Check whether the specified queue is exist or not in queue management table, if not, then goto step 3;
2. Update the enqueue & dequeue options based on the input parameters;
3. Exit the operation.

***prc\_stop\_queue()***

**Purpose**

This call is used for a MySQL Advanced Queuing to stop specified queue.

**Prototype**

PROCEDURE prc\_stop\_queue(

IN p\_queue VARCHAR(80),

IN p\_queue\_option TINYINT(1) )

**Parameters**

**p\_queue (IN)**

the name of the queue which will be disabled its functionalities in the MAQ system.

**p\_queue\_option(IN)**

the functionalities option of the queue for stopping.

0 – disable enqueue; 1—disable dequeue; 2 – disable both enqueue & dequeue.

**Algorithm/Routine**

The routine of the stop queue as below:

1. Check whether the specified queue is exist or not in queue management table, if not, then goto step 3;
2. Update the enqueue & dequeue options based on the input parameters;
3. Exit the operation.

***prc\_add\_subscriber()***

**Purpose**

This call is used for a MySQL Advanced Queuing to add subscriber for specified queue.

**Prototype**

PROCEDURE prc\_add\_subscriber(

IN p\_queue VARCHAR(80),

IN p\_sub\_name VARCHAR(80),

IN p\_sub\_addr VARCHAR(50) )

**Parameters**

**p\_queue (IN)**

the name of the queue which specified to add subscriber in.

**p\_sub\_name(IN)**

the name of the subscriber which will be added to specified queue**.**

**p\_sub\_addr(IN)**

the address of the subscriber which will be added to specified queue**.** The address format should be: *[schema]*[*.queue\_name@ip:port*](mailto:.queue_name@ip:port) which need propagate to consumer(s) by using the propagation mechanism.

**Algorithm/Routine**

The routine of the add subscriber as below:

1. Check whether the queue is exist or not, if not, then goto step 5;
2. Add the entry to subscriber table based on the input parameters;
3. Update the subscriber count value in the queue management table.
4. If the address existing, then raise unique volient error and exit.
5. Exit the operation.

**Private interface:**

***prc\_enq\_others()***

**Purpose**

This call is used for an internal MySQL Advanced Queuing to enqueue data to the dispatch & subscriber tables based on the input recipient list parameter.

**Prototype**

PROCEDURE prc\_enq\_others(

IN p\_queue VARCHAR(80),

IN p\_msgid VARCHAR(38),

IN p\_sid BIGINT(20) unsigned,

IN p\_recipient VARCHAR(4000) )

**Parameters**

**p\_queue (IN)**

the name of the queue which will be enqueue data in.

**p\_msgid(IN)**

the message id which generated by *prc\_enqueue()*, the id need to enqueue into dispatch table for further use to retrieve data from the queue table.

**p\_sid (IN)**

the message 64bit sequence ID which generated by function *func\_seq\_get\_next\_number()*.

**p\_recipient (IN)**

the recipient lists of the message. The format of the recipient lists shown below:

for only one recipient: *name,address*

for more than one recipients*: name1,address1;name2,address2;....nameN,addressN*

**Algorithm/Routine**

The routine of the enqueue other MAQ system tables as below:

1. Start loop the recipient lists;
2. Parse the recipient name & address pair from the input recipient list parameter;
3. Invoke *prc\_enq\_table()* to enqueue the entries to dispatch & subscriber table;
4. Check if reach the end of the recipient list, then leave the loop; otherwise goto step 2;
5. Exit the procedure.

***prc\_enq\_table()***

**Purpose**

This call is used for an internal MySQL Advanced Queuing to enqueue data to the dispatch & subscriber tables.

**Prototype**

PROCEDURE prc\_enq\_table(

IN p\_queue VARCHAR(80),

IN p\_msgid VARCHAR(38),

IN p\_rec\_name VARCHAR(80),

IN p\_rec\_addr VARCHAR(80),

IN p\_sid BIGINT(20) unsigned

)

**Parameters**

**p\_queue (IN)**

the name of the queue which will be enqueue data in.

**p\_msgid(IN)**

the message id which generated by *prc\_enqueue()*, the id need to enqueue into dispatch table for further use to retrieve data from the queue table.

**p\_rec\_name(IN)**

the recipient name of the message passed in by recipient list.

**p\_rec\_addr(IN)**

the recipient address of the message passed in by recipient list, the address can be null in which case the message will not be interpreted by the system propagation and can be dequeue by specified the consumer name; otherwise the message will be interpreted by the system propagation to propagate the message to the specified address. The address format is: QUEUENAME@IP:PORT.

**p\_sid (IN)**

the passed in message 64bit sequence ID which generated by function *func\_seq\_get\_next\_number()*.

**Algorithm/Routine**

The routine of the enqueue MAQ system tables as below:

1. If the address is null, set the subscriber id to 0 and then goto step 8;
2. Retrieve the subscriber id from the subscriber table based on the passed in address;
3. If subscriber id found from the subscriber table, then goto step 8;
4. Retrieve the count of subscribers of the queue from the management queue table;
5. If the count greater than 512, then set output to failed and goto step 9;
6. Insert a new entry into the subscriber table and get the new added subscriber id;
7. Update the queue management table for the subscriber count of the queue;
8. Insert a new entry into the dispatch table based on the subscriber id info and passed in parameters;
9. End of the procedure and return the result.

***func\_seq\_get\_next\_number()***

**Purpose**

This call is used for an internal MySQL Advanced Queuing to generate the next seqence ID.

**Prototype**

FUNCTION func\_seq\_get\_next\_number(

IN p\_name VARCHAR(80) )

RETURNS decimal(20,0)

**Parameters**

**p\_name(IN)**

the name of the sequence identifier which match the name of sequence table *ADM\_SEQUENCE*.

**Algorithm/Routine**

The routine of the get next sequence ID as below:

1. Retrieve the maxmum sequence ID, increment step, initial value and rotate flag from sequence table *ADM\_SEQUENCE* based on the passed in sequence identifier;
2. Update the last number to the newest generated seqence ID increased by increment step and set the return sequence ID to the new seqence ID by using system function *last\_insert\_id()*;
3. If the increment step greatter than 0 and the new sequence ID greater than the maxmum sequence ID, then goto step 5;
4. Return the new generated sequence ID;
5. If the rotate flag is ‘N’, then raise error and goto step 7;
6. Update the last number to its initial value based on the passed in sequence identifier and return the initial value;
7. Exit the function.

***func\_seq\_get\_curr\_number()***

**Purpose**

This call is used for an internal MySQL Advanced Queuing to retrieve the current seqence ID.

**Prototype**

FUNCTION func\_seq\_get\_curr\_number(

IN p\_name VARCHAR(80) )

RETURNS decimal(20,0)

**Parameters**

**p\_name(IN)**

the name of the sequence identifier which match the name of sequence table *ADM\_SEQUENCE*.

**Algorithm/Routine**

The routine of the get current sequence ID as below:

1. Retrieve the last number from sequence table *ADM\_SEQUENCE* based on the passed in sequence identifier;
2. Return the last generated sequence ID.
3. **C/C++ APIs**

Five classes are being introduced to accomplish the C/C++ APIs for MySQL Advance Queue listed below:

*MESSAGE* – the object to hold the message which traveral through the network

*PRODUCER –* the object for message producer to generate message

*CONSUMER* – the object for message consumer/subscriber to retrieve message

*AGENT –* the object to hold the information for message producer/consumer

*LISTENER –* the object to provide the capability to monitor sepecified message from AQ.

### Class Diagram

The interaction of the MAQ classes shown below:

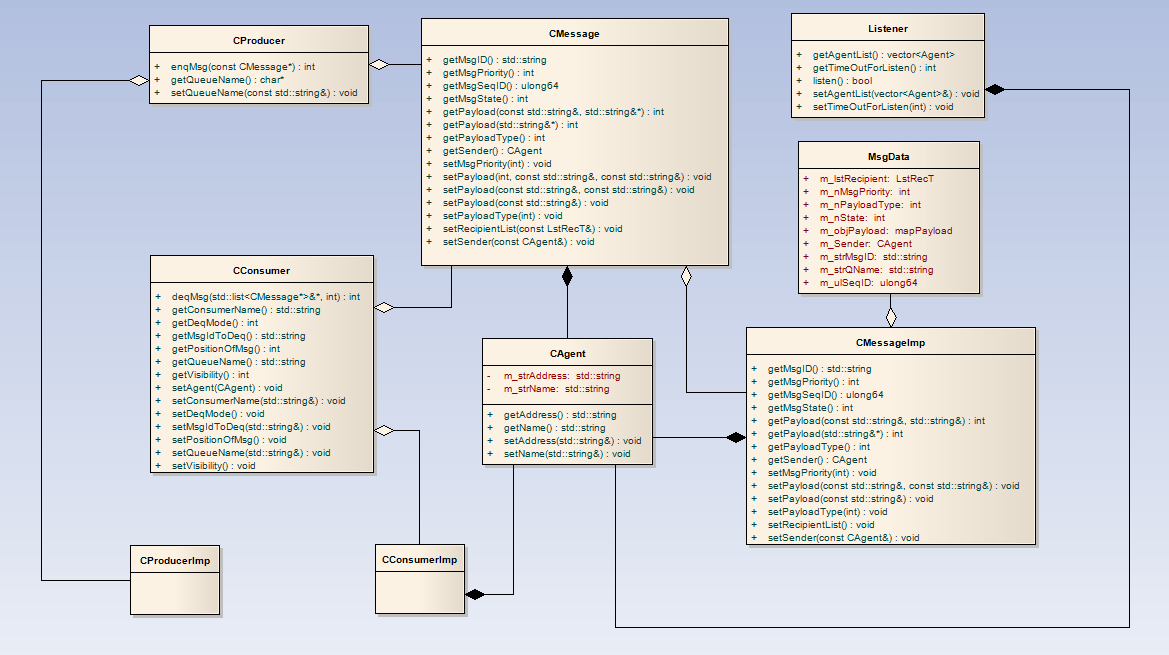


Figure 5: class diagram of AQ library

### Componment Diagram

In the architecture view of the MAQ library, it shown that there were four layers in this library. The detail of each layer lists below:

**AQ Facade:** Encapsulated the main interfaces of AQ such as enqueue, dequeue and listen from the queue.

**DB Facade:** Encapsulated the connection to various databases such as MySQL and Oracle.

**MAQCmd:** Encapsulated functions to enable the MySQL client library features the AQ funcationalities.

**mcl:** the open source, third party MySQL client connection library provided by MySQL.

**ocilib:** the open source, third party Oracle client connection library provided by open source community, the library encapsulated the Oracle C library OCI and contains AQ interfaces.

**MySQL Database Package:** the procedures which enable the MySQL features AQ functionalities. Contains two kinds of procedures: AQ functional & adminstration.

**Oracle Database Package:** the built-in Oracle AQ packages.

The componment diagram of the MAQ library shown below:

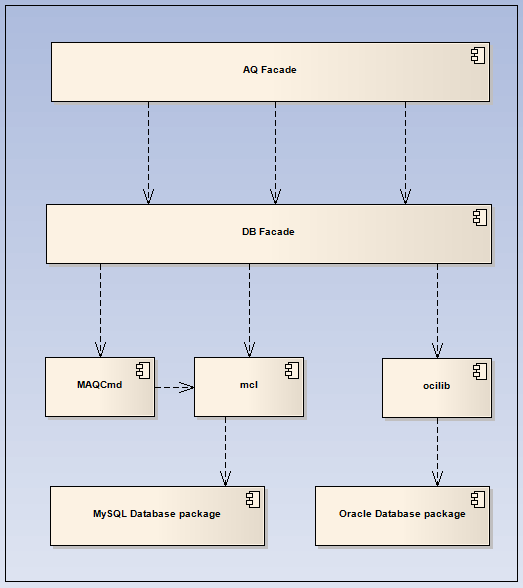


Figure 6: componment digram of AQ library

### Dependencies

The dependence of the AQ library shown below:

mcl --- the MySQL client connection library.

ocilib --- the Oracle client connection library.

### Configuration Information

## Testability

## Maintainability

## Impact Analysis

## Deployment Considerations

## Flow-on Effects

## Extensibility and Reuse

# gui-component design information

# Appendix A: Software module design review

| **DTL (C955) Design Module Review** | | | **Date:** | | |
| --- | --- | --- | --- | --- | --- |
| **Doc. No.:** | | | | | |
| **Subject :** | | | | | |
| **Reviewer** : | | | | | |
| **Status** : | | | | | |
|  |  |  | | |  |
| **Status :** | 1.Draft | 2. Conditional Accepted | | 3. Accepted | 4. Rejected |

| **No** | **Ref** | **Page No** | **Comments** | **Response** |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

Review Outcome (completed immediately following the review)

| **Signature of Reviewer**  **Name** |  | **Date:** |
| --- | --- | --- |
| **Signature of Independent Verifier**  **Name** |  | **Date:** |

Review Closeout (completed when review actions are completed)

| **All actions completed and closed** | **✓/🗶** | |
| --- | --- | --- |
| **Signature**  **Name** |  | **Date:** |