Queue Usage Types

In **IBM MQ**, queues can have a **usage type** of either **NORMAL** or **TRANSMISSION (XMITQ)**. This setting affects how the queue is used within the messaging system.

**🟢 NORMAL Queue**

* **Purpose**: Used for standard application messaging.
* **Used By**: Applications that put and get messages.
* **Behavior**:
  + Applications send messages directly to a NORMAL queue.
  + Consumers or listeners retrieve messages from these queues.
  + Example: APPLICATION.QUEUE, INVENTORY.UPDATES, etc.

**🟡 TRANSMISSION Queue (XMITQ)**

* **Purpose**: Used internally by IBM MQ for **inter-queue manager communication** (i.e., **message routing between different queue managers**).
* **Used By**: MQ channel processes (like sender channels).
* **Behavior**:
  + Messages sent to a **remote queue** are actually placed on the associated XMITQ.
  + A **sender channel** then picks up the message from the XMITQ and transmits it to the remote queue manager.
  + Example: QM2.XMITQ for routing messages to QM2.

**Summary Table**

| **Feature** | **NORMAL Queue** | **TRANSMISSION Queue (XMITQ)** |
| --- | --- | --- |
| Used by | Applications | MQ channels |
| Message flow | App → Queue → App | App → XMITQ → Channel → Remote QM |
| Holds messages for | Local consumption | Remote delivery |
| Accessed by | MQI API calls by apps | Channel processes |
| Example name | ORDER.INPUT | QM2.XMITQ |

**Practical Example**

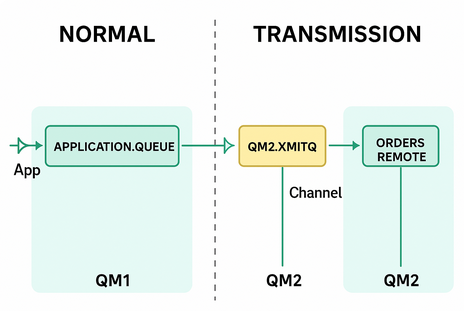
Suppose you have two queue managers:

* QM1 (local)
* QM2 (remote)

You want to send a message from an app connected to QM1 to a remote queue ORDERS.REMOTE on QM2.

1. You define a **remote queue** ORDERS.REMOTE on QM1 that points to QM2.
2. That remote queue is configured to use a **transmission queue** like QM2.XMITQ.
3. When the app puts a message to ORDERS.REMOTE, MQ puts the message onto QM2.XMITQ.
4. The **sender channel** on QM1 sends the message to QM2, which delivers it to the real ORDERS.REMOTE queue.

**Diagram**



In **IBM MQ**, several core objects form the backbone of the messaging infrastructure. Understanding these objects, their key properties, and how they are used in typical **banking topologies** is essential for designing reliable and secure systems.

**🔧 Main IBM MQ Objects and Their Key Properties**

**1. Queue Manager**

* **Definition**: The central MQ server process that manages queues and messaging.
* **Key Properties**:
  + QMNAME: Name of the queue manager
  + LISTENER: Port and protocol settings
  + CHANNELS: Channel definitions for communication
  + LOG / STORAGE: Logging and persistence configurations

**2. Queues**

* **Definition**: Objects that store messages.
* **Types**:
  + **Local Queue**: Stores messages for retrieval by applications.
  + **Remote Queue**: A definition that routes messages to another queue manager.
  + **Alias Queue**: Points to another queue (local or remote).
  + **Transmission Queue (XMITQ)**: Used to stage messages for delivery to remote queue managers.
* **Key Properties**:
  + MAXDEPTH: Maximum number of messages
  + MAXMSGL: Maximum message length
  + DEFPSIST: Default persistence (yes/no)
  + USAGE: NORMAL or XMITQ
  + GET, PUT: Permissions

**3. Channels**

* **Definition**: Communication paths between queue managers or between applications and queue managers.
* **Types**:
  + **Sender/Receiver (SND/RCVR)**: Used for QM-to-QM communication
  + **Server Connection (SVRCONN)**: For client-to-QM communication
  + **Requester/Server**: Less common, for asymmetric comms
  + **Cluster Sender/Receiver**: Used in clustered setups
* **Key Properties**:
  + XMITQ: Transmission queue name
  + TRPTYPE: Protocol (e.g., TCP)
  + CONNAME: Host:port of the remote endpoint
  + SSLCIPH: SSL/TLS settings
  + MCAUSER: Channel user identity (used for authorization)

**4. Listeners**

* **Definition**: Network listeners that wait for incoming connections on a specific port.
* **Key Properties**:
  + TRPTYPE: Protocol (e.g., TCP)
  + PORT: Listening port
  + IPADDR: Optional IP binding

**5. Process Definitions**

* **Definition**: Metadata used in triggering to start external applications.
* **Key Properties**:
  + APPLTYPE: Application type (e.g., Windows, UNIX)
  + APPLICID: Command to execute

**6. Triggers**

* **Definition**: Mechanisms to initiate actions when messages arrive on queues.
* **Key Properties**:
  + TRIGTYPE: Type (e.g., FIRST, EVERY)
  + INITQ: Initiation queue
  + PROCESSNAME: Linked process definition

**7. Security Objects**

* **AUTHREC (Authority Records)**: Control what users/groups can do with MQ objects.
* **SSL/TLS Configs**: Cipher specs, keystores, and certificates.

**🏦 Common IBM MQ Topologies in Banking**

**1. Hub-and-Spoke (Star)**

* **Use Case**: Central queue manager (hub) routes messages to multiple branch systems.
* **Example**: Core banking system at HQ communicates with branch applications.

**2. Point-to-Point**

* **Use Case**: Direct connection between two queue managers.
* **Example**: ATM system communicates directly with the fraud detection engine.

**3. Store-and-Forward**

* **Use Case**: Intermediary MQ nodes store messages temporarily.
* **Example**: Payment gateway → DMZ MQ node → Core processing MQ

**4. Clustered MQ**

* **Use Case**: Scalability and high availability using queue manager clusters.
* **Example**: Load-balanced processing of SWIFT messages across multiple servers.

**5. Gateway/DMZ Design**

* **Use Case**: Securely relay external messages to internal MQ systems.
* **Example**: Secure file transfers or external partner integration via MQ.

**🛡️ Banking-Specific Considerations**

* **High Security**: TLS channels, MCAUSER hardening, object-level authorization.
* **High Availability**: MQ clusters, HA queue managers (e.g., using multi-instance or queue sharing groups).
* **Regulatory Compliance**: Logging, audit trails, message persistence.
* **Transactionality**: Use of **persistent messages**, **XA transactions**, and **syncpoint control** for consistency.

A **template for defining IBM MQ objects** is a reusable set of parameters and structure that helps standardize the creation of MQ objects—like queues, channels, listeners, etc.—across environments (e.g., dev, test, prod).

Templates make deployments **repeatable, version-controlled, and easier to audit**, especially important in enterprise settings like banking.

**🧩 MQ Object Definition Template – Overview**

IBM MQ objects are typically defined using:

* **MQSC scripts** (\*.mqsc)
* **PCF (Programmable Command Format)** commands (used by APIs/tools)
* **Infrastructure-as-Code tools** (e.g., Ansible, Terraform, or proprietary deployment frameworks)

For scripting and templating, **MQSC** is the most common. Here's how templates work.

**📄 MQSC Template Examples**

**1. Local Queue Template**

DEFINE QLOCAL('<<QUEUE\_NAME>>') +

DESCR('<<DESCRIPTION>>') +

MAXDEPTH(<<DEPTH>>) +

MAXMSGL(<<MSGLENGTH>>) +

DEFPSIST(<<PERSISTENCE>>) +

DEFSOPT(SHARED) +

PUT(ENABLED) +

GET(ENABLED) +

USAGE(NORMAL)

🔁 **Example:**

DEFINE QLOCAL('PAYMENTS.INCOMING') +

DESCR('Incoming payment requests') +

MAXDEPTH(10000) +

MAXMSGL(10485760) +

DEFPSIST(YES) +

PUT(ENABLED) +

GET(ENABLED) +

USAGE(NORMAL)

**2. Remote Queue Template**

DEFINE QREMOTE('<<ALIAS\_NAME>>') +

RNAME('<<TARGET\_QUEUE>>') +

RQMNAME('<<REMOTE\_QMGR>>') +

XMITQ('<<TRANSMISSION\_QUEUE>>') +

DESCR('<<DESCRIPTION>>')

**3. Channel Template (Sender)**

DEFINE CHANNEL('<<CHANNEL\_NAME>>') +

CHLTYPE(SDR) +

TRPTYPE(TCP) +

CONNAME('<<REMOTE\_HOST>>(<<PORT>>)') +

XMITQ('<<XMITQ\_NAME>>') +

SSLCIPH('<<CIPHER\_SUITE>>') +

MCAUSER('<<MCA\_USER>>') +

DESCR('<<DESCRIPTION>>')

**4. Listener Template**

DEFINE LISTENER('<<LISTENER\_NAME>>') +

TRPTYPE(TCP) +

PORT(<<PORT>>) +

IPADDR('<<BIND\_IP>>') +

CONTROL(QMGR)

**5. Trigger Template**

DEFINE QLOCAL('<<TRIGGERED.QUEUE>>') +

TRIGGER +

TRIGTYPE(FIRST) +

TRIGDPTH(1) +

TRIGMPRI(0) +

PROCESS('<<PROCESS\_NAME>>') +

INITQ('SYSTEM.DEFAULT.INITIATION.QUEUE')

**🏗️ How to Use These Templates**

You can integrate them into:

* **Manual scripts**: Replace placeholders manually or via sed/PowerShell
* **CI/CD pipelines**: Use templating engines like **Jinja2** (Ansible), **Helm** (Kubernetes), or **Terraform** variable files
* **Parameterization**: Store values in .properties or .yaml files

**✅ Best Practices**

* Use **naming conventions** (APP.ENV.OBJECTTYPE) for consistency
* Include + at line ends in MQSC for multiline continuity
* Track all object definitions in **version control (Git)**
* Use a **change management process** to update or delete objects
* Validate with runmqsc -n (syntax check without applying)