UNIVERSITÄT LEIPZIG

Enterprise Computing
Einführung in das Betriebssystem z/OS

Prof. Dr. Martin Bogdan
Prof. Dr.-Ing. Wilhelm G. Spruth

WS2012/2013

WebSphere MQ Teil 4

MQI API

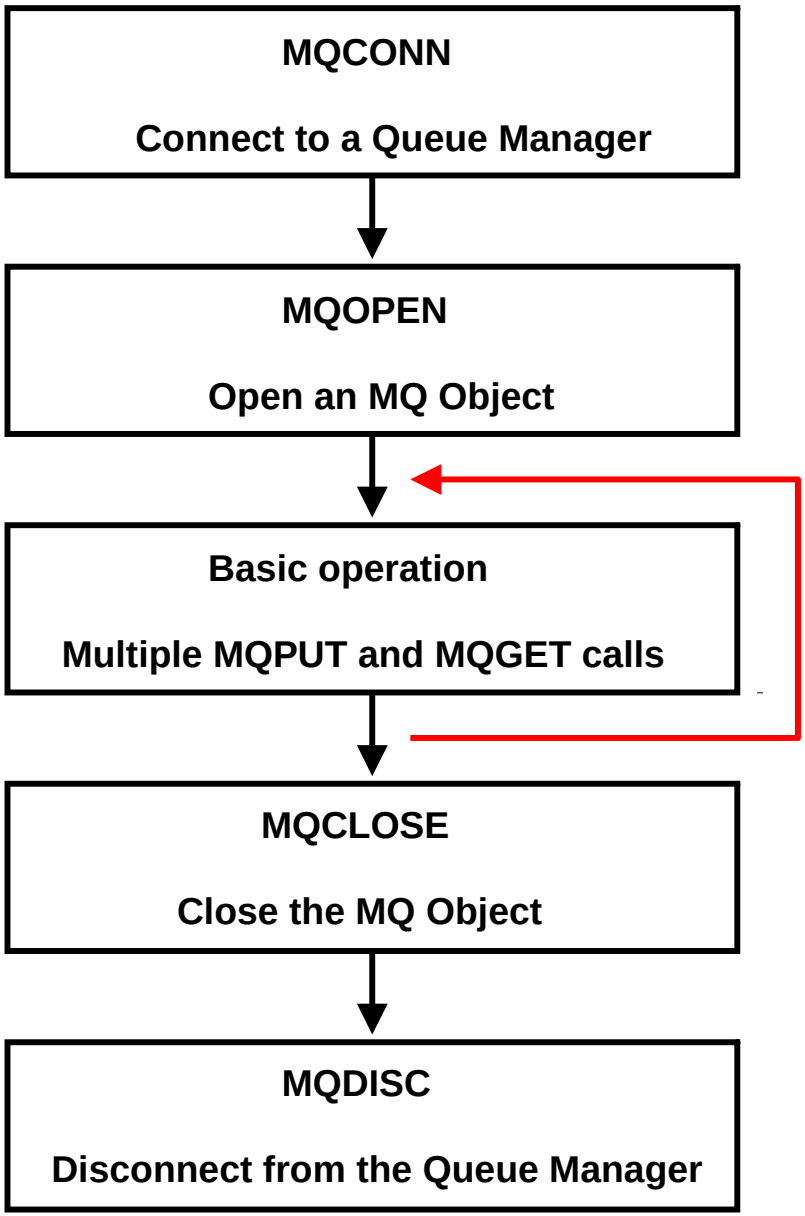
MQI Application Programming Interface

Die MQI API besteht aus 13 API Kommandos. Dies sind die 6 am häufigsten benutzten Kommandos:

MQCONN	**Verbindung mit einem (normalerweise entfernten) Queue Manager herstellen**
MQOPEN	**Öffnen (Open) einer spezifischen Queue**
MQPUT	**Eine Message in eine Queue schreiben**
MQGET	**Eine Message aus einer Queue auslesen**
MQCLOSE	**Eine Queue schließen (Close)**
MQDISC	**Verbindung mit einem Queue Manager auflösen**

Die übrigen 7 Kommandos sind:

MQPUT1	**Kombination von MQOPEN + MQPUT + MQCLOSE**
MQINQ	**Eigenschaften eines Objektes erfragen**
MQSET	**Eigenschaften (Properties) eines Objektes setzen**
MQCONNX	**Standard oder fast Path Bindings**
MQBEGIN	**Eine Unit of Work starten (database coordination)**
MQCMIT	**Commit eine Unit of Work**
MQBACK	**Back out**



Ein Anwendungsprogramm benutzt verschiedene WebSphere MQ MQI Kommandos.

Es beginnt mit den MQCON und MQOPEN Kommandos. In der Regel beziehen diese sich auf einen entfernten QUEUE-Manager und eine entfernte Target-Queue. Bedenken Sie, dass der Remote Queue Manager mehrere Target Queues enthalten kann. Mit MQOPEN wählen wir die richtige Target Queue.

Das Anwendungsprogramm führt dann seinen Code aus, in der Regel durch den Einsatz mehrerer MQPUT und/oder MQGET Anrufe.

Wenn es damit fertig ist, beendet es seine Arbeit durch Ausgabe von MQCLOSE und MQDISC Kommandos.

MQI Call Parameters

Es gibt zwei Typen von Parametern, die von allen Kommandos benutzt werden:

Handles

Diese werden von den Queue-Manager MQCONN und MQOPEN Kommandos zurückgegeben, und werden dann als Eingangsgrößen für die nachfolgenden MQPUT und MQGET Kommandos verwendet.

Der Begriff „Handle“ wird hier benutzt, um eine Message Channel Verbindung oder eine Target Queue eindeutig zu definieren.

Return codes

Zwei Return-Codes sind für alle Kommandos gebräuchlich: ein Completion-Code und ein Reason-Code.

Der Completion-Code gibt an, ob die Kommandoausführung erfolgreich war (mit einem MQCC_OK), oder ob ein Fehler aufgetreten ist (mit einem MQCC_FAILED).

Der Reason-Code ist MQCC_NONE, wenn der Comletion-Code MQCC_OK ist. Wenn nicht, wird ein anderer Wert zurückgegeben, der die Ursache der Warnung oder des Fehler im Completion-Code erklärt.

Verbinden mit dem Queue Manager

Um eine MQSeries Aktivität mittels der MQI Schnittstelle zu starten, sollten Sie sich zunächst mit einem (in der Regel entfernten) QUEUE-Manager mit einem der beiden verfügbaren Verbindungs-Kommandos, MQCONN oder MQCONNX verbinden. Beispielsweise unter Verwendung von MQCONN:

```
MQCONN (QMgrName, Hconn, CompCode, Reason)
```

Als Input für MQCONN müssen wir den symbolischen Namen des (in der Regel entfernten) Queue-Managers (QMgrName) angeben.

Das Kommando gibt die folgenden Werte zurück:

- **Eine Connection Handle (Hconn) zu dem Queue-Manager. Hconn wird durch das Anwendungsprogramm in zukünftigen Kommandos wendet, um diesen besonderen Queue Manager zu adressieren.**
- **Die Ergebnis-Codes (Completion- und Reason-Codes).**

MQCONN bewirkt, dass der lokale Queue Manager eine Transmission Queue einrichtet, und ein Message Channel mit seinen beiden MCAs erstellt wird.

Opening MQSeries objects

Das MQOPEN Kommando ermöglicht es einer Anwendung, Nachrichten in eine Queue zu schreiben oder Nachrichten aus einer Queue zu lesen.

```
MQOPEN (Hconn, ObjDesc, Options, Hobj, CompCode, Reason)
```

Das Kommando hat diese Input Parameter:

- **Eine Connection Handle. Der Wert von Hconn war von dem vorangehenden MQCONN Kommando zurückgegeben worden.**
- **Eine Beschreibung der Target Queue, die wir öffnen (Open) möchten. Dies geschieht in der Form eines MQ Object Descriptor (MQOD). MQOD ist eine Structure (in C++) oder ein Copybook (in Cobol). Diese Struktur identifiziert die Target Queue, die geöffnet werden soll.**
- **Eine oder mehrere Optionen. Eine mögliche Option ist es z.B., der Anwendung zu erlauben, eine Nachricht in die Target Queue zu stellen.**

Das Kommando gibt die folgenden Werte zurück:

- **Eine Object Handle Hobj, die den Zugriff auf die Target Queue mittels ihres Namens spezifiziert.**
- **Eine modifizierte Object-Descriptor Structure (MQOD, wenn eine Abänderung erforderlich ist).**
- **Die Result-Codes (Completion- und Reason-Codes).**

Weiterführende Information finden Sie in:

- **MQ Application Programming Guide: http://publib.boulder.ibm.com/iseries/v5r2/ic2924/books/csqzal05.pdf**
- **MQSeries Primer: http://www.informatik.uni-leipzig.de/cs/Literature/Textbooks/MQSerPrimer.pdf**

Schreiben einer Nachricht in eine Queue

Wir benutzen das MQPUT Kommando, um eine Nachricht in eine Queue zu schreiben:

```
MQPUT (Hconn, Hobj, MsgDesc, PutMsgOpts, BufferLength, Buffer, CompCode, Reason)
```

Dieses Kommando erhält als Input Parameter:

- **Eine Connection Handle Hconn, die durch das MQCONN Kommando zurückgegeben wurde.**
- **Eine Queue Handle `Hobj` , die von dem MQOPEN Kommando zurückgegeben wurde.**
- **Eine Beschreibung der Nachricht, die Sie in die Queue stellen wollen, in der Form eines Message Descriptors.**
- **Control Informationen, in Form einer Put-Message Optionen (MQPMO) Struktur.**
- **Die Länge der Daten in dieser Nachricht.**
- **Die Nachricht selbst, enthalten in einem Puffer.**

Das Kommando gibt diese Werte zurück:

- **Die Result-Codes (Completion- und- Reason-Codes).**
- **Aktualisierte Message Descriptor und Optionen, wenn der Aufruf erfolgreich ausgeführt wurde.**

Lesen einer Nachricht aus einer Queue

Wir benutzen das MQGET Kommando, um Nachrichten aus einer Queue zu lesen:

```
MQGET (Hconn, Hobj, MsgDesc, GetMsgOpts, BufferLength, Buffer, DataLength,
CompCode, Reason)
```

Die Eingabeparameter für diesen Aufruf sind:

- **Eine Connection Handle `Hconn`.**
- **Eine Queue Handle `Hobj`.**
- **Eine Beschreibung der Nachricht, die wir aus der Queue lesen wollen, in der Form einer MQMD Structure.**
- **Control Information in Form einer Get Message Options (MQGMO) Struktur.**
- **Die Größe des Puffers, in dem die Nachricht gespeichert werden soll.**
- **Die Adresse des Puffers.**

Die Ausgabe Parameter dieses Aufrufs sind:

- **Die Result-Codes (Completion- und Reason-Codes).**
- **Die Message in dem angegeben Puffer, wenn das Kommando erfolgreich abgeschlossen wurde.**
- **Die Get Message Options Struktur, modifiziert, um den Namen der Queue zu zeigen, aus dem die Nachricht abgerufen wurde.**
- **Die Message Descriptor Struktur, mit Informationen über die eingelesene Nachricht.**
- **Die tatsächliche Länge der Nachricht.**

Closing the MQ object

Um ein MQ Object zu schließen (Close) benutzen wir das MQCLOSE Kommando.

```
MQCLOSE (Hconn, Hobj, Options, CompCode, Reason)
```

Dieses Kommando benutzt die folgenden Eingabe Parameter:

- **Eine Connection Handle `Hconn`.**
- **Eine Queue Handle `Hobj` des Remote-Queue-Objekts das wir schließen wollen wir.**
- **Die Close Optionen.**

Die folgenden Parameter werden zurückgegeben:

- **Die Result-Codes (Completion und Reason Codes).**
- **Die Object Handle `Hobj,` auf den Wert MQHO_UNUSABLE_HOBJ zurückgesetzt.**

Typischerweise wird eine Queue gelöscht, sobald das Programm, das sie geschaffen hat, ein MQCLOSE Kommando für diese Queue ausführt. In diesem Fall wird die Close Option MQCO_NONE erzeugt.

Das folgende Codefragment zeigt die APIs, um eine Nachricht in eine Queue zu schreiben und eine Antwort von einem anderen Queue zu erhalten. Die C++ MQI wird benutzt.

Das Codefragment verwendet die folgenden Definitionen. Die Felder CompCode und Reason enthalten die Fertigstellung Codes für die APIs. Wenn Sie interessiert sind, können Sie sie in der Application Programming Reference Dokumentation finden:
http://publib.boulder.ibm.com/iseries/v5r2/ic2924/books/csqzak05.pdf

```
                                // Definitions
MQHCONN HCon;                   // Connection handle
MQHOBJ HObj1;                   // Object handle for queue 1
MQHOBJ HObj2;                   // Object handle for queue 2
MQLONG CompCode, Reason;        // Return codes
MQLONG options;
MQOD od1 =  {MQOD_DEFAULT};     // Object descriptor for queue 1
MQOD od2 =  {MQOD_DEFAULT};     // Object descriptor for queue 2
MQMD md =   {MQMD_DEFAULT};     // Message descriptor
MQPMO pmo = {MQPMO_DEFAULT};    // Put message options
MQGMO gmo = {MQPMO_DEFAULT};    // Get message options
:
```

```
// 1 Connect application to a queue manager.
strcpy (QMName,"MYQMGR");
MQCONN (QMName, &HCon, &CompCode, &Reason);
// 2 Open a queue for output
strcpy (od1.ObjectName,"QUEUE1");
MQOPEN (HCon,&od1, MQOO_OUTPUT, &Hobj1, &CompCode, &Reason);
// 3 Put a message on the queue
MQPUT (HCon, Hobj1, &md, &pmo, 100, &buffer, &CompCode, &Reason);
// 4 Close the output queue
MQCLOSE (HCon, &Hobj1, MQCO_NONE, &CompCode, &Reason);
// 5 Open input queue
options = MQOO_INPUT_AS_Q_DEF;
strcpy (od2.ObjectName, "QUEUE2");
MQOPEN (HCon, &od2, options, &Hobj2, &CompCode, &Reason);
// 6 Get message
gmo.Options = MQGMO_NO_WAIT;
buflen = sizeof(buffer - 1);
memcpy (md.MsgId, MQMI_NONE, sizeof(md.MsgId);
memset (md.CorrelId, 0x00, sizeof(MQBYTE24));
MQGET (HCon, Hobj2, &md, &gmo, buflen, buffer, 100, &CompCode, &Reason);
// 7 Close the input queue
options = 0;
MQCLOSE (HCon, &Hobj2,options, &CompCode, &Reason);
// 8 Disconnect from queue manager
MQDISC (HCon, &CompCode, &Reason);
```

Kommentar:

1 Dieses Statement verbindet die Anwendung mit dem dem QUEUE-Manager mit dem Namen MYQMGR. Wenn der Parameter QMName keinen Namen enthält, dann wird der Default-QUEUE-Manager verwendet. MQ speichert die Handle des Queue-Managers in der Variablen HCon. Diese Handle muss in allen nachfolgenden APIs genutzt werden.
2 Um eine Queue zu öffnen muss der Name der Queue in den Objektdeskriptor kopiert werden, der für diese Queue verwendet wird. Dieses Kommando öffnet QUEUE1 nur für Output (Open Option MQOO_OUTPUT). Die Handle der Queue und Werte in dem Objektdeskriptor werden zurückgegeben. Die Handle Hobj1 muss in dem MQPUT Kommando angegeben werden.
3 MQPUT platziert die Nachricht, die in einem Puffer steht, in eine Queue. Parameter für MQPUT sind:

- **Die Handle des Queue-Manager (aus MQCONN)**
- **Die Handle der Queue (von MQOPEN)**
- **Den Message Descriptor**
- **Eine Struktur, die Optionen für den MQPUT Befehl enthält (siehe MQ Application Programming Reference)**
- **Die Länge der Nachricht**
- **Der Puffer, der die Daten enthält**

4 Dieses Kommando schließt die Output Queue. Da die Queue vordefiniert ist, findet kein Close Processing statt (MQOC_NONE).
5 Dieses Kommando öffnet Queue2 nur für die Eingabe, unter Benutzung der Queue-Voreinstellungen (Defaults).
6 Für das Get Kommando wird die nowait Option verwendet. MQGET braucht die Länge des Puffers als Eingangs Parameter. Da keine Nachrichten-ID oder Korrelations-ID angegeben ist, wird die erste Nachricht aus der Queue gelesen. Sie können einen Warteintervall (in Millisekunden) hier spezifizieren. Sie können den Return Code überprüfen, um herauszufinden, ob die Zeit abgelaufen ist und keine Nachricht eingetroffen ist.
7 Dieses Kommando schließt die Eingabe-Queue.
8 Die Anwendung schließt die Verbindung mit dem Queue-Manager.

Die gezeigten Codebeispiele benutzen C++ und die MQI API. Sie würden sehr ähnlich mit anderen Programmiersprachen wie Cobol oder PL/1.aussehen

Java MQ-Code sieht anders aus, weil der JEE (Java Enterprise Edition) Standard für JMS (Java Message Service) eine eigene API festlegt, die sich von der MQI API unterscheidet. Java-Anwendungen verwenden normalerweise die JMS API anstelle der MQI API.

Es gibt 2 Versionen von WebSphere MQ. Die "non-integrated"-Version benötigt keinen WebSphere Java Application Server (WAS), und wird in der Regel für andere Programmiersprachen als Java verwendet. Die integrierte Version ist Teil der WebSphere Java Application Server (WAS), nutzt aber die gleiche Code-Basis wie die nicht-integrierte Version.

MQ Code Beispiele für Cobol, C + + und Java sind verfügbar unter;

http://www.informatik.uni-leipzig.de/cs/Literature/Textbooks/MQ/index.html

Wenn Sie glauben, all dies ist kompliziert: Die meisten Unternehmen stehen vor dem Problem, bestehende Anwendungs-Software, die von unabhängiger Seite in der Vergangenheit entwickelt wurde, in verschiedenen Programmiersprachen implementiert wurde, und auf verschiedenen Hardware-, Betriebssystem-und Middleware-Plattformen läuft, zu integrieren. Viele Experten betrachten die Verwendung von WebSphere MQ als den einfachsten und effizientesten Weg, um die Integration von Anwendungen zu erreichen.